Table des matières

1 Introduction

1.1 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.

1.2 Architecture d'un ordinateur

1.2.1 Matériel

En première approche, on peut considérer qu'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 ...

1.2.2 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, ...

1.2.3 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.

1.3 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.

1.4 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.

1.4.1 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.

1.4.2 Codage des caractères

Les caractères doivent 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).

Par exemple, avec ce code (voir figure 1, 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

1.5 Rappels sur la numération

Le système de numération utilisé habituellement est le système décimal. Un ordinateur étant basé sur le système binaire, il peut être 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 :

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

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

Cette décomposition est unique.
On note généralement :

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

1.5.2 Système décimal

Dans le cas du système décimal :

- la 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 :

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

1.5.3 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 $0 \leq ai \leq 1$

Exemple :

$$1010,101 = 1x2^3 + 0x2^2 + 1x2^1 + 0x2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3}$$
$$= 8 + 2 + 0,5 + 0,125$$
$$= 10,625 \text{ en base } 10$$
$$= 10,625_{10}$$

Exercice : 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

1.5.4 Le système octal

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

1.5.5 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)

1.6 Conversions

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

1.6.1 Conversion base b – système décimal

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

Exemple :

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

$= 8 + 2 + 0{,}5 + 0{,}125$

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

1.6.2 Conversion système décimal – base b :

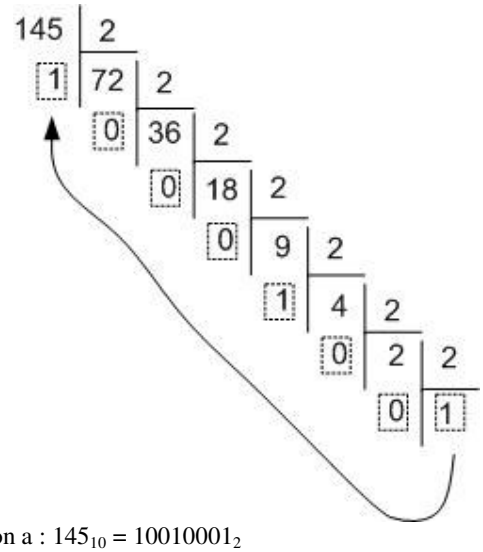
On applique le principe de la division euclidienne :

$n = b * q + r$ 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.

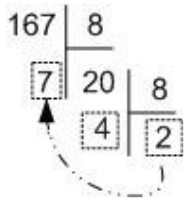
La liste **inversée** des restes ainsi obtenus constitue la décomposition recherchée.

Exemple 1 : Décimal --> Binaire



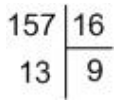
Ainsi, on a : $145_{10} = 10010001_2$

Exemple 2 : Décimal --> Octal



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

Exemple 3 : Décimal --> hexadécimal



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

1.6.3 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, et 8 = 23 => 3 bits.

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

1.6.4 Conversion Hexadécimal <--> Binaire

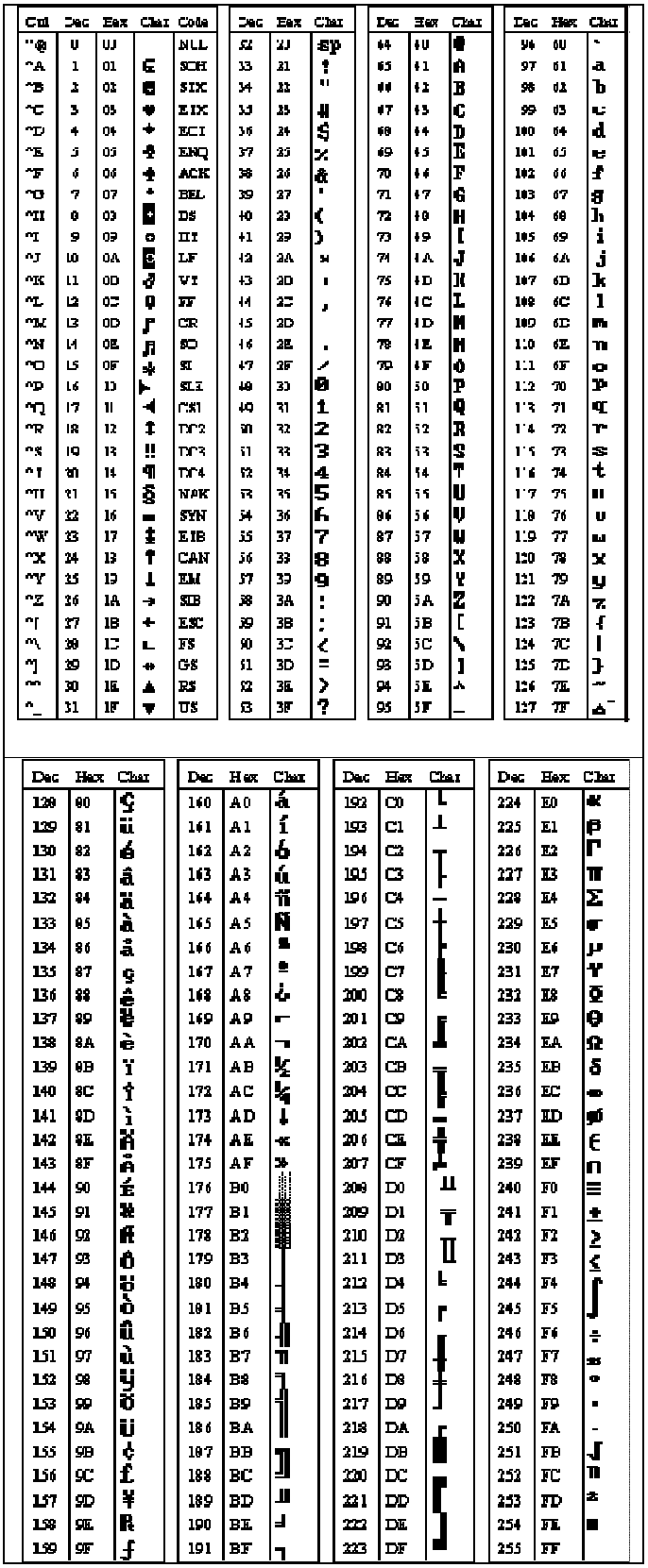
Cette fois-ci, chaque chiffre du nombre en base hexadécimale peut être représenté par un nombre de 4 chiffres en binaire. On complète à gauche par des zéros si nécessaire.
On regroupe les bits par 4, car nous sommes en base 16,
et $16 = 2^4$ => 4 bits.
Exemples :
$B5E_{16}$ --> 1011 0101 1110
1100 0111 --> $C7_{16}$

1.6.5 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).

Figure 1: Table ASCII



Ctl	Dec	Hex	Char	Code
^@	0	00		NUL
^A	1	01	☺	SOH
^B	2	02	☻	STX
^C	3	03	♥	ETX
^D	4	04	♦	EOT
^E	5	05	♣	ENQ
^F	6	06	♠	ACK
^G	7	07	•	BEL
^H	8	08	◘	BS
^I	9	09	○	HT
^J	10	0A	◙	LF
^K	11	0B	♂	VT
^L	12	0C	♀	FF
^M	13	0D	♪	CR
^N	14	0E	♫	SO
^O	15	0F	☼	SI
^P	16	10	►	DLE
^Q	17	11	◄	DC1
^R	18	12	↕	DC2
^S	19	13	‼	DC3
^T	20	14	¶	DC4
^U	21	15	§	NAK
^V	22	16	▬	SYN
^W	23	17	↨	ETB
^X	24	18	↑	CAN
^Y	25	19	↓	EM
^Z	26	1A	→	SUB
^[	27	1B	←	ESC
^\	28	1C	∟	FS
^]	29	1D	↔	GS
^^	30	1E	▲	RS
^_	31	1F	▼	US

Dec	Hex	Char
32	20	sp
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	⌂

Dec	Hex	Char
128	80	Ç
129	81	ü
130	82	é
131	83	â
132	84	ä
133	85	à
134	86	å
135	87	ç
136	88	ê
137	89	ë
138	8A	è
139	8B	ï
140	8C	î
141	8D	ì
142	8E	Ä
143	8F	Å
144	90	É
145	91	æ
146	92	Æ
147	93	ô
148	94	ö
149	95	ò
150	96	û
151	97	ù
152	98	ÿ
153	99	Ö
154	9A	Ü
155	9B	¢
156	9C	£
157	9D	¥
158	9E	₧
159	9F	ƒ

Dec	Hex	Char
160	A0	á
161	A1	í
162	A2	ó
163	A3	ú
164	A4	ñ
165	A5	Ñ
166	A6	ª
167	A7	º
168	A8	¿
169	A9	⌐
170	AA	¬
171	AB	½
172	AC	¼
173	AD	¡
174	AE	«
175	AF	»
176	B0	░
177	B1	▒
178	B2	▓
179	B3	│
180	B4	┤
181	B5	╡
182	B6	╢
183	B7	╖
184	B8	╕
185	B9	╣
186	BA	║
187	BB	╗
188	BC	╝
189	BD	╜
190	BE	╛
191	BF	┐

Dec	Hex	Char
192	C0	└
193	C1	┴
194	C2	┬
195	C3	├
196	C4	─
197	C5	┼
198	C6	╞
199	C7	╟
200	C8	╚
201	C9	╔
202	CA	╩
203	CB	╦
204	CC	╠
205	CD	═
206	CE	╬
207	CF	╧
208	D0	╨
209	D1	╤
210	D2	╥
211	D3	╙
212	D4	╘
213	D5	╒
214	D6	╓
215	D7	╫
216	D8	╪
217	D9	┘
218	DA	┌
219	DB	█
220	DC	▄
221	DD	▌
222	DE	▐
223	DF	▀

Dec	Hex	Char
224	E0	α
225	E1	ß
226	E2	Γ
227	E3	π
228	E4	Σ
229	E5	σ
230	E6	µ
231	E7	τ
232	E8	Φ
233	E9	Θ
234	EA	Ω
235	EB	δ
236	EC	∞
237	ED	φ
238	EE	ε
239	EF	∩
240	F0	≡
241	F1	±
242	F2	≥
243	F3	≤
244	F4	⌠
245	F5	⌡
246	F6	÷
247	F7	≈
248	F8	°
249	F9	∙
250	FA	·
251	FB	√
252	FC	ⁿ
253	FD	²
254	FE	■
255	FF	

2 Algorithmes et langages

2.1 Algorithmes

2.1.1 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.

2.1.2 Exemple 1

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

2.1.3 Exemple 2

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")
      fsi
  sinon
      solution ← - b / a
      afficher ("La solution est : ", solution)
  fsi
```

Remarques :

- 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

2.1.4 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.

2.1.5 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

2.2 Notion de variable

2.2.1 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 l'exemple en 2.1.3 les données sont a et b.

2.2.2 Constante

Une **constante** est une valeur fixe utilisée par le programme.

Exemple : Pi, la constante de gravitation, etc.

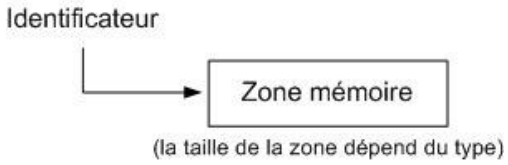
2.2.3 Variable

Une **variable** représente une valeur susceptible de changer au cours de l'exécution du programme.

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

2.2.4 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.



2.2.5 Instructions de base

• ***Affectation***

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

Notation :

variable ← expression

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.

Exemples :

```
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)
```

- ***Structure de sélection simple***

si <condition> alors
< séquence d'instructions >
fsi

ou

si <condition> **alors**
< séquence d'instructions >
sinon
< séquence d'instructions >
fsi

Exemple : Maximum de deux nombres :
si a ≥ b alors
max ← a
sinon
max ← b
fsi

- ***Structures répétitives***

tant que <condition> **faire**
<séquence d'instructions >
Ftq

ou

répéter
<séquence d'instructions>
jusqu'à condition

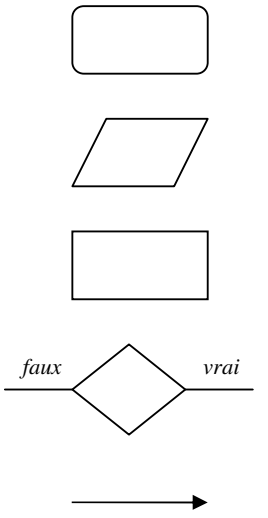
Exemple :
répéter
ecrire(« entrez un nombre inférieur à 10 »)
lire(n)
jusqu'à n < 10

2.3 Représentations d'un algorithme

2.3.1 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

2.3.2 Organigramme



début/fin d'un algorithme

entrées / sorties (lire, afficher)

traitement (séquence d'instructions)

test

sens de lecture

2.3.3 Exemples

- ***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	

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

- ***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 non plafonnées

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

- ***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

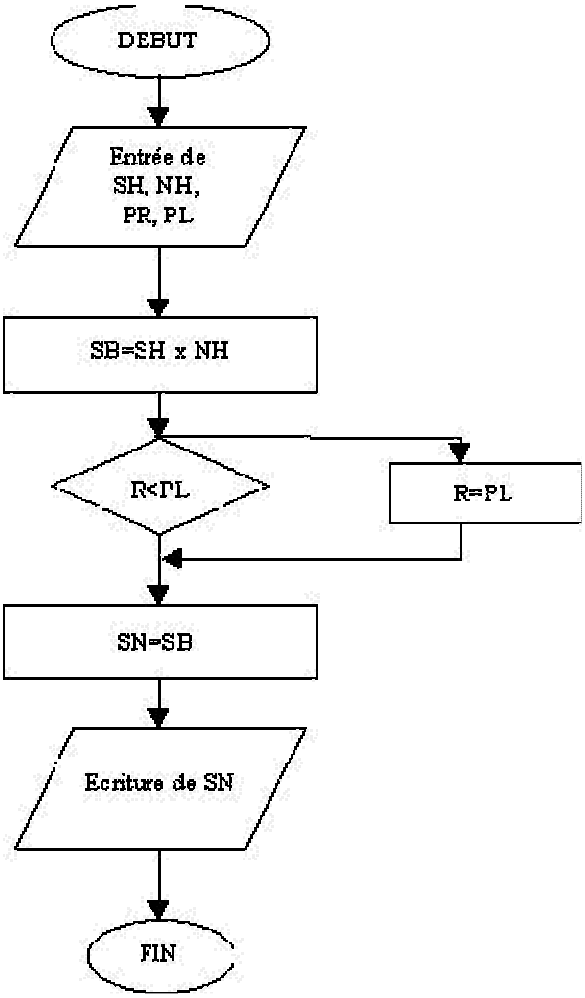
Calculer :

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



2.4 Grammaires et langages

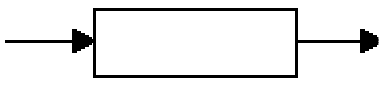
Comme les langages naturels, les langages informatiques utilisent une grammaire (ou syntaxe). La syntaxe d'un langage de programmation est cependant plus rigide et dispose d'un vocabulaire plus limité.

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

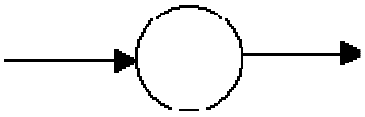
- le formalisme BNF (Backus Naur Form)
- les diagrammes de Conway

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.

2.4.1 Diagrammes de Conway



Un rectangle représente une *catégorie syntaxique*.

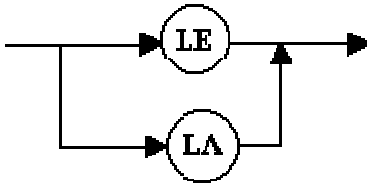


Un cercle représente un *symbole terminal.*

Exemples :

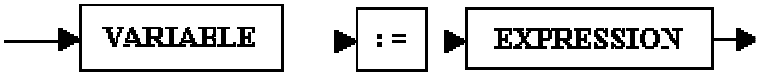
- PHRASE

- ARTICLE

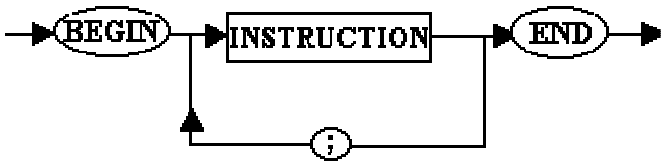


Exemples pour la syntaxe du Pascal :

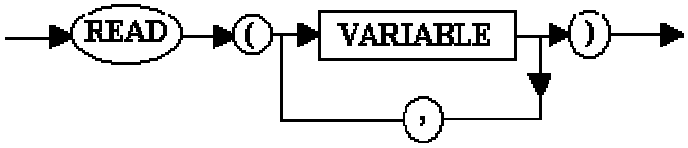
- ***- Affectation***



- ***- Instruction composée (liste d'instructions)***



- ***- Lecture***



2.4.2 Formalisme BNF (Backus-Naur Form)

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 séfini par ».

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>.

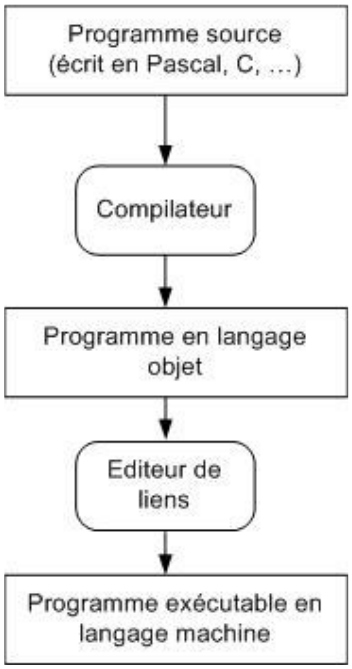
2.4.3 Programmation

- ***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 notepad. 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.



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.

3 Premiers éléments de Pascal

3.1 Le Langage Pascal

3.1.1 C'est un langage typé

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

3.1.2 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.

3.1.3 C'est un langage récursif

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

3.2 Structure globale d'un programme Pascal

En-tête

Déclarations

Constantes

Types

Variables

Fonctions / Procédures

Bloc d'instructions exécutables

• ***En-tête***

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 ;
```

• ***Déclarations***

En Pascal, on peut déclarer des :

- des constantes
- des types
- des variables

L'ordre indiqué doit être impérativement respecté.

- ***Instructions***

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

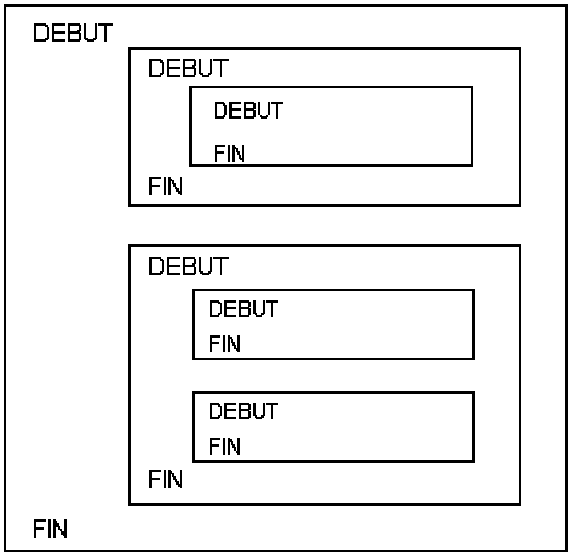
On distingue :

Les **instructions simples** :
Ordre unique, inconditionnel (Ex : affectation)

Les **instructions structurées**
- instructions composées
- instructions itératives
- instructions conditionnelles

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
  - un identificateur
- Fin d'une instruction : par un point virgule ;

- ***Structure de bloc***



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

- ***Exemple de programme***

Nous présentons ci-dessous un programme qui donne la moyenne de n nombres entrés au clavier par l'utilisateur, qui précisera le nombre de données qu'il va taper. 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)

```
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}
```

Remarques : Il est conseillé d'utiliser
- des indentations pour refléter la structure du programme.
- des commentaires pour souligner des points importants.

Un commentaire peut être ajouté en n'importe quel point du programme Un commentaire est un texte encadré par des accolades ou par les caractères (* et *).

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

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.

3.2.1 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" : ^ (utile pour les fichiers ,Cf. chapitre XI)
- Les signes de ponctuation : . , ; : ' ` ! ?

3.2.2 Les mots du langage

• ***Définition***

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

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

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 ...

• ***Les identificateurs***

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 " _ ".

Exemples d'identificateurs légaux :
x2 Z31 xBarre SOMME salaire_net

Exemples d'identificateurs non légaux :
3Mots U.T.C. mot-bis A!8 $PROG AUTRE MOT

• ***Les identificateurs standards***

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).

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

3.3 Déclarations

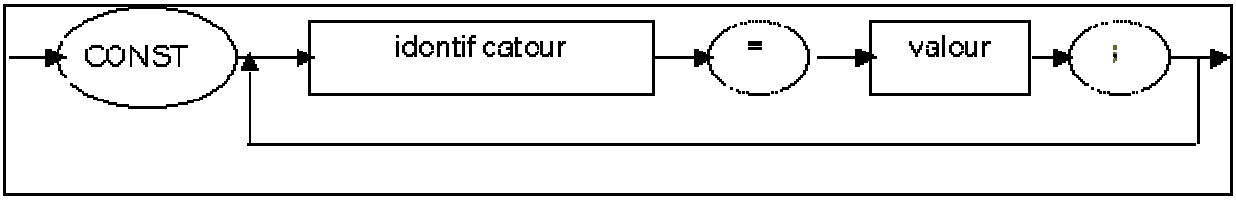
En Pascal, tout symbole (constante, type, variable, procédure, fonction) utilisé dans un programme doit être explicitement déclaré.

3.3.1 Constantes

L'utilisation de constantes en programmation est vivement conseillée.
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 :



Exemples :

```
const
    DEUX = 2;
    PI = 3.14;
```

```
VRAI = true; {  }
FAUX = false;
CAR_A  = 'A';
PHRASE = 'il fait beau';
```

Le point virgule est obligatoire à la fin de chaque déclaration.

3.3.2 Types

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.

- ***Les types standards***

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.

- Le type **integer** :
  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.

  Exemples d'entiers corrects :
  589 0 7 +7 -28 -9999
  Exemples d'entiers incorrects :
  79. 644 895

  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)

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

- Le type **real**
  Les valeurs sont des réels.

  Représentation décimale :
  Signe + partie entière + point + partie décimale

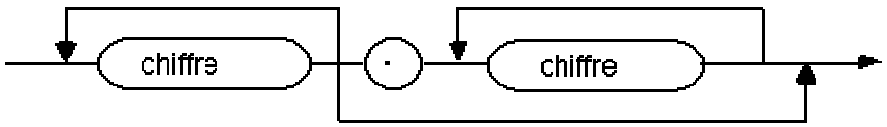


Diagramme syntaxique d'un réel non signé :

Exemples de réels correctement écrits :
3.5 -7.80 0 -0.656 +95000.0
Exemples de réels incorrectement écrits :
8. 75,632 100.

Représentation en virgule flottante
On ajoute un exposant.
Notation : lettre E suivie d'un entier signé ou non signé

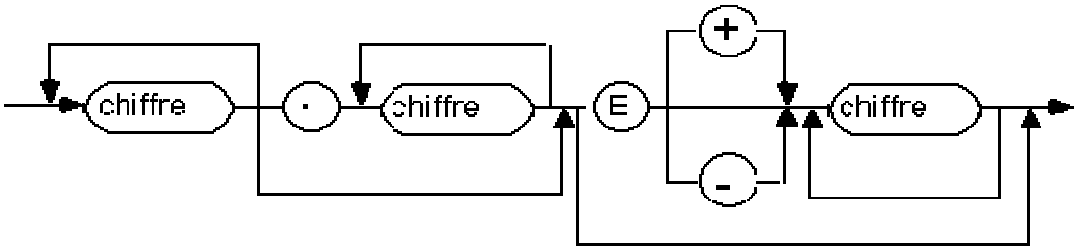


Diagramme syntaxique général (valable aussi bien pour les entiers que les réels) :

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.

- Le type **boolean**
  Les valeurs sont dites *logiques*.
  Ce sont les deux constantes : *true* (vrai) et *false* (faux)

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

- Le type **char**
  Les valeurs sont les caractères, en particulier :
  - alphanumériques : 'a' . . 'z' 'A' . . 'Z' '0' . . '9'
  - le caractère blanc : ' '
  - les caractères accentués
  - les caractères de ponctuation

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

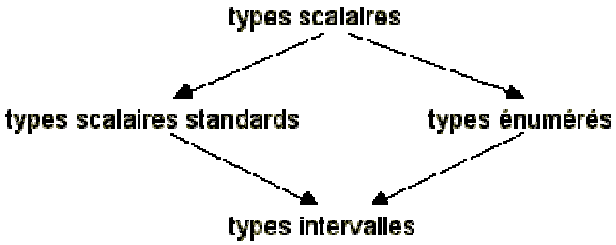
- Le type **string**
  Les valeurs sont des chaînes de caractères.
  Elles sont également représentées entres apostrophes. Lorsqu'il y a une apostrophe dans la chaîne de caractères, elle doit être doublée :
  Ex : 'il fait beau aujourd''hui'

  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.

- ***Les types scalaires et non standards***

Un type est dit scalaire s'il est :
- soit un type scalaire standard (integer ou real),
- soit un type énuméré,
- soit un type intervalle.

Organisation des types scalaires :



- Le type énuméré

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

Syntaxe :

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

Exemples :

```
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);
```

N.B. : Le mot réservé **type** ne doit être écrit qu'une seule fois.

Remarques :

- 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 prédecesseurs d'un élément.

  mardi : prédécesseur de mercredi.
  mercredi : successeur jeudi.

- Le type intervalle

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

Syntaxe :

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

Toutes les valeurs de l'intervalle sont autorisées.

Exemples :

Intervalle d'entiers :
```
type
    Decimal =  0 .. 9 ;
    Octal =  0 .. 7 ;
    Age =  0 .. 150 ;
```

Intervalle de caractères :
```
type
    ABC = 'A' .. 'C' ;
    Maj = 'A' .. 'Z' ;
```

Exemples avec un type non-standard
```
type
    Ouvrable = lundi  .. vendredi ;
    WeekEnd  = samedi .. dimanche ;
    Lettres  = 'A' .. 'Z' ;
```

Remarques :
- 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.

*Exemples de déclarations **incorrectes** :*
```
type
    Octal= 7 .. 0 ;
    Ouvrable = vendredi .. lundi ;
```

3.3.3 Variables

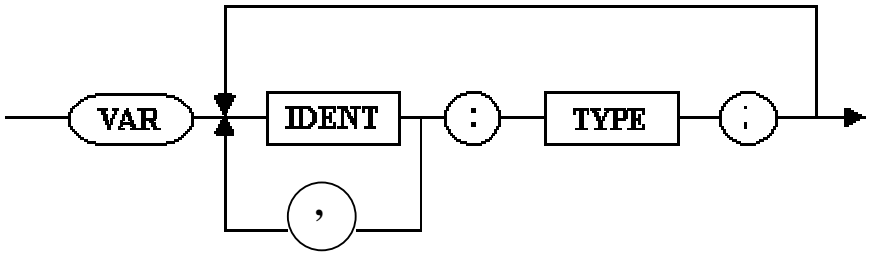
Déclarer une variable, c'est définir l'ensemble des valeurs qu'elle peut prendre.
Toutes les variables utilisées dans un programme doivent être déclarées.

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 ;

Diagramme syntaxique :



Remarques :
- **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).

Exemples

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

avec déclaration locale explicite :

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

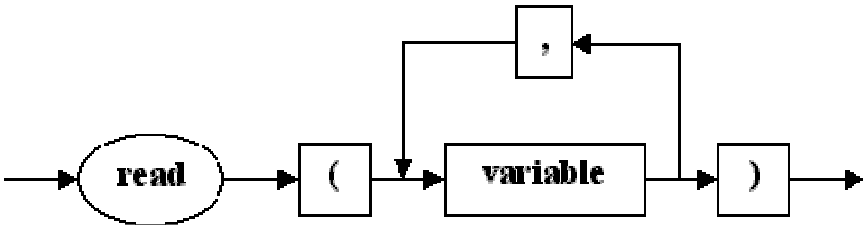
3.3.4 Exemples de déclarations de constantes, types et variables.

```
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 ;
```

3.4 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.

3.4.1 Lecture




```
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);
```

L'instruction :

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

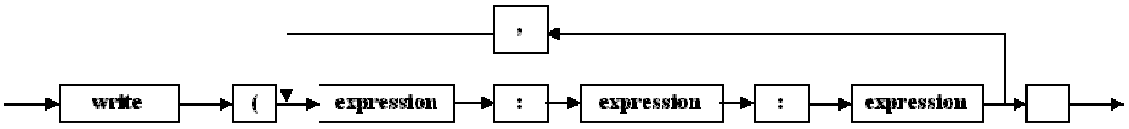
permet, de manière analogue, de lire n valeurs et passe ensuite à la ligne suivante en ignorant ce qui reste éventuellement sur la ligne.

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.
```

3.4.2 Ecriture



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.
```

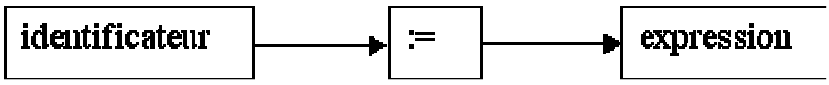
Le programme lit les valeurs de i et r tapées par l'utilisateur et les affiche ensuite à l'écran, sur 5 caractères pour i, et sur 6 caractères pour r, dont 2 chiffres après la virgule.

Autre 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.
```

3.5 Instruction d'affectation



L'instruction d'affectation à un double rôle :

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

Remarques

- 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)

3.6 Opérateurs et fonctions arithmétiques

3.6.1 Opérateurs disponibles

+	somme
-	soustraction
*	multiplication
/	division
DIV	division entière (Ex : 5 div 3 = 1)
MOD	modulo (Ex : 5 mod 3 = 2)

Exemples

```
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.
```

3.6.2 Expressions

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

Exemples :
"i+1", "2.08E3 * x" ou encore "(x>2) OR (x<8)"

3.6.3 Evaluation des expressions

Il suffit d'utiliser les règles de décomposition syntaxique et l'ordre des priorités mathématiques.

Exemples :

- a*b+c se décompose en :
  Expression → Expression simple → Terme + Terme → (Facteur * Facteur) + Facteur

  donc a*b+c est équivalent à : (a*b)+c

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

3.6.4 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

3.6.5 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

3.7 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**.

3.7.1 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"

3.7.2 La Modularité

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

3.7.3 L'Efficacité

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

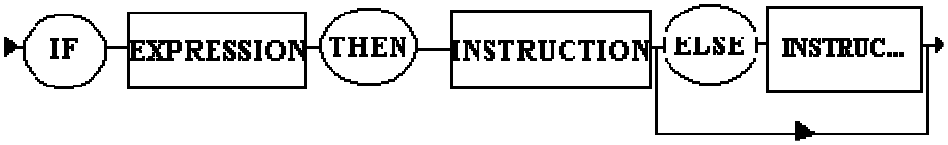
3.7.4 Bénéfices attendus

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

4 Instructions alternatives

4.1 Choix simple

Syntaxe :



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*).

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

Points importants :

- Surtout pas de point virgule immédiatement avant ELSE !
- L'instruction alternative est facultative
- La valeur de la condition doit être booléenne
- Les instructions peuvent être simples ou composées

Exemple 1 : Equation du premier degré

Ecrire un programme qui résoud une équation du premier degré Ax+b=0 qui lit les valeurs de A et B entrées par l'utilisateur.

```
program Premier_Degre;
var
    A, B : real;
begin
    write('entrez les coefficients A et B : ');
    readln(A,B);
    if A=0 {évaluation de la condition}
    then
        if B=0
        then
            writeln('Indéterminé !')
        else
            writeln('Impossible !')
    else
        writeln('La solution est : ',-B/A:10:3);
end.
```

Exemple 2 : Maximum de deux nombres

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

```
program MAXIMUM_DEUX ;
var
    X,Y : real ;
    MAX : real ;
begin
    writeln('Tapez les deux nombres:')
    read (X, Y) ;
    if  X > Y {évaluation de la condition}
    then
        MAX := X
    else
        MAX := Y ;
    writeln('Le plus grand nombre est ',MAX);
end.
```

Exemple 3 : Exemple avec expressions relationnelles et booléennes

```
program GRAND_PETIT ;
var
    SEX : (M,F) ;
    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.
```

4.2 Choix multiple

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).

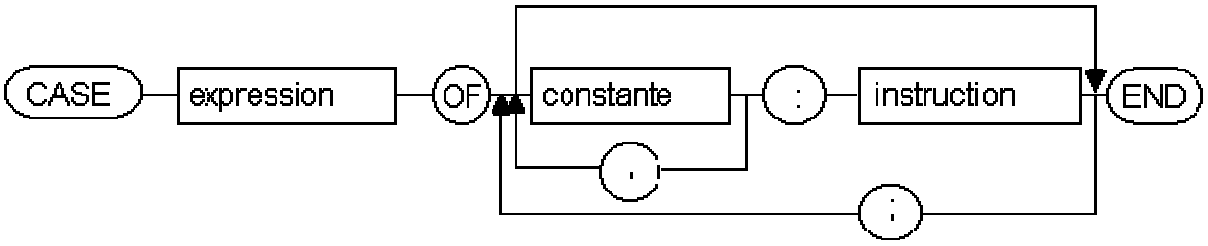
Ainsi au lieu d'écrire :

```
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;
```

Syntaxe



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** !

Exemple 1 : Simuler une calculatrice

```
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.
```

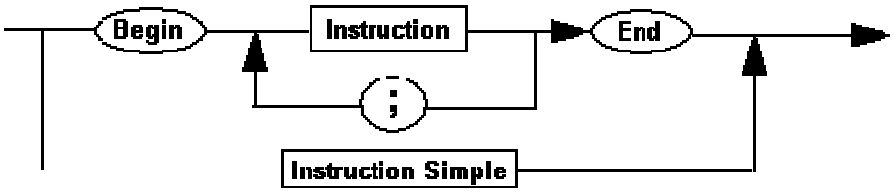
Exemple 2 : Le loto

```
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.
```

4.3 Instructions composées

Les **instructions composées** permettent 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



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

5 Instructions itératives

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.

5.1 Boucles à bornes définies

5.1.1 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>

Exemples :

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>

5.1.2 Exemple d'itérations à bornes définies

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 :
   2.1 Lire NH, SH
   2.2 SB ← SH x NH
   2.3 R← SB x PR
   2.4 Si R > PL alors R ← PL
   2.5 SN ← SB - R
   2.6 Ecrire SN
3. Fin de l'itération

5.1.3 La boucle à bornes définies en Pascal : for

Syntaxe :

```
for variable := exp.1 to exp.2 do Bloc d'instructions;
```

Remarques :

- la variable doit être de type scalaire (entier, énuméré, intervalle ou caractère). Elle ne peut pas être de type réel.
- si `exp.1 > exp.2` 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;
```

5.2 Boucles à bornes non définies

Syntaxe de la boucle TANT QUE :

5.2.1 Boucle TANT QUE

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

Exemples :

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

TANT QUE le feu est vert
FAIRE action de passer

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

5.2.2 En Pascal : while ... do

Syntaxe :

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

Remarques :

- 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
- 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.
```

5.2.3 Boucle REPETER ... JUSQU'A

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

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

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

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

5.2.4 En Pascal : repeat ... until

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

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

5.2.5 Comparaison des 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 (vue précédemment).

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:
    3.1 Lire NH, SH
    3.2 SB ← SH x NH
    3.3 R← SB x PR
    3.4 Si R > PL alors R ← PL
    3.5 SN ← SB - R
    3.6 Ecrire SN
    3.7 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 :
    3.1 Lire NH, SH
    3.2 SB ← SH x NH
    3.3 R← SB x PR
    3.4 Si R > PL alors R ← PL
    3.5 SN ← SB - R
    3.6 Ecrire SN
    3.7 Donner à I la valeur suivante.
4. Jusqu'à ce que I=N

5.2.6 Exemples

Exemple1 : Calculer la somme des N premiers entiers naturels

---> On utilise une boucle à bornes définies, puisqu'on connaît l'intervalle de variations (de 1 à N)

1) Lire N
2) Somme ← 0
3) Pour i variant de 1 à N faire
    Somme ← Somme + i
4) Ecrire les résultats : 'Résultat = ' Somme

Exemple comparatif :
Nous allons à présent traîter le même exemple, avec trois boucles différentes.
Il s'agit de reconstruire l'opération de multiplication, en effectuant des sommes successives.

Soit à effectuer le produit des entiers naturels M et K (distincts de 0)
Données :
M multiplicande
K multiplicateur
Résultat :
P produit
Méthode :
ajouter K fois le multiplicande

Forme 1 : POUR

1. Lire K, M
2. P ← 0
3. Pour i variant de 1 à K faire
   P ← P+M
4. Afficher le résultat P

FORME 2 : avec TANT QUE

1. Lire K, M
2. P ← 0
   i ← 1
3. Tant que i ← K faire
   P ← P+M
   i ← i+1
4. Afficher le résultat P

FORME 3 : avec REPETER

1. Lire K, M
2. P ← 0
   i ← 1
3. Répéter
   P ← P+M
   i ← i+1
   Jusqu'à i>K
4. Afficher le résultat P

6 Tableaux

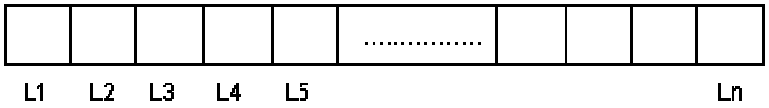
6.1 Tableaux à une dimension

6.1.1 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*

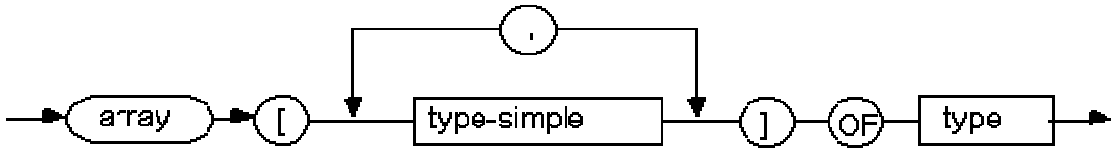
Il peut être représenté sous la forme suivante :



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

Déclaration d'un type tableau en Pascal :

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



Remarques :

- 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 connait 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**.

Exemples :

```
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 ;
```

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;
```

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; ...
```

6.1.2 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]);

Exemple 1:
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.
```

Exemple 2 :

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 ;
```

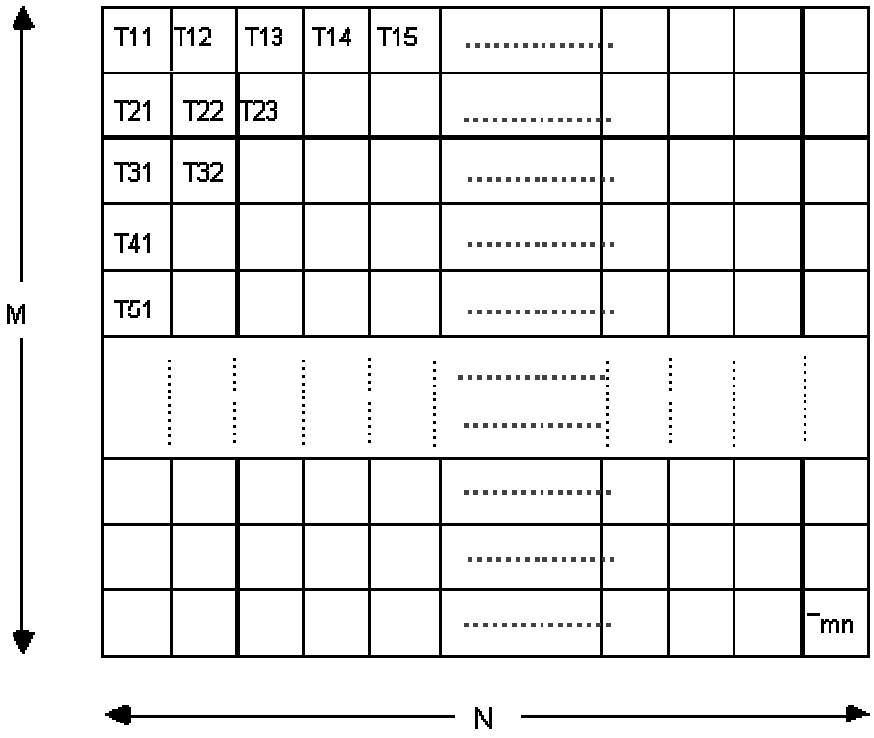
```
var
    U, V  :  vecteur ;
    RESULTAT : real ;
    C : coordonnee ;
begin
    resultat := 0 ;
    for  C := X1  to  X3  do
    begin
        read (u[c]) ;
        readln (v[c]) ;
        resultat := resultat + u[c] * v[c] ;
    end ;
    writeln ('le produit scalaire est : ', resultat) ;
end.
```

6.2 Tableaux à plusieurs dimensions

6.2.1 Tableau à 2 dimensions

Un tableau de dimension 2 est parfois également appelé "**MATRICE**".

Un tel tableau peut être représenté sous la forme suivante :



Les Tij (i = 1 ... M, j = 1 ... N) doivent être de même type
La dimension du tableau est : 2 (matrice)
Le nombre de cases disponibles est : M x N

Déclaration du type :

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

Accès à un élément d'un tableau T de ce type :
T[i][j] ou T[i,j]

6.2.2 Exemples

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;
```

--> accès à un élément par : **`lieu[i][j]`** ou **`lieu[i,j]`**

Exemple de programme :
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 ou 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.
```

7 Chaînes de caractères

7.1 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 :

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

Exemples :
```
s := 'Bonjour';
s[4] := 's';
s[6] := 'i';
    { A présent, s vaut 'Bonsoir' }
s := 'ok';
    { A 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).

7.2 Opérateurs et fonctions

7.2.1 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

7.2.2 Concaténation

La concaténation est une opération qui permet d'accoler 2 ou plusieurs chaines de caratères.

Syntaxe Pascal :
```
s:= s1 + s2 + s3... ;
```

Exemple :

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

7.2.3 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

7.2.4 Position d'une sous-chaîne dans une chaîne

Fonction **pos** :

```
pos(souschaîne, chaîne)
```

renvoie la position de la sous chaîne dans la chaîne.

7.2.5 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'

7.3 Fonctions de codage/ décodage des caractères

7.3.1 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

7.3.2 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'

7.3.3 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'

7.4 Exemples

Exemple 1

```
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.
```

Exemple 2 :

```
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.
```

8 Fonctions et Procédures

8.1 Procédures

8.1.1 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.

8.1.2 Déclaration

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

Déclaration d'une procédure :

```
procedure <identificateur> <liste de paramètres> ;
```

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.

8.1.3 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.

Appel d'une procédure :

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

Pour appeler la procédure d'affichage de l'exemple précédent on écrirait par exemple :

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

8.2 Fonctions

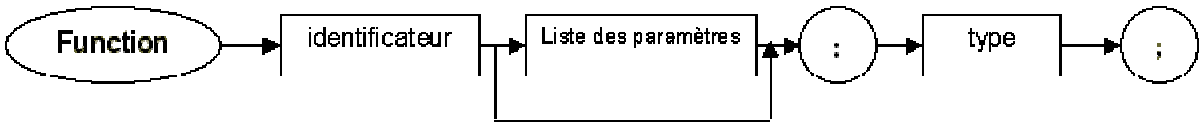
8.2.1 Définition

Une **fonction** est analogue à une procédure mais elle doit de plus retourner une valeur.
Le type de cette valeur doit être spécifié explicitement.

8.2.2 Déclaration

On déclare les fonctions au même niveau que les procédures (après **const, type, var**)

Diagramme de déclaration d'une fonction :

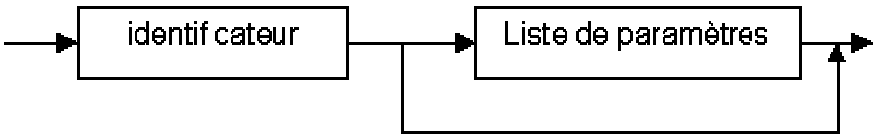


8.2.3 Appel

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.

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.

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 }
```

8.2.4 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 :

Examinons le programme suivant

```
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, d'où l'erreur.

8.3 Variables globales et variables locales

8.3.1 Définitions

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**.

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.
```

8.3.2 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.

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

Exemple :

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

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

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).

Remarques :

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.

8.3.3 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

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

8.4 Paramètres

8.4.1 Exemple

Supposons qu'on ait à résoudre des équations du second degré en divers points d'un programme :
- la première fois $Rx^2 + Sx + T = 0$,
- la deuxième fois $Mx^2 + Nx + P = 0$,
- la troisième fois $Ux^2 + Vx + W = 0$.

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.

La déclaration sera :

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

Un paramètres 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.

8.4.2 Passage de paramètre par valeur

Principe *:* passer à la procédure des valeurs qui seront les données d'entrée et de travail pour la procédure.

Exemple de 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**).

Exemple 1 : *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 2 : *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).

8.4.3 Passage de paramètre par adresse

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 de 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.

Exemple 1 :

```
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 2 : 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.

8.4.4 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.

8.4.5 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.

Nous donnons ci-dessous un 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.

9 Ensembles

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

9.1 Définition et syntaxe

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;
```

Exemple 1 :

```
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.

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

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

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

Exemple 2 :

```
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] ;
```

9.2 Opérations sur les ensembles

9.2.1 Union de deux ensembles--->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]

9.2.2 Intersection de deux ensembles---> 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]

9.2.3 Différence de deux ensembles ---> 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]

9.2.4 Egalité ---> opérateur =

Exemple :

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

On pourra écrire :

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

9.2.5 Inégalité ---> opérateur <>

Exemple :

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

On pourra écrire :

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

9.2.6 Inclusion ---> opérateurs <= ou >=

Exemple :

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

L'expression **[mi, sol] <= [ré . . sol]** sera évaluée à *true*

De même : **[4,7,9] >= [7, 9]**

9.2.7 Appartenance ---> opérateur in

Exemples :

```
    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 (Cf 7.4) :

```
    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;
```

10 Enregistrements

10.1 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**.

10.2 Déclaration d'un type enregistrement

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

Spécification d'un champ dans la déclaration :

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

Exemples :

```
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;
```

10.3 Accès aux champs d'une variable de type enregistrement

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 ;
```

Exemples : soient les déclarations suivantes :

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

Pour afficher la marque et le prix de la variable *auto*, on pourra écrire :

```
writeln(auto.marque, auto.prix...);
```

10.4 Ecriture dans un enregistrement

- 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 :
  enregistrement-1 := enregistrement-2 ;

- Affectation directe sur un champ :
  id_nom.id_champ := valeur ;

- Par une instruction de lecture :
  read (id_nom.id_champ) ;

Exemple 1 : 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 2 : 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' ;
 ...
```

10.5 Instruction with

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;
```

Exemple :

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

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

10.6 Tableaux d'enregistrements

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

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

```
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.

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';
```

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;
```

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;
```

11 Fichiers

11.1 Définitions

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

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

11.2 Manipulation des fichiers

Le système d'exploitation fournit :

- *des primitives de programmation pour la :*
  - création de fichier
  - ouverture d'un fichier
  - lecture dans un fichier
  - écriture dans un fichier
  - fermeture d'un fichier
  - destruction d'un fichier
- *une bibliothèque de programmes utilitaires pour :*
  - copier des fichiers
  - renommer des fichiers
  - lister des fichiers
  - détruire des fichiers

11.3 Les supports physiques

11.3.1 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.

11.3.2 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 !

11.4 Organisation et accès

11.4.1 Définition

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

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.

11.4.2 Organisation séquentielle

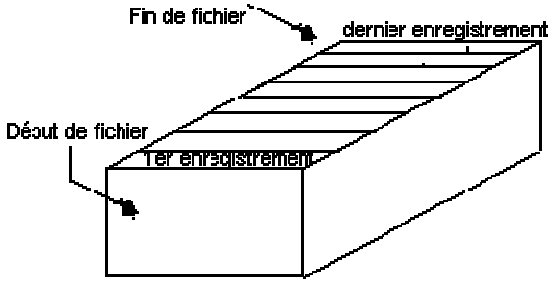
Elle ne permet qu'un seul accès : le séquentiel.
Toutes les informations sont enregistrées de façon **séquentielle** (linéaire) les unes à la suite des autres.

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 $i_{ème}$ enregistrement du fichier, il faudra donc d'abord lire les i-1 enregistrements qui le précèdent.

Caractéristiques :

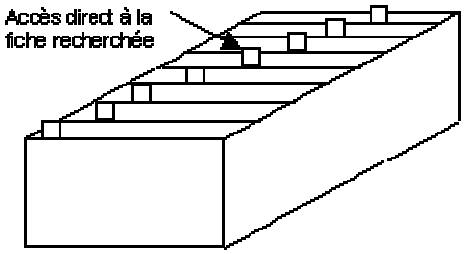
Pour ajouter une information, il faut se placer en fin de fichier
A la fin du fichier, on trouve un enregistrement spécial FIN_DE_FICHIER



11.4.3 Organisation relative (accès direct)

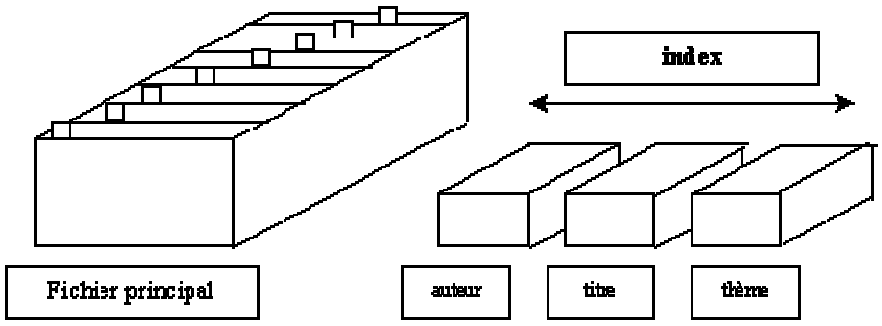
Chaque enregistrement possède un **numéro**. On accède à la fiche recherchée en spécifiant ce numéro d'enregistrement. L'indication d'un numéro permet donc un **accès direct** à l'information ainsi référencée.

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



11.4.4 Organisation indexée

On crée des fichiers supplémentaires d'index



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 (=, <>, <, <=, >, >=).

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....

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.

11.5 Les fichiers séquentiels en Pascal

11.5.1 Définitions

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.

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.

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

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

11.5.2 Déclaration d'un fichier

Syntaxe

```
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 ;
    COORDONNES = record
                     abscisse, ordonnee : real ;
                  end ;
    TRACE = file of coordonnees ;

var
    fichier_client : REPERTOIRE ;
    courbe : TRACE ;
```

11.5.3 Création d'un fichier

Syntaxe

```
    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.

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*

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.

11.5.4 Ecriture dans un fichier

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 :

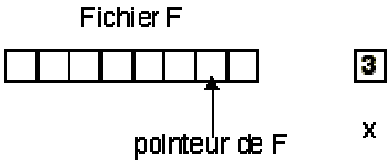
```
write (f,x)
```

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

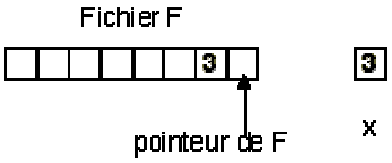
Remarque : Le type de *x* doit être compatible avec le type des enregistrements du fichier.

Principe :

- avant l'exécution de la commande :



- après l'exécution de la commande :



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.

11.5.5 Ouverture d'un fichier existant

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

```
reset (id_fichier);
```

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 »).

11.5.6 Lecture dans un fichier

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

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

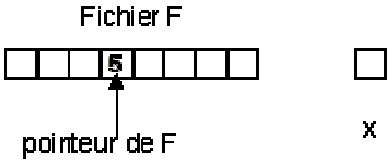
Cette instruction permet de lire l'enregistrement du fichier repéré par le pointeur de fichier, puis de placer cet enregistrement dans la variable x passée en paramètre.

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

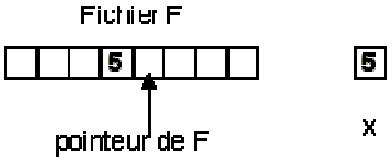
Cette instruction ne peut être exécutée que si la commande `reset(f)` a été utilisée auparavant.

Principe :

- avant l'exécution de la commande :



- après l'exécution de la commande :



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.

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.

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

11.5.7 Association entre un nom interne et un nom externe

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.

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.

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') ;
```

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 : 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) ;
...
```

11.5.8 Fermeture d'un fichier

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

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.

11.6 Les fichiers structurés

11.6.1 Définition

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**.

11.6.2 Lecture d'un fichier

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

```
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.
```

11.6.3 Ecriture dans un fichier

```
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.
```

11.7 Les fichiers 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.

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).

11.7.1 Déclaration

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

Il suffit donc de déclarer une variable de type Text :

```
var
    fich : Text ;
```

Remarque : en fonction des compilateurs, on doit parfois utiliser à la place de Text : `File of char;` `File of string; File of Text; ou TextFile;`

11.7.2 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.

```
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.
```

11.7.3 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.

```
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.
```

12 Récursivité

12.1 Introduction

12.1.1 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.

12.1.2 Programmation itérative et programmation récursive

Exemple : la fonction factorielle

- ***Programmation itérative***

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;
```

- ***Programmation récursive.***

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;
```

12.2 Autres exemples

12.2.1 Somme des n premiers entiers

On désire calculer la somme des *n* premiers entiers naturels,
$S(n)=1+2+3+...+ n$

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** en utilisant le fait que :

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

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.
```

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 d'execution du programme avec N=4 :

```
début
appel de SOMME avec N = 3{début de l'empilement}
    début
        appel de SOMME avec N = 2
            début
                appel de SOMME avec N = 1
                    début
                        appel de SOMME avec N = 0{fin de l'empilement}
                            début
                                SOMME ← 0
                            fin {début du dépilement}
                        SOMME ← SOMME(0) + 1
                    fin
                SOMME ← SOMME(1) + 2
            fin
            SOMME ← SOMME(2) + 3
    fin
    SOMME ← SOMME(3) + 4
fin{fin du dépilement}
Résultat : SOMME(4) = 10
Fin
```

12.2.2 Puissance d'un entier

On peut écrire fonction récursive pour calculer la puissance d'un entier en utilisant :
$\mathbf{x^n = x * x^{n-1}}$ si n > 0
$\mathbf{x^n = x^{n+1} / x}$ si n < 0
avec la condition d'arrêt: $\mathbf{x^0 = 1}$,

```
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.
```

12.2.3 Exercice d'application

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

```
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.
```

12.2.4 PGCD Récursif

On utilise le résultat suivant :
PGCD (a , b) = PGCD (b , r) si r <> 0, avec $r = a \bmod b$
(r est le reste de la division euclidienne de a par b)

Condition d'arret : PGCD (a , b) = b si $r = 0$

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

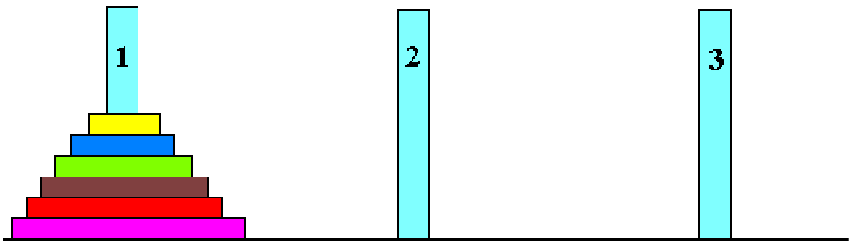
Le programme sera donc le suivant :

```
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.
```

12.3 Tours de HANOI

12.3.1 Problème

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.



12.3.2 Analyse

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.

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 $x^n = x * x^{n-1}$).

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*)

12.3.3 Programme

```
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.
```

12.3.4 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

13 Exercices

13.1 Programmation et boucles

13.1.1 Enoncé 1

I. Programmation

Il est maintenant temps de vous confronter à un problème complet et d'en résoudre toutes les étapes. Ecrire un programme réalisant la facturation d'un article livré en un ou plusieurs exemplaires. On fournira en données le nombre d'articles et leur prix unitaire hors-taxe. La taux TVA sera toujours de 18.6 %. Si le montant TTC dépasse 1000 FF, on établira une remise de 5 %. On cherche à ce que le dialogue se présente ainsi :

```
Nombre d'articles  :  xxxx
Prix unitaire HT   :  xxxx
Montant TTC        :  xxxx
Remise             :  xxxx
Net à payer        :  xxxx
```

N'oubliez pas de rester synthétique et de ne pas brûler les étapes. Prenez suffisamment de temps pour bien analyser le problème, et éventuellement le décomposer en sous-problèmes. Cernez les entrées et les sorties, ainsi que l'usage précis que vous ferez des différentes variables. Une fois l'algorithme élaboré, le codage en Pascal ne devrait pas vous poser trop de difficultés. Essayez de vous habituer à commenter le programme, et utiliser des noms de variables significatifs (compléter la table A). On affichera les résultats sur 8 caractères avec 2 chiffres après la virgule.

II. Boucles

Il s'agit cette fois de comprendre les méandres d'un programme qui n'est pas de votre cru. Les plus courageux pourraient essayer de le faire tourner à la main, mais une série d'exécutions en machine devrait vous être plus profitable. Une fois le principe compris et clairement expliqué, vous pourrez formuler des suggestions quant à d'éventuelles modifications tant sur la forme que sur le fond. Faites attention à ne pas généraliser trop rapidement un comportement singulier.

```
PROGRAM Inconnu; USES WINCRT; VAR i, n, m, p : INTEGER; BEGIN
REPEAT WRITE(' Introduire un entier m : '); READLN(m); WRITE('
Introduire un entier p : '); READLN(p); WRITE(m:2,' avec
',p:2,' donne : '); FOR i := 1 TO m DO BEGIN n := 1 + p * (i -
1); WRITE(n,' ',); END; WRITELN; UNTIL m = 0; END.
```

1. Ce programme est très mal présenté, recopiez-le proprement en utilisant les règles d'indentation du Pascal.
2. Un bon moyen pour découvrir le but d'un programme est de le tester (compléter la table B).
3. Réécrivez le programme en ajoutant des commentaires.

TABLE A

Nombre d'articles	prix unitaire HT (FF)	Montant TTC (FF)	Remise (FF)	Net à payer (FF)
4	246			
2	67			
6	200			
28	78			
7	69			
12	157			
15	168			
18	247			
10	86			
4	75			
4	72			
20	165			
14	64			
21	100			
2	92			

TABLE B

m	p	Itérations									
2	5										
11	7										
5	6										
1	1										
5	0										
0	7										
1	0										
5	1										
3	7										
11	3										
5	6										
11	2										

13.1.2 Corrigé 1

I. Programmation

Programme réalisant la facturation d'un article livré en un ou plusieurs exemplaires. On fournira en données le nombre d'articles et leur prix unitaire hors-taxe. La taux TVA sera toujours de 18.6 %. Si le montant TTC dépasse 1000 FF, on établira une remise de 5 %. On affichera les résultats sur 8 caractères avec 2 chiffres après la virgule. On cherche à ce que le dialogue se présente ainsi :

```
Nombre d'articles   :  xxxx
Prix unitaire HT    :  xxxx
Montant TTC         :  xxxx
Remise              :  xxxx
Net à payer         :  xxxx
```

```
constante taux_tva = 0.186, taux_remise = 0.05
entier n_articles
réel prix_ht, prix_ttc, montant_ttc, remise, net_a_payer
écrire ``Introduisez le nombre d'articles : ''
lire n_articles
écrire ``Introduisez le prix unitaire (FF) : ''
lire prix_ht
montant_ttc = prix_ht * n_articles * 1.186
si montant_ttc > 1000 alors
remise = montant_ttc * 0.05
sinon
remise = 0
fsi
net_a_payer = montant_ttc - remise
écrire '' Nombre d'articles : '', n_articles
écrire '' Prix unitaire HT : '', prix_ht, '' FF ''
écrire '' Montant TTC : '', montant_ttc, '' FF ''
écrire '' Remise : '', remise, '' FF ''
écrire '' Net à payer : '', net_a_payer, '' FF ''
```

```
PROGRAM Facturation_avec_remise;
USES WINCRT;
CONST
taux_tva = 0.186;
taux_remise = 0.05;
VAR
prix_ht, prix_ttc, montant_ttc, remise, net_a_payer : REAL;
n_articles : INTEGER;
BEGIN
WRITE('Introduisez le nombre d'articles : ');
READLN(n_articles);
WRITE('Introduisez le prix unitaire : ');
READLN(prix_ht);
montant_ttc := prix_ht * n_articles * 1.186;
IF montant_ttc > 1000 THEN
remise := montant_ttc * 0.05
ELSE
remise := 0;
net_a_payer := montant_ttc - remise;
WRITELN('Nombre d'articles : ',n_articles:8:2);
WRITELN('Prix unitaire HT : ',prix_ht:8:2,' FF');
WRITELN('Montant TTC : ',montant_ttc:8:2,' FF');
```

```
WRITELN('Remise : ',remise:8:2,' FF');
WRITELN('Net à payer : ',net_a_payer:8:2,' FF');
END.
```

TABLE A

Nombre d'articles	prix unitaire HT (FF)	Montant TTC (FF)	Remise (FF)	Net à payer (FF)
4	246	1167.02	58.35	1108.67
2	67	158.92	0	158.92
6	299	2127.68	106.38	2021.30
28	78	2590.22	129.51	2460.71
7	69	572.84	0	572.84
12	157	2234.42	111.72	2122.70
15	168	3088.72	148.44	2939.28
18	247	5272.96	263.65	5009.31
19	86	1937.92	96.90	1841.03
4	75	355.80	0	355.80
4	72	341.57	0	341.57
20	165	3913.80	195.69	3718.11
14	64	1062.65	53.13	1009.52
21	199	4956.29	247.81	4708.48
2	92	218.22	0	218.22

II. Boucles

Ce programme n'a aucune utilité (pour l'instant en tout cas). Il permet de générer , en partant de 1, une suite de m entiers avec un pas p.

```
entier i, n, m, p
répèter
écrire " Introduisez un entier m : "
lire m
écrire " Introduisez un entier p : "
lire p
écrire m," avec ",p," donne : "
pour i = 1 jusqu'à m faire
n = 1 + p * (i - 1)
écrire n
fpour
jusqu'à m = 0
```

```
PROGRAM De_p_en_p;
USES WINCRT;
VAR
i, n, m, p : INTEGER;
BEGIN
REPEAT
```

```
WRITE(' Introduire un entier m : ');
READLN(m);
WRITE(' Introduire un entier p : ');
READLN(p);
WRITE(m:2,' avec ',p:2,' donne : ');
FOR i := 1 TO m DO
BEGIN
n := 1 + p * (i - 1);
WRITE(n,' ',);
END;
WRITELN;
UNTIL m = 0;
END.
```

TABLE B

m	p	Itérations										
2	5	1	6									
11	7	1	8	15	22	29	36	43	50	57	64	71
6	6	1	7	13	19	25	31					
1	1	1										
6	9	1	10	19	28	37	46					
9	7	1	8	15	22	29	36	43	50	57		
1	9	1										
6	1	1	2	3	4	5	6					
8	7	1	8	15	22	29	36	43	50			
11	3	1	4	7	10	13	16	19	22	25	28	31
6	6	1	7	13	19	25	31					
11	2	1	3	5	7	9	11	13	15	17	19	21

13.1.3 Enoncé 2

Exercice 1 :

Ecrire un programme Pascal qui permet d'afficher le triangle de Pythagore ; triangle dans lequel l'insertion d'une ligne correspond au résultat de la multiplication de l'indice de la ligne avec les indices des colonnes.

Exemple : (pour n = 5)

```
1
2  4
3  6   9
4  8    12 16
5  10 15 20 25
```

Exercice 2 :

La suite de fibonacci forme une séquence intéressante, dans la quelle chaque terme est égal à la somme des deux termes précédents. Formellement la suite est définie par :

Fi+1 = Fi + Fi-1

où Fi correspond au ième terme de Fibonacci. Par définition,les deux premiers termes de Fibonacci sont égaux à 1 (F1 = F2 = 1).

F3 = F2 + F1 = 2
F4 = F3 + F2 = 3
F5 = F4 + F3 = 5

et ainsi de suite . . .

- Ecrivez un programme Pascal affichant les n premiers termes de Fibonacci. Testez le avec n = 23
- Partez ensuite de ce premier programme pour en concevoir un second permettant de lire la valeur d'un entier positif et de déterminer s'il fait partie de la suite de Fibonacci.
- Rédigez tout programme de manière à ce qu'il s'exécute de manière répétitive (boucle) jusqu'à ce que l'utilisateur spécifie la valeur zéro en entrée, indiquant ainsi que l'utilisateur veut quitter le programme. Effectuez des tests pour plusieurs valeurs entières de votre choix.

NB : faites en sorte que vos programmes soient lisibles (commentés et bien structurés)

13.1.4 Corrigé 2

Programme 1

```
Program Pythagore;
var
   i, j , n : integer ;
begin
   (* Lecture des variables *)
   writeln('n = ' ); readln(n) ;
   (* Calcul et Affichage du triangle*)
   for i := 1 to n do
   begin
      for j:= 1 to i do write(i*j :3,'  ');
      writeln;
   end ;
end;
```

Programme 2

```
program Fibonacci;
  var
  n , x, y, z : integer ; (* x = Fn ; y = Fn-1 ; z = Fn-2 *)
begin
        (* lecture des variables *)
        write('n = ' );
        readln(n);
        while (n > 0) do (* condition d'arret *)
        begin
              writeln('voici les ',n, ' premiers termes de la suite:' ) ;
              if  (n >= 1)  then writeln('Terme 1 = 1'); (* Affichage du premier terme *)
              if  (n >= 2) then writeln('Terme 2 = 1');  (* Affichage du second terme *)
              y:=1; z:=1;
              for i:=3 to n do
              begin
                        x := y + z ; (* Fn = Fn-1 + Fn-2 *)
                        z := y ;   (* Fn-2 := Fn-1 *)
                        y := x ;   (* Fn-1 := Fn *)
                        writeln('Terme ',i, ' = ' , x);(* Affichage de la suite*)
        end;
        write('Entrer un nombre, 0 pour terminer, n= ');
        readln(n);
end;
end.
```

```
program TestFibo;
var
    k, x, y, z ,i: integer ;  (* x = Fn ; y = Fn-1 ; z = Fn-2 *)
begin
 (* lecture des variables *)
    write('Entrer un nombre : ' ) ; readln(k);
    while (k<>0) do
    begin
        if (k = 1) then
            writeln( '1 fait partie de la suite de Fibonacci
                                                                F1=F2=1')
        else
            begin
                i:=2;  (* l'indice du terme *)
                y=1 ; z=1; x:=2;
                while (x<k) do (* tant que le terme calcule
                                  est < a k continuer la recherche *)
                begin
                    i := i+1;
                    x := y + z ;  (* Fn = Fn-1 + Fn-2 *)
                    z := y ;    (* Fn-2 := Fn-1 *)
                    y := x ;    (* Fn-1 := Fn *)
                end;
                (* tester si k est de fibonacci *)
                if (k=x) then
                    writeln(' k, ' fait partie de la suite de
                                    Fibonacci, c'est le terme ',i )
                else writeln(' k, ' ne fait pas partie de la suite
                                   de Fibonacci' );
            end;
            write('entrer un autre nombre , 0 pour terminer); readln(k);
```

```
    end;
end .
```

13.2 Procédures - Fonctions et Tableaux

Exercice

On souhaite gérer les notes finales de NF01 des étudiants inscrits dans cette UV. Pour ceci nous utilisons deux tableaux. Le premier pour sauvegarder les noms des étudiants, le second contient les notes finales.

- Ecrire une procédure qui permet de saisir ces données
- Ecrire une procédure qui affiche ces informations à l'écran
- Ecrire une fonction qui retourne la moyenne générale
- Ecrire une procédure qui recherche un étudiant par son nom et affiche sa note finale
- Ecrire une procédure qui modifie la note d'un étudiant.
- Ecrire un programme Pascal intégrant toutes ces fonctionalités dans un menu proposé à l'utilisateur. Le programme principal édite le menu suivant :

Menu principal

- *S . Saisie des Données*
- *A . Affichage des notes*
- *M . Moyenne générale*
- *C. Changement de note*
- *R . Recherche*
- *F . Fin*

Choix :

et s'exécute de manière répétitive jusqu'à ce que l'utilisateur entre 'F' indiquant le fin du programme

---

Pour l'affichage, utiliser :

- clrscr : procédure sans paramètre ; efface le contenu de l'écran et positionne le curseur au coin supérieur gauche
- gotoxy(x,y) procédure à deux paramètres entiers 0<x<25 et 0<y<80 ; positionne le curseur au point de coordonnées (x,y) ; Le point supérieur gauche a comme coordonnées (1,1).

NB. Faites en sorte que vos programmes soient lisibles (commentés et bien structurés).

Correction - *Procédures - Fonctions et Tableaux*

```
program sort;
type
      notes =  array[1..200] of real;
      etudiants = array[1..200] of string[25];
var
not    : notes;   (* tableau des notes *)
   etud    : etudiants;(* tableau des etudiants *)
 n   : integer;(* nombre d'étudiants *)
 choix : char;
{==================PROCEDURE SAISIE DES DONNEES==============}
procedure saisie(var taille : integer ; var t1: etudiants ;var t2: notes ;;
{-============================================================}
```

```
var i: integer;
begin
write('Donner le nombre d''etudiants inscrits en NF01: '); readln(taille );
writeln(Vous allez saisir les noms et les notes de tous les etudiant
inscrits');
for i := 1 to taille do
begin
write(' Le nom de l''etudiant :,') ; readln(t1[i]);
write(' Sa note finale : ') ; readln(t2[i]);
end;
end;
{===============PROCEDURE AFFICHAGE DES NOTES==============}
procedure affiche(taille : integer ; t1 : etudiants ; t2: notes);
{=========================================================}
var i: integer;
begin
clrscr;
writeln('Voila la liste des notes : ');
for i := 1 to taille do  writeln(t1[i],'   ', t2[i]:2:2);
end;
{============PROCEDURE CALCUL MOYENNE GENERALE===========}
function moyenne(taille:integer ; t2 : notes ) : real;
{=========================================================}
var i : integer;
moy : real;
moy:=0;
begin
   clrscr;
   for  i := 1to taille do moy := moy + t2[i];
   moyenne := moy/taille;
end;
{==================FONCTION RECHERCHE================}
function recherche(taille : integer ; t1:etudiants ; t2 : notes ): integer;
{=========================================================}
var i : integer;
    nom : string[25];
begin
    clrscr;
    i:=1;
    write(' Entrer le nom de l'étudiant : '); readln (nom);
    while (t1[i] <> nom) and (i<= taille) do i:=i+1;
    if (i<=taille) then
    begin
        recherche:= i;
        writeln('Nom : ', nom,'Note = ', t2[i]:2:2);
    end
else
    begin
        recherche:=0;
        writeln ('Cet etudiant n''est pas inscrit');
    end;
end;
{==================PROCEDURE CHANGEMENT DE NOTE============}
procedure change(taille : integer ; t1:etudiants ; var t2 : notes );
{=================================================================}
var
    i: integer;
begin
    i :=recherche(taille,t1,t2);
    if (i>0) then
    begin
```

```
        writeln ( 'modifier sa note : '); readln(t2[i]);
        writeln('mise à jour terminée')
    end;
end;
{==================PROGRAMME PRINCIPAL===============}
begin
    repeat
    clrscr;
    gotoxy(24,9); writeln('  Menu Principal ');
    gotoxy(22,10);writeln(' ====================================');
    gotoxy(25,11);writeln(' S.   Saisie des données  ' );
    gotoxy(25,12);writeln(' A.  Affichage des notes  ');
    gotoxy(25,13);writeln(' M. Moyenne générale    ');
    gotoxy(25,14);writeln(' C. Changement de note');
    gotoxy(25,15);writeln(' R.  Recherche   ');
    gotoxy(25,16);writeln(' F.   Fin ');
    gotoxy(22,17);writeln(' ====================================');
    gotoxy(30,18);write('Votre choix :  ');
    readln(choix);

    case choix of
    'S' : saisie(n , etud, not);
    'A' : affiche(n , etud , not) ;
    'M' : writeln(' la moyenne est = ', moyenne(n, not));
    'C' : change(n , etud, not);
    'R' : writeln('L''indice de l''etudiant recherche est : ', recherche(n ,
    etud, not));
    until choix = 'F';
end.
```

13.3 Programmation et boucles

I. Programmation

Il est maintenant temps de vous confronter à un problème complet et d'en résoudre toutes les étapes. Ecrire un programme pour transformer des coordonnées cartésiennes (x,y) en coordonnées polaires (ρ, θ) sachant que :

$$\rho^2 = x^2 + y^2$$
$$\theta = \arctan(y/x)$$

avec

a. $\theta = \theta + \pi$ si $x < 0$

b. $\theta = \pi/2$ si $x = 0$ et $y > 0$

c. $\theta = -\pi/2$ si $x = 0$ et $y < 0$

d. θ n'existe pas si $x = 0$ et $y = 0$

N'oubliez pas de rester synthétique et de ne pas brûler les étapes. Prenez suffisamment de temps pour bien analyser le problème, et éventuellement le décomposer en sous-problèmes. Cernez les entrées et les sorties, ainsi que l'usage précis que vous ferez des différentes variables. Une fois l'algorithme élaboré, le codage en Pascal ne devrait pas vous poser trop de difficultés. Essayez de vous habituer à commenter le programme, et utiliser des noms de variables significatifs (compléter la table A). On prendra $\pi = 3.1416$ et on affichera les résultats sur 8 caractères avec 4 chiffres après la virgule.

II. Boucles

Il s'agit cette fois de comprendre les méandres d'un programme qui n'est pas de votre cru. Les plus courageux pourraient essayer de le faire tourner à la main, mais une série d'exécutions en machine devrait vous être plus profitable. Une fois le principe compris et clairement expliqué, vous pourrez formuler des suggestions quant à d'éventuelles modifications tant sur la forme que sur le fond. Faites attention à ne pas généraliser trop rapidement un comportement singulier.

```
PROGRAM Inconnu; VAR n : LONGINT; BEGIN WRITE(' Introduisez un
entier n : '); READLN(n); WRITE(n,' donne'); WHILE NOT(n = 1)
DO BEGIN IF n MOD 2 = 0 THEN n := n DIV 2 ELSE n := n * 3 + 1;
WRITE(' ',n); END; READLN; END.
```

1.

Ce programme est très mal présenté, recopiez-le proprement en utilisant les règles d'indentation du Pascal.

2.

Un bon moyen pour découvrir le but d'un programme est de le tester (compléter la table B).

3.

Que fait ce programme ?

4.

Réécrivez le programme en ajoutant des commentaires.

TABLE A

x	y	ρ	θ (radian)	θ (degré)
8.4215	0.6239			
0.0000	0.0000			
3.2533	2.2733			
1.0000	0.0000			
0.0000	-1.0000			
-1.0000	1.0000			
1.0000	1.0000			
-4.2455	3.6888			
-12.6683	4.4838			
-1.0000	-1.0000			
-1.7314	-4.2131			
0.0000	1.0000			
-2.0410	-7.6688			
-7.0781	-1.6048			
1.0000	-1.0000			

TABLE B

n	Itérations												
3													
6													
17													
24													
35													
52													
85													
113													
212													
640													
1365													
2048													

Corrigés

I. Programmation

Programme pour transformer des coordonnées cartésiennes (x,y) en coordonnées polaires (ρ, θ) sachant que :

$$\rho^2 = x^2 + y^2$$
$$\theta = \arctan(y/x)$$

avec

a. $\theta = \theta + \pi$ si $x < 0$

b. $\theta = \pi/2$ si $x = 0$ et $y > 0$

c. $\theta = -\pi/2$ si $x = 0$ et $y < 0$

d. θ n'existe pas si $x = 0$ et $y = 0$

---

réel *x, y, rho, theta*
écrire *``Introduisez l'abscisse x : ''*
lire *x*

```
écrire ``Introduisez l'ordonnée y : ''
lire y
rho = sqrt(x * x + y * y);
si x = 0 alors
        si y > 0 alors
                écrire ``rho = '', rho ,`` et theta = '', pi/2
        sinon
                si y < 0 alors
                        écrire ``rho = '', rho,`` et theta = '', -pi/2
                sinon
                        écrire ``rho = '', rho,`` et theta n'existe pas.''
                fsi
        fsi
sinon
        theta = arctan(y/x)
        si x < 0 alors
                theta = theta + pi
        fsi
        écrire ``rho = '', rho ,`` et theta = '', theta
fsi
```

```
PROGRAM CartesienPolaire;
VAR
 x, y, rho, theta, pi : REAL;
BEGIN
 WRITE('Introduisez l''abscisse x : ');
 READLN(x);
 WRITE('Introduisez l''ordonnée y : ');
 READLN(y);
 pi := 3.1416;
 rho := SQRT(x * x + y * y);
 IF x = 0 THEN
     IF y > 0 THEN
         WRITELN('rho =', rho:8:4 ,' et theta = ', pi/2:8:4)
     ELSE
         IF y < 0 THEN
             WRITELN('rho = ', rho:8:4 ,' et theta = ', -pi/2:8:4)
         ELSE
             WRITELN('rho = ', rho:8:4 ,' et theta n''existe pas.')
 ELSE
     BEGIN
         theta := ARCTAN(y/x);
         IF x < 0 THEN theta := theta + pi;
         WRITELN('rho = ', rho:8:4 ,' et theta = ', theta:8:4);
     END;
 READLN;
END.
```

TABLE A

x	y	ρ	θ (radian)	θ (degré)
8.4215	0.6239	8.4446	0.0740	4.2371
0.0000	0.0000	0.0000	-	-
3.2533	2.2733	3.9689	0.6099	34.9440
1.0000	0.0000	1.0000	0.0000	0.0000
0.0000	-1.0000	1.0000	-1.5708	-90.0000
-1.0000	1.0000	1.4142	2.3562	135.0001
1.0000	1.0000	1.4142	0.7854	44.9999
-4.2455	3.6998	5.6314	2.4248	138.9297
-12.6683	4.4839	13.4393	2.8014	160.5103
1.0000	1.0000	1.4112	3.9270	224.9999
-1.7314	-4.2131	4.5550	4.3225	247.6595
0.0000	1.0000	1.0000	1.5708	90.0000
-2.0410	-7.6698	7.9357	4.4523	255.0986
-7.0781	-1.6048	7.2577	3.3646	192.7749
1.0000	-1.0000	1.4142	-0.7854	-44.9999

II. Boucles

Ce programme n'a aucune utilité (pour l'instant en tout cas). Il montre seulement une propriété des nombres naturels pour laquelle on trouvera sûrement une application.

1. demander et lire un nombre *n* entier positif non nul.

2. si *n* est pair, il est divisé par 2.

3. si *n* est impair, il est multiplié par 3 et augmenté de 1.

4. arrêter quand *n* vaut 1 sinon repartir de l'étape 2.

Note : Il a été vérifié par ordinateur que tout nombre entier compris entre 1 et 1 099 511 627 776 était ramené à 1 après un nombre fini d'étapes. On attend toujours la découverte d'un nombre pour lequel cette propriété ne serait pas vraie. Exemple : *n* = 26 devient 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.

```
lire n
tant que n <> 1 faire
    si n mod 2 = 0 alors n = n / 2
```

```
sinon
        n = 3 * n + 1
fsi
écrire ``n = '', n
ftant
```

```
PROGRAM PropNaturel;
VAR
 n : LONGINT;
BEGIN
 WRITE(' Introduisez un entier n : ');
 READLN(n);
 WRITE(n,' donne');
 WHILE NOT(n = 1) DO
 BEGIN
     IF n MOD 2 = 0 THEN
         n := n DIV 2
     ELSE
         n := n * 3 + 1;
     WRITE(' ',n);
 END;
 READLN;
END.
```

TABLE B

n	Itérations												
3	10	5	16	8	4	2	1	-	-	-	-	-	-
6	3	10	5	16	8	4	2	1	-	-	-	-	-
17	52	26	13	40	20	10	5	16	8	4	2	1	-
24	12	6	3	10	5	16	8	4	2	1	-	-	-
35	106	53	160	80	40	20	10	5	16	8	4	2	1
52	26	13	40	20	10	5	16	8	4	2	1	-	-
85	256	128	64	32	16	8	4	2	1	-	-	-	-
119	940	170	85	256	128	64	32	16	8	4	2	1	-
212	106	53	160	80	40	20	10	5	16	8	4	2	1
640	320	150	80	40	20	10	5	16	8	4	2	1	-
1365	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
2048	1024	512	256	128	64	32	16	8	4	2	1	-	-

13.4 Programmation des boucles

Exercice 1 :

Ecrire un programme Pascal qui calcule la somme suivante (une seule boucle est suffisante) :

$$S = \sum_{i=0}^{i=n} (-1)^i \frac{x^{2i}}{(2i)!}$$

Compléter le tableau suivant :

n	x	S
4	2	
3	0	
7	0.75	
0	0	
10	1.5	
0	3.25	

Exercice 2 :

Ecrire un programme Pascal qui affiche à l'écran les 20 premiers nombres premiers

NB : faites en sorte que vos programmes soient lisibles (commentés et bien structurés)

Correction

Programme 1 :

```
program Somme ;
var
      den , n , i, sign: integer ;
      som, num, x      : real ;
begin
          (* lecture des variables *)
          writeln( 'n =  '); readln(n) ;
          writeln('x =  ' ); readln(x) ;

          (* initialisation des variables *)
          den := 1 ;    (* le dénominateur d'un terme de la somme *)
          num := 1 ;   (* le numérateur ---------------------------------*)
          som := 1 ;   (* la somme *)
          sign := -1 ;   (* le signe *)

          (* calcul de la somme *)
          for i := 1 to n do
          begin
                    den:= den * (2*i – 1) * (2*i) ;
                    num := num  * x * x ;
                    som := som  + sign* num  / den;
                    sign := -sign ;
          end ;
```

```
        (* Affichage du résultat *)
        writeln('S : = ', Som) ;
end .
```

n	x	S
4	2	-0.4159
3	0	1
7	0.75	0.7317
0	0	1
10	1.5	0.0707
0	3.25	1

Programme 2

```
program  Nb_Premiers ;
var
n , k , i : integer ;

begin
      writeln('voici les 20 premiers nombres premiers :' ) ;
      writeln(2) ;   (* le 2 est un nombre premier *)
      n := 3 ; (* commencer le test à partir de n =3 *) ;
      k :=1 ;   (* compteur de nombres premiers affichés*)
      while ( k< 20) do  (* chercher 20 nb premiers *)
      begin
            i :=2 ;  (* les diviseurs possibles de n sont i ∈ [2 , n div 2] *)
            while  ((i<= n div 2) and (n mod i <> 0)) do   i :=i+1 ;
              (*chercher un diviseur*)
            if  (i > n div 2) then  (*aucun diviseur de n n'a été trouvé*)
                  begin
                        writeln(n) ; (* afficher n *)
                        k :=k+1 ;
                        (* incrémenter le compteur de nombres premiers *)
                        n :=n+2 ;

                  (*successeur d'un nb premier est pair donc non premier*)
                  end
            else (* n est non premier *)
                  n :=n+1 ; (*continuer le test*)
      end ;
end .
```

13.5 Procédures – Fonctions et Tableaux

Ecrire un programme Pascal qui permet à l'utilisateur le choix d'exécution des différentes fonctions existantes dans le programme. A l'aide d'un menu affiché à l'écran, l'utilisateur doit pouvoir :

- Générer un tableau de valeurs aléatoires de taille n
- Afficher les éléments du tableau
- Trouver le plus petit élément du tableau
- Trier les éléments du tableau dans l'ordre décroissant

Utilisez trois procédures pour l'initialisation, l'affichage et le tri des éléments du tableau et une fonction pour la recherche de la valeur minimale. Cette fonction servira pour le tri du tableau.
Le programme principal édite le menu suivant :

Menu principal
1- Initialisation du tableau
2- Affichage des éléments
3- Valeur minimale
4- Tri des éléments
5- Fin

Choix :

Quelques procédures et fonctions standards utiles pour le programme

Génération de nombres aléatoires :

- randomize : procédure sans paramètre initialisant le générateur de nombres aléatoires
- Random(sup : entier) : fonction à un paramètre entier ; envoie un entier aléatoire n tel que $0 \leq n < sup$

Affichage :
Pour pouvoir utiliser ces procédures et fonctions mettre en début de programme (après program toto;) la commande : uses crt ;
- clrscr : procédure sans paramètre ; efface le contenu de l'écran et positionne le curseur au coin supérieur gauche

- goto(x,y) procédure à deux paramètres entiers $1 \leq x \leq 24$ et $1 \leq y \leq 80$; positionne le curseur au point de coordonnées (x,y) ; Le point supérieur gauche a comme coordonnés (1,1).

13.6 Programmation et boucles

I. Programmation

Il est maintenant temps de vous confronter à un problème complet et d'en résoudre toutes les étapes.

Comme tout le monde, vous avez besoin d'argent. Un banquier sympa (si, ça existe !) a mis en place un système d'emprunt à taux fixe dont le niveau des intérêts dépend de votre situation sociale. Le taux d'intérêt annuel nominal est de 9%. Une réduction de 2% est attribuée d'office aux 18-25 ans et aux étudiants de moins de 30 ans. Si, en plus, vous faites partie d'une grande école, vous bénéficiez d'une réduction supplémentaire de 1%. Les moins de 18 ans n'ont pas droit au prêt.

Ecrivez un programme qui calcule le montant du remboursement global à effectuer en partant du principe que tout client doit rendre la somme empruntée (avec les intérêts) au bout d'un an et en un seul paiement. Testez votre programme à l'aide de la table A.

N'oubliez pas de rester synthétique et de ne pas brûler les étapes. Prenez suffisamment de temps pour bien analyser le problème, et éventuellement le décomposer

en sous-problèmes. Cernez les entrées et les sorties, ainsi que l'usage précis que vous ferez des différentes variables. Une fois l'algorithme élaboré, le codage en

Pascal ne devrait pas vous poser trop de difficultés. Essayez de vous habituer à commenter le programme, et utiliser des noms de variables significatifs.

Table A

Somme	Age	Etudiant	Grande école	Somme à rembourser
15000	17	-	-	
15000	22	N	-	
15000	26	O	N	
20000	23	O	O	
17000	31	O	N	
22000	27	N	-	

II. Boucles

Il s'agit cette fois de comprendre les méandres d'un programme qui n'est pas de votre cru. Les plus courageux pourraient essayer de le faire tourner à la main, mais

une série d'exécutions en machine devrait vous être plus profitable. Une fois le principe compris et clairement expliqué, vous pourrez formuler des suggestions quant

à d'éventuelles modifications tant sur la forme que sur le fond. Faites attention à ne pas généraliser trop rapidement un comportement singulier.

```
PROGRAM Inconnu; USES WINCRT; VAR x,y,z,i : INTEGER; BEGIN
WRITELN ('Entrez x :'); READLN(x); WRITELN ('Entrez y :');
READLN(y); WRITELN ('Entrez z :'); READLN(z); i := x; WHILE i
<= y DO BEGIN IF (i mod z = 0) THEN WRITE(' ', i) ELSE
WRITE(' ',i mod z); i := i + 1; END; END.
```

1. Ce programme est très mal présenté, recopiez-le proprement en utilisant les règles d'indentation du Pascal.

2. Un bon moyen pour découvrir le but d'un programme est de le tester

3. Réécrivez le programme en le simplifiant et en ajoutant des commentaires.

Corrigés

I. Programmation

```
PROGRAM Emprunt;
 USES WINCRT;
 CONST
    NOMINAL = 9;    {Valeur de départ du taux d'intérêt}
 VAR
     somme, age, taux : INTEGER; {somme à emprunter, age de
                                  l'emprunteur et taux d'intérêt }
     etudiant, gde_ecole : CHAR; {situation sociale de l'emprunteur}
     total : REAL;   {Somme totale à rembourser}
 BEGIN
     {Saisie de l'âge de l'emprunteur}
     WRITELN ('Entrez votre age :');
     READLN(age);
     {Rejet des clients trop jeunes, et donc pas solvables}
     IF age < 18 THEN
        WRITELN('Vous n''avez pas droit au prêt.')
     ELSE
        BEGIN
            {Initialisation du taux à la valeur nominale}
            taux := NOMINAL;

            {Saisie de la somme à emprunter}
            WRITELN ('Entrez la somme à emprunter :');
            READLN(somme);

            {Saisie de la situation sociale de l'emprunteur}
            WRITELN('Etes-vous étudiant (O/N) ? ');
            READLN(etudiant);

            {Cas des étudiants de moins de 30 ans}
            IF ( etudiant = 'O') AND (age <= 30) THEN
               BEGIN
                  {Première réduction du taux}
                  taux := taux - 2;


                  {Demande de précision sur la situation sociale}
                  WRITELN('Etes-vous dans une grande école (O/N) ? ');
                  READLN(gde_ecole);


                  {Deuxième réduction du taux pour les étudiants des}
                  {grandes écoles   }
                  IF gde_ecole = 'O' THEN taux := taux - 1;
               END

            {Cas des moins de 25 ans non-étudiants}
            ELSE
               IF age <= 25 THEN taux := taux - 2;

            {Calcul et affichage de la somme à rembourser}
            total := somme + ((taux / 100) * somme);
            WRITELN('Vous devrez rembourser la somme de ', total:8:2);
        END;
   END.
```

Table A

Somme	Age	Etudiant	Grande école	Somme à rembourser
15000	17	-	-	-
15000	22	N	-	16050
15000	26	O	N	16050
20000	23	O	O	21200
17000	31	O	-	18530
22000	27	N	-	23980

II. Boucles

Voici le programme indenté :

```
PROGRAM Inconnu;
USES WINCRT;
VAR
    x,y,z,i : INTEGER;
BEGIN
    WRITELN ('Entrez x :');
    READLN(x);
    WRITELN ('Entrez y :');
    READLN(y);
    WRITELN ('Entrez z :');
    READLN(z);
    i := x;
    WHILE i <= y DO
    BEGIN
        IF (i mod z = 0) THEN
            WRITE(' ', i)
        ELSE
            WRITE(' ',i mod z);
        i := i + 1;
    END;
END.
```

Ce programme n'a strictement aucun intérêt. Il scrute un ensemble de valeurs entières entre deux bornes et affiche les valeurs divisibles par une valeur z quelconque, ou le reste de la division de ces valeurs par z quand cette division n'est pas entière. La seule chose importante à voir dans ce code est qu'on peut remplacer la boucle WHILE par une boucle FOR puisqu'on doit parcourir l'ensemble des valeurs de l'intervalle [x,y]. Le programme peut donc s'écrire :

```
PROGRAM Inconnu;
USES WINCRT;
VAR
    x,y,z,i : INTEGER;
BEGIN
    WRITELN ('Entrez x :');
    READLN(x);
    WRITELN ('Entrez y :');
    READLN(y);
    WRITELN ('Entrez z :');
    READLN(z);
    FOR i := x TO y DO   {On gagne deux lignes !}
```

```
    BEGIN
        IF (i mod z = 0) THEN
            WRITE(' ', i)
        ELSE
            WRITE(' ',i mod z);
    END;
END.
```

13.7 Boucle et Tableaux

Exercice 1 :

1. Ecrire un programme Pascal qui permet de :

i . Saisir des éléments réels dans un tableau. L'utilisateur est autorisé à saisir autant d'éléments qu'il souhaite.

ii . Ranger dans un autre tableau les éléments en position paire d'abord, puis ceux en position impaire dans le premier tableau

2. Proposer une méthode qui permet de classer les éléments du premier tableau dans l'ordre indiqué dans la question *ii* sans utiliser un deuxième tableau.

Exemple :

tableau initial 25.25 40.05 10.00 20.50 18.75 40.00 5.25 15.20

résultat 40.05 20.50 40.00 15.20 25.25 10.10 18.75 5.25

Exercice 2 :

- Ecrire un programme Pascal qui permet premièrement de remplir un tableau avec n nombres entiers générés aléatoirement et de supprimer en suite tous les nombres non premiers du tableau.Vous affichez sur ecran les résultat de chaque étape du traitement.

- Rédigez ce programme de manière à ce qu'il s'exécute de manière répétitive (boucle) jusqu'à ce que l'utilisateur spécifie la valeur zéro en entrée, indiquant ainsi que l'utilisateur veut quitter le programme.

NB : faites en sorte que vos programmes soient lisibles (commentés et bien structurés)

Corrigé

Programme 1

1.

```
Program positions;(* Ce programme classe les éléments d'un tableau suivant leur position*)
const
      nmax=100;   (* Ceux en position paire d'abord, ensuite ceux en position impaire *)
type
      tab = array[1..nmax] of real;
var
      t1, t2 : tab;
      n , i : integer ;
begin
      writeln('Combien d"éléments vous souhaitez ?? ) ;
      (* Saisie de la taille et des éléments du tableau *)
      readln(n);
```

```
    for i := 1 to n do (*lecture des éléments du tableau*)
    begin
        write('entrez l"élément', i, ': ');
        realdln (t1[i]);
    end;
    for i := 1 to n div 2 do   (* Classement des éléments*)
    begin
        t2[i] := t1[2*i] ;(* en position paire *)
        t2[n div 2 + i] := t1[2*i - 1] ;(* en position impaire *)
    end;
    writeln ('=======Résultat==========='); (* affichage du résultat *)
    for i := 1 to n do write(t1[i] , ' ');
    writeln ;
    for i := 1 to n do write(t2[i], ' ');
end.
```

2. Le même principe que la première question, il suffit d'effectuer des échanges d'éléments dans le même tableau plutôt que de faire des copies dans un autre tableau.

Programme 2

```
Program sup_premier; (* Ce programme supprime les nombres premiers
 dans un tableau *)
const
    nmax=100;  (* d'éléments générés aléaloirement *)
type
    tab = array[1..nmax] of integer;
var
    t : tab;
    n , i , k, j : integer ;
begin
    write('Combien d"éléments souhaitez-vous (< 100) ?  ') ;
    readln(n);
                        (*génère aléaloirement et affiche les éléments du tableau*)
    randomize;
    Clrscr; writeln('Un tableau de ', n,' elements :) ;
    for i := 1 to n do begin
        t[i] : = random(100);
        writeln('t[',i,'] = ', t[i]);
    end;
    i :=1;
    while (i<=n) do(* arret quand tous les éléments on été testés *)
    begin
        k :=2 ;          (* les diviseurs possibles de t[i] sont k dans l'intervalle [2 , t[i] div 2] *)
        while ((k<  t[i] div 2) and (t[i] mod k<> 0)) do k :=k+1 ;  (*chercher un diviseur*)
        if (k = t[i] div 2) then            (*aucun diviseur de n n'a été trouvé*)
            begin
                for j : = i to n-1 do   t[i] : = t[i+1];
                                        (*suppression de l'élément par decalage*)
                n :=n-1;
                                        (*decrementer la taille du tableau *)
            end
            else
                                (* t[i] est non premier *)
                i:=i+1 ;
                                (*tester l'élément suivant *)
    end ;
    writeln('Voici les elements non premiers du tableau');
                                (* affichage des éléments restants du tableau*)
    for i := 1 to n do writeln('t[',i,'] = ', t[i]);
end.
```

13.8 String et Tableaux

Exercice 1 :

i. Ecrire une procédure ***FREQ*** qui détérmine les fréquences des lettres de "a" à "z" d'un texte de moins de 255 caractères.

ii. Ecrire une fonction **DEL** qui supprime tous les espaces dans le texte.

iii. Ecrire une procédure **DIV** qui découpe ce texte en une série de chaines de caractères (<25 caractères) et les range dans un tableau *"chaines"*.

iv. Proposer une méthode simple et rapide **TRI** pour trier ce tableau dans un ordre croissant.

v. Ecrire une fonction **FUS** qui fusionne deux tableaux de chaines de caractères dèja triés par la méthode proposée en iv.

- Toutes les procédures et fonctions doivent avoir des arguments. Justifier votre mode de passage de paramètres (par adresse ou par valeur).
- Le programme principal ne doit comporter que des appels de fonctions et procédures.
- Penser à des procédures d'affichage visualisant le résultat pour chaque réponse.
- Le jeu d'essai doit suivre les étapes suivantes :
    - afficher le texte saisi et les fréquences de chaque lettre
    - afficher le texte avant et après la suppression des espaces
    - afficher le texte avant et après le découpage en chaines de moins de 25 caractères
    - afficher le tableau avant et après le tri
    - afficher les deux tableaux avant et après la fusion

NB : faites en sorte que vos programmes soient lisibles (commentés et bien structurés)

Corrigé

```
program sort;
uses wincrt;
type
      frequence = array['a'..'z'] of integer;  (* tableau d'entier indexé par caractère*)
      texte = string[255];
      chaines = array[1..50] of texte
var
      FR : frequence;
      CH : texte;
      T_ch : chaines;
      choix : char;

{==================PROCEDURE SAISIE ================}
 procedure Saisie(var S:texte);
{-=================================================}
begin
      write('Entrer le texte (<255 caracteres) : readln(S);    (* lecture du texte *)
end;

{==================PROCEDURE FREQUENCE================}
 procedure FREQ(S:texte ; var F : frequence);
```

```
{-============================================================}
var
      i : char;
      j : integer;
begin
      write('Entrer le texte (<255 caracteres) : readln(S);
      for i := 'a' to 'z' do F[i] := 0;      (* initialisation du tableau des fréquences*)
      for i := 'a' to 'z' do
            for j := 1 to length(S) do
                  if (S[j] = i) then F[i] := F[i] +1;
                                          (* incrémenter le nombre de fréquence de i *)
            writeln('Les frequence des letteres :') ; (* affichage du résultat*)
            for i := 'a' to 'z' do writeln(i,' ===> ', F[i]);
end;
{====================PROCEDURE DEL ESPACE=================}
function DEL(S:texte) : texte
{-============================================================}
var
      i , l : integer;
begin
      writeln( 'Texte avant suppression : ', S);
      l :=length(S);
      for i:= 1 to l do
      begin
            if S[i]="" then
                        begin
                              S := delete(S,i,1);  (* suppression du caractère espace*)
                              l := l -1;(*décrémenter la taille de la chaine *)
                              i := i-1;
                        end;
      end;
      DEL:=S;
end;
{=============PROCEDURE AFFICHAGE DES ELEMENT================}
procedure AFF(taille : integer ; t : table);
{============================================================}
var
      i: integer;
begin
      clrscr;
      writeln('les éléments du tableau : ');
      for i := 1 to taille do write(t[i],'   ');
end;
{============PROCEDURE DECOUPAGE EN CHAINES==============}
procedure DIV(S : texte ; var T : chaines);
{============================================================}
var
      i : integer;
begin
      i:=1; j:=1;
      while (i<length(S)) do
      begin
            T[j] := copy(S,i,8); (* extraire 8 caractères de la chaine *)
            j := j+1;
            i := i+8;
      end;
end;
{====================PROCEDURE ECHANGE=====================}
procedure echange (i , j : integer ;  var t : table ):
{============================================================}
```

```
var
      c: integer;
begin
      c := t[i];
      t[i] := t[j];
      t[j] : = c;
end;

{==================PROCEDURE TRI DECROISSANT=================}
procedure tri (taille : integer ; var t: table);
{============================================================}
var
      i, j,  Imin: integer;
begin
      for i:= 1 to taille -1 do
      begin
            Imin:= i;
            for j:= i+1 to taille do
                                    (* recherche de l'indice (>i) du +petit élément du tableau *)
                  if t[j]<T[Imin] then Imin := j ;
            echange(i,Imin,t);     (* echange avec l''lément de la position courante i*)
      end;
end;

{===============PROCEDURE FUSION DE 2 TABIEAUX ================}
procedure FUS (t1, t2: table ; taille1, taille2; integer; var t : table, var taille);
{============================================================}
var
       i, j : integer;
begin
      i:=1; j:=1; k:=1;
      while (i<=taile1) and (j<=taille2) do
      if (t1[i]<t2[j]) then
      begin                                    (* copier les éléments du t1 tant qu'ils*)
            t[k] : = t1[i];                    (* sont inférieurs a l'élément courant de t2*)
            i:=i+1;
             k:= k+1;
      end
      else
      begin                                    (* et inversement*)
             t[k]:= t2[j];
             j:=j+1;
             k:=k+1;
      end;

      if (i>taille1) then                (* recopier des éléments  de t2, s'il en reste*)
      for k1 := j to taille2 do
      begin
            t[k]:= t2[k1];
            k:=k+1;
      end
      else                                     (* idem pour t1*)
      for k1 := i to taille1 do
      begin
            t[k]:= t1[k1];
            k:=k+1;
      end;
end;
```

```
{==================PROGRAMME PRINCIPAL===============}
BEGIN
  repeat
clrscr;
gotoxy(24,9); writeln('Menu Principal ');
gotoxy(22,10);writeln(' ======================================');
  gotoxy(25,11);writeln(' 1. saisie du texte ' );
  gotoxy(25,12);writeln(' 2. fréquence des lettres');
  gotoxy(25,13);writeln(' 3. Suppression de espaces ');
  gotoxy(25,14);writeln(' 4. Decoupage en chaines');
  gotoxy(25,15);writeln(' 5. Tri du tableau de chaines ');
gotoxy(22,16);writeln(' 6. Fusion de deux tableaux ');
gotoxy(22,17);writeln(' 7. Fin ');
gotoxy(30,18);write(' ======================================');
gotoxy(30,19);write(' Votre choix : ');
readln(choix);
case choix of
'1' : saisie(CH);
'2' : FREQ(CH, FR);
'3' : writeln(' Apres suppression', Del(CH));
'4' : DIV(CH , T_ch);
'5' : Tri(T_ch);
'6' : FUS(T_ch1, Tch_2, Tch3);
  end;
 until choix = '5';
END.
```

13.9 Les boucles

Exercice 1 :

Ecrire un programme Pascal qui cherche et affiche les nombres de trois chiffres (compris entre 100 et 999) tels que la somme des cubes des trois chiffres soit égale au nombre.

Exemple : $153 = 1^3 + 5^3 + 3^3$

Exercice 2 :

La suite de Newton permet de calculer de manière itérative, la racine carrée d'un nombre A. La suite est définie par :

$$U_i = \frac{\left(U_{i-1} + \frac{A}{U_{i-1}} \right)}{2}$$

$$U_0 = \frac{A}{2}$$

Ecrire un programme Pascal qui calcul la racine carrée d'un réel. Le calcul doit s'arrêter lorsque La condition suivante est satisfaite :

$$\left| U_i - U_{i-1} \right| \leq \text{Epsilon}$$

avec :

Epsilon : précision du calcul

Le programme doit afficher la valeur de la racine carrée et le nombre d'itération nécessaire pour le calcul.

Conseil : utilisation de l'instruction "repeat".

Vous compléterez ensuite la table des résultats suivante :

A	Epsilon	$\sqrt{A}$	Nbr d'itérations
144	0.1		
529	0.01		
4588.654	0.001		
35435.7584	0.0001		

13.10 Boucle et Tableaux

EXERCICE 1 :

Faire un programme qui lit au clavier les coordonnées de deux points, *A* et *B*, situés dans l'espace à *N* dimensions, puis qui calcule et affiche leur distance :

$$d = \sqrt{\sum_{1}^{N} (A_i - B_i)^2}$$

On exécutera ce programme pour deux points du plan, puis pour deux points de $\mathbf{R}^3$

EXERCICE 2 :

Soit une matrice entière *X* de dimension *N x M* triée à la fois dans le sens des lignes et des colonnes, c'est-à-dire telle que :

$$X[i,j] \leq X[i,j+1]$$

$$X[i,j] \leq X[i+1,j]$$

Exemple : N = 3, M = 4

1	3	7	14
2	5	9	15
5	8	13	19

Déterminer si un nombre donné *a* est présent dans l'ensemble des éléments *X[i,j]*.

1. Implanter la solution en Pascal.

2. Améliorer votre programme, s'il vous reste du temps.

13.11 Chaines de caractères et Tableaux

1. *Ecrire une procédure **Lecture** qui permet de lire un texte, phrase par phrase, en boucle, jusqu'à confirmation de l'utilisateur pour la fin de saisie, ou que le nombre maximum de 1000 caractères soit atteint.*

2. *Ecrire une procédure **Réduction** qui permet de transformer une chaîne de caractère résultant de (1.) en majuscules. Tout caractère autre que les caractères alphabétiques sont remplacés par des blancs.*

3. *Ecrire une procédure **remplir_tableau** qui permet de remplir un tableau de mots par la série des mots de la phrase réduite. L'ensemble du texte doit se trouver dans ce tableau.*

Remarque : le tableau peut comporter jusqu'à 100 mots de 20 caractères au maximum par mot.

4. *Ecrire une fonction **occurrence** qui permet de calculer le nombre d'occurrence de chaque mot du texte.*

5. *Ecrire une procédure **Tri** qui permet de trier le tableau dans l'ordre alphabétique des mots.*
6.
7. *Ecrire une procédure **affichage** qui permet d'afficher la liste des mots par ordre alphabétique tel que chaque mot est suivie de son occurrence.*

- Toutes les procédures et fonctions doivent avoir des arguments. Justifier votre mode de passage de paramètres (par adresse ou par valeur).
- Le programme principal ne doit comporter que des appels de fonctions et procédures. Une procédure ou fonction peut appeler une autre procédure ou fonction.
- Le jeu d'essai doit suivre les étapes suivantes :
- afficher le texte saisi
- afficher le texte réduit
- afficher la liste des mots par ordre alphabétique, où chaque mot est suivi de son occurrence

NB : faites en sorte que vos programmes soient lisibles (commentés et bien structurés)

14 Annales d'examens Médians

Problème n°0 (6,5 points) :

Ecrire un programme Pascal qui construit une matrice carrée NxN (N vaut au maximum 10, la valeur de N étant saisie par l'utilisateur) dans laquelle le carré le plus externe ne contient que des 1, le carré interne voisin du carré externe ne contient que des 2, et ainsi de suite...

Exemple : si N=6, la matrice qu'on veut construire est :

```
1 1 1 1 1 1
1 2 2 2 2 1
1 2 3 3 2 1
1 2 3 3 2 1
1 2 2 2 2 1
1 1 1 1 1 1
```

Problème n°1(6,5 points) :

Soit le programme Pascal suivant (qui est très mal présenté !) :

```
Program inconnu;
var i:string;j:integer;k:integer;a:char;
begin readln(i); j:=length(i);
for k:=1 to (j div 2) do
begin
a:=i[k];i[k]:=i[j-k+1];i[j-k+1]:=a; end;
writeln(i);
end.
```

Q1 : Si l'utilisateur d'un tel programme saisit la chaîne de caractère 'alpha' dans la variable i :
a/ Que vaut `j div 2` ?
b/ Donner pour toutes les valeurs de k les valeurs respectives de `a`, `i[k]` et `i[j-k+1]`

Q2 : Que fait ce programme ? Expliquer !

Q3 : Réécrire ce programme en donnant des noms d'identificateurs signifiant quelque chose à la lecture du programme et en le présentant correctement.

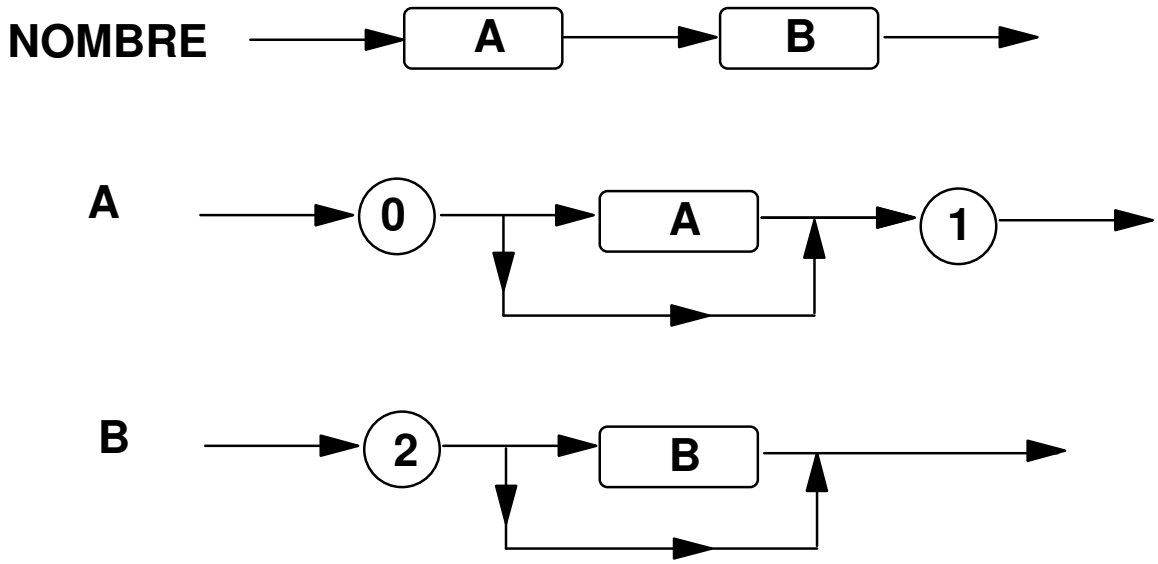
NB : On rappelle que la fonction `length` retourne la longueur de la chaîne de caractère fournie en paramètre.

Problème n°2 (7 points) :
Convertir en décimal les nombres suivants : 11100110_2, 10111011_2, 53218_8, $F78C_{16}$
Convertir en binaire les nombres suivants : 321_{10}, 2735_8, $8A3B5F_{16}$
Convertir en octal les nombres suivants : $ABCDE_{16}$, 10110011_2, 1025_{10}, 1024_{10}, 1023_{10}, 693_{10}
Convertir en hexadécimal les nombres suivants : 11100110_2, 76543_8, 897_{10}

Problème n°3 :

a/ Langage (2 points) *10 mn*

Définir le langage exprimé par le diagramme suivant :



Que génère cette grammaire ? Donner des exemples de phrases acceptées par ce langage ?

b/ Numération (4 points) *20 mn*

Convertir en base 10 les nombres suivants : 11111110_2, 11011101_2, 1265_8, $ADBC_{16}$

Convertir en binaire les nombres suivants : 1025_{10}, 1023_{10}, 129_{10} , 7684_8, $F32E3D_{16}$

Convertir en octal les nombres suivants : $2ABC9_{16}$, 10011011_2

Convertir en hexadécimal les nombres suivants : 11001011_2, 56734_8

Problème n° 4 :

A/ Que fait le programme suivant ? (3 points) *10 mn*

```
program abc;
var a, b, c, temp  : integer;
begin write('a=');  readln(a);
write('b=');  readln(b); write('c=');  readln(c);
if b > a then begin temp := a;
a := b; b := temp;end;
if c > a then  begin
temp := a; a := c;  c := temp;end;
if c > b then begin
temp := b; b := c;
c := temp;end;
writeln(a, b, c); end.
```

Donner un exemple de données d'entrée et de sortie.

B/ Algorithme (3 points) *10 mn*

Ecrire un **algorithme** qui demande une valeur V (réelle) comprise entre 1 et 2, et qui calcule le plus petit N (entier) tel que: $1 + 1/2 + 1/3 + ... + 1/N > V$

Problème n°5 (8 points) : Je calcule mes impôts ***35 mn***

Je voudrais calculer les impôts de ma famille. Soit MI le montant imposable de ma famille (montant imposable de mon conjoint et de moi-même). Je dois d'abord calculer RI le revenu imposable, en appliquant d'abord un abattement de 10 % sur MI, puis un abattement de 20% sur le résultat précédent.

Soit N le nombre de parts :

N = 1 si célibataire
N = 2 si couple
N = 2,5 si couple avec 1 enfant
N = 3 si couple avec 2 enfants
N = 4 si couple avec 3 enfants
N = 5 si couple avec 4 enfants
....

Soit QF le quotient familial : QF = RI / N

L'impôt à payer I est calculé de la manière suivante :

I = 0	si	QF < 25610
I = (RI * 0,105) - (2689 * N)	si	25610 < QF <= 50380
I = (RI * 0,24) - (9490 * N)	si	50380 < QF <= 88670
I = (RI * 0,33) - (17470 * N)	si	88670 < QF <= 143580
I = (RI * 0,43) - (31828 * N)	si	143580 < QF <= 233620
I = (RI * 0,48) - (43509 * N)	si	233620 < QF <= 288100
I = (RI * 0,54) - (60795 * N)	si	288100 < QF

1 - Ecrire un **programme PASCAL** permettant de calculer l'imposition d'une famille ou d'un célibataire.

2 - Sachant que les salaires augmentent en fonction du coût de la vie (1,5% par an) et que l'on peut espérer une augmentation de 500 F tous les deux ans, calculer l'imposition de cette famille sur les 20 ans à venir (on ne considérera pas de changement de N).
Le **programme** écrira le résultat sous la forme :

```
Imposition année 1 : 12980 F
Imposition année 2 : 13380 F
Imposition année 3 : 13580 F
....
Imposition année 20 : 20765 F
```

Problème n°6 (6 points) : Les tableaux

Un supermarché veut informatiser la gestion du prix de 20 articles différents. Chaque article a un prix d'achat et une marge bénéficiaire (en %) spécifiques. Le gérant désire placer ces articles dans un tableau de manière à être en mesure de connaître le prix à afficher en magasin, celui-ci tenant compte de la marge bénéficiaire. Les colonnes du tableau sont : le nom de l'article, le prix d'achat, la marge bénéficiaire et le prix affiché. Dans un but de simplification, la marge bénéficiaire sera la même pour chaque article.

1. Faire la déclaration du tableau en PASCAL.

2. a. Donner l'algorithme qui permet de saisir les 20 articles (prix d'achat et marge bénéficiaire).

   b. Donner l'algorithme qui permet de calculer pour tous les articles le prix à afficher.

3. Faire le programme Pascal associé à l'algorithme de la question 2-a.

4. Écrire un programme affichant les noms des articles, suivis de leurs prix affichés.

Problème n°7 (7 points) : Numération

Convertir en décimal les nombres suivants : 10000001_2, 11100011_2, 953_8, $D1FEC5_{16}$
Convertir en binaire les nombres suivants : 957_{10} , 452368, $1C2F3A_{16}$
Convertir en octal les nombres suivants : $7B254_{16}$, 11011101_2, 513_{10}, 512_{10}, 511_{10}, 753_{10}
Convertir en hexadécimal les nombres suivants : 10000101_2, 14738, 8092_{10}

Problème n°8 (7 points) : Programme inconnu

Une personne un peu tête en l'air a écrit ce programme il y a 6 mois pour faire un travail. Puis elle l'a oublié dans un coin. Pourriez-vous l'aider à trouver ce qu'il fait ?

```
program inconnu ; const d = 0.01; var i: integer; a, b, c, e: real;
begin write('entrez a : '); readln(a); i := 0; e := a; c := (1 + a) / 2;
writeln('i : ', i : 2, '  c : ', c : 8 : 6); repeat i := i + 1; b := c;
c := (b + a / b) * 0.5; e := (abs(c - b) / b);
writeln('i : ', i : 2, '  c : ', c : 8 : 6, ' e : ', e : 8 : 6); until (e < d);
writeln(' FIN, i : ', i : 2, '  c : ', c : 8 : 6, ' e : ', e : 8 : 6); end.
```

Question 1 : (1 point)

Ce programme est très mal présenté, en effet l'impression ne marche plus très bien. Réécrivez-le proprement en utilisant les règles d'indentation du pascal.

Question 2 : (1 point)

Un bon moyen pour savoir ce que fait un programme est de le faire fonctionner manuellement une fois. Donnez exactement les affichages que fait le programme quand on le fait fonctionner en saisissant la valeur 9.

Question 3 : (1 point)

Vous commencez probablement à avoir une petite idée de ce que fait ce programme. Mais un deuxième essai est nécessaire. Donnez exactement les affichages que fait le programme quand on le fait fonctionner en saisissant la valeur 2.

Question 4 : (1 point) Que fait ce programme ?

Question 5 : (1 point)

Vous n'êtes probablement pas comme cette personne et vous ne commettez jamais deux fois la même erreur. Réécrivez le programme en donnant des noms d'identificateurs qui veulent dire quelque chose et en ajoutant des commentaires.

Question 6 : (1 point) Donnez la formule de récurrence qu'il utilise.

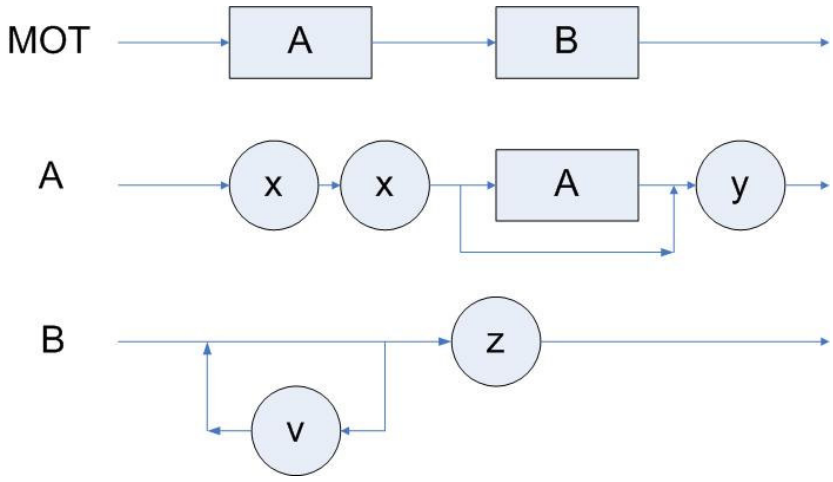
Question 7 : (1 point)

Soit la formule de récurrence suivante : $x_{n+1} = x_n + \frac{1}{3} \left(\frac{a}{(x_n)^2} - x_n \right)$

Modifiez le programme en utilisant cette formule correctement.

Problème n°9 (6 points) : Diagrammes de Conway

Soit les diagrammes suivants :



Définir le langage exprimé par ces diagrammes.
Indiquez à quoi correspond ce langage, par une formule si possible.
Donner des exemples de phrases acceptées par ce langage et des exemples de phrases refusées.

Problème n°10 (7 points) : Fibonacci

Ecrire un algorithme puis un programme Pascal qui lit un entier n positif et non nul, puis calcule le $n_{ème}$ nombre de la suite de Fibonacci. La suite se définie ainsi :
F(0) = 0, F(1) = 1 et F(i) = F(i-1) + F(i-2) pour i > 1.

Calculez également la somme des n premiers nombres de la suite, *(S = F(1) + F(2)… + F(n))* ainsi que la valeur moyenne de ces nombres.

Ecrire l'algorithme et le programme Pascal correspondant.

Problème n°11 (7 points) : Programme inconnu

Il s'agit de comprendre les méandres d'un programme qui n'est pas de votre cru. Les plus courageux pourraient essayer de le faire tourner à la main. Faites attention à ne pas généraliser trop rapidement un comportement singulier.

```
PROGRAM Inconnu; VAR n : LONGINT; BEGIN WRITE(' Introduisez un entier n :
'); READLN(n); WRITE(n,' donne'); WHILE NOT(n = 1) DO BEGIN IF n MOD 2 = 0
THEN n := n DIV 2 ELSE n := n * 3 + 1; WRITE(' ',n); END; READLN; END.
```

1. Ce programme est très mal présenté, recopiez-le proprement en utilisant les règles d'indentation du Pascal.

2. Un bon moyen pour découvrir le but d'un programme est de le tester (compléter la table B). On vous demande d'écrire la valeur de la variable n pour chaque itération. Chaque ligne du tableau correspond à une valeur initiale différente pour n. Chaque colonne correspond à une itération.

3. Que fait ce programme ?

4. Réécrivez le programme en ajoutant des commentaires. !

TABLE B

n	Itérations												
3													
6													
17													
24													
35													
52													
85													
113													
212													
640													
1365													
2048													

Problème n°12 (6 points) : Conversions

Effectuez les conversions demandées. On justifiera en expliquant clairement chaque conversion. La base de chaque nombre à convertir est indiquée en indice.

a/ Convertir en base 2 les nombres suivants : 127_{10}, 377_8, FF_{16}, 49_{10}
b/ Convertir en base 10 les nombres suivants : 382_8, 207_8, $FA3_{16}$, 110001_2
c/ Convertir en base 8 les nombres suivants : 58_{16}, 23_{10}, 333_4, 127_{10}
d/ Convertir en base 11 les nombres suivants : 56_8, 22_{10}, 133_{10}, EFH_{16}

Problème n°13 (4 points) : Diagramme de CONWAY

On vous demande de dessiner les diagrammes de CONWAY qui permettent de vérifier la grammaire suivante : aa+, ba+, ab+, bb+, aa-, ab-, ba-, bb-, aaa++, aba+-, bab-- ; soit une suite de signes a ou b, suivie d'une suite de signes + ou -. On remarquera cependant qu'il y a toujours un signe de type 'a' ou 'b' de plus que de signe '+' ou '-'.

Problème n°14 (10 points) : Programme inconnu

Soit le programme Pascal suivant (qui est très mal présenté !) :

```
Program inconnu;
var i,j : integer; a,d,c,b,m :longint;
begin read(i,j);m:=0; a:=1;
for b:=1 to j do  a:=a*i; d:=1;
writeln('étape 1 : valeur de a et de d :',a:6,d:6);
for c:=1 to a do
d:=d*c; writeln('étape 2 : valeur de a et de d :', a:6,d:6);
d:=1;a:=0;for b:=1 to j do d:=d*b;
writeln('étape 3 : valeur de a et de d :', a:6,d:6);
```

```
a:=1; for b:=1 to d
do a:=a*i; writeln('étape 4 : valeur de a et de d :', a:6,d:6);
end.
```

Q1 : Remplir les tableaux ci-dessous, afin d'indiquer les valeurs des variables à différentes étapes du programme, avec les données proposées en tête de tableau. Ne pas oublier d'indiquer votre nom.

Q2 : Réécrire ce programme en donnant des noms d'identificateurs signifiant quelque chose à la lecture du programme, en le présentant correctement et en y ajoutant des commentaires.

Q3 : Identifier les parties communes du programme et essayer de décrire ce qu'elles font.

Q4 : A quoi correspond chaque `writeln` lors des 4 étapes ? Quel est le calcul présenté ?

Q3 : Que fait ce programme dans son ensemble ? A quoi correspond-il ? Expliquer !

Voici les tableaux que vous devez remplir pour répondre à la question Q1 :

i =2 et j=3	**a**	**b**	**c**	**d**
étape 1				
étape 2				
étape 3				
étape 4				

i =3 et j=2	**a**	**b**	**c**	**d**
étape 1				
étape 2				
étape 3				
étape 4				

Problème n°15 (6 points) : Exponentiation indienne

On se propose de calculer x puissance y (noté x^y avec x et y des entiers) sachant que l'on ne dispose que de l'élévation au carré, de la division euclidienne par 2 et de la multiplication de deux nombres, **le tout en minimisant le nombre d'opérations**.

Dans le cas où y est pair les facteurs de x sont regroupés deux à deux. Prenons un exemple : x^8

x*x	x*x	x*x	x*x	y=8
x^2 * x^2		x^2 * x^2		y=4
x^4 * x^4				y=2
x^8				y=1

Il n'y a donc que 3 multiplications à effectuer : x*x, x^2 * x^2 et x^4 * x^4
contre 7 dans le calcul classique : x*x*x*x*x*x*x

Dans le cas ou y est impair on se ramène au cas précédent. Exemple avec x^5

x*x	x*x	x	y=5 ramené à y=4
x^2 * x^2			y=2
x^4			
x^5			y=1

1) Donner l'algorithme qui calcul la puissance x^y grâce à cette méthode, les valeurs de x et de y étant entières et à demander à l'utilisateur.

2) Exécuter cet algorithme dans les deux cas de figure suivants : x=3, y=4 et x=2, y=10. On donnera dans un tableau la valeur des variables dans leur état initial et à la fin de chaque boucle de calcul.

3) Traduire cet algorithme en Pascal.

Problème n°16 (3 points) : Langage

On désire donner un définition syntaxique du langage constitué des mots suivants :
acb, aaacbbb, c, aacbb, aaaaaacbbbbbb.

1) Donner une formule pour définir ce langage.

2) Définir le diagramme de Conway associé.

Problème n°17 (5 points) : Boucles

Ecrire le programme permettant d'afficher figure suivante :

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

L'utilisateur doit préciser le nombre (toujours impair) de lignes et de colonnes souhaitées. Il y a toujours autant de lignes que de colonnes (il s'agit d'une matrice carrée). Dans notre exemple, ce nombre est 11.

Problème n°18 (6 points) : Expressions

Une expression est une combinaison d'opérandes (variables et constantes), d'opérateurs et de fonctions, comme par exemple : "i+1", "2.08E3 * x" ou encore "(x2) OR (x<8)" .

Pour évaluer des expressions, il suffit d'utiliser les règles de décomposition syntaxique inscrites dans les diagrammes de Conway du langage Pascal.
Ainsi, par exemple, l'expression "a*b+c"se décompose en :

```
 Expression => Expression simple
       Expression simple => Terme + Terme
=> Facteur * Facteur  +  Facteur
                            =>  variable * variable +  variable
```

donc "a*b+c"est correct en Pascal. De plus, cette expression correspond à la somme de la variable c avec le résultat du produit de la variables a par la variable b.

Utiliser les diagrammes de Conway et les règles de décomposition syntaxique pour déterminer si les expressions suivantes sont correctes en Pascal. Préciser, en cas de succès, l'ordre de priorité des opérations

1) `2*sqrt(5+x)+a*b-6`
2) `a + ord((i)*10)  < sqr(b*i+a)`
3) `sqr(x*sqr(y+15)) > sqrt(x+y)`

15 Annales d'examens Finaux

Problème n°1 (7 points) : ***(environ 45 mn)***

On dispose de deux fichiers de texte, contenant chacun une liste de noms et prénoms d'étudiants, ainsi qu'une moyenne pour chaque étudiant. Chaque fichier correspond en fait à une matière (ou une UV). Les données sont rangées en lignes, chaque ligne correspondant à un étudiant. Une ligne comporte le nom, le prénom (séparé par un espace) et la moyenne (séparée également par espace), selon le format suivant :

```
Dupond Paul 12.5
Durand Pierre 8
Mozitard Julie 15
Onagirt François 16.25
```

Chaque fichier est déjà trié, par ordre alphabétique sur le nom de l'étudiant. Ces fichiers peuvent comporter des noms et des prénoms communs, c'est à dire qu'un même étudiant peut être enregistré dans les deux fichiers, avec une moyenne éventuellement différente (il s'agit de matières distincts). On peut également trouver des étudiants ayant le même nom, mais un prénom différent (exemple : 'Bennani Youssef' et 'Bennani Younes' ou 'Joly Fabien' et 'Joly Guilhem'). Les fichiers se terminent tous deux par une ligne composée uniquement de blancs.

Nous désirons écrire un programme effectuant la fusion des deux fichiers. On fabriquera un troisième fichier de texte, contenant une seule liste d'étudiants, triée par ordre alphabétique sur le nom, et comportant pour chaque ligne le nom de l'étudiant, son prénom (séparé par un espace), ainsi que sa note globale, recopiée à partir des deux fichiers (séparée par un espace). Dans le cas d'un étudiant présent dans les deux fichiers, il faudra calculer la moyenne générale des deux notes obtenues. Dans le cas contraire, il suffira de recopier la note de l'étudiant, à partir du fichier concerné.

1. Ecrire une procédure permettant de lire des nombres réels, écrits sous forme de caractères (chiffres séparés éventuellement par un point), et de convertir ces caractères en un nombre réel (type `real`).

2. Définir l'algorithme général de la fusion des deux fichiers

3. Préciser le traitement dans le cas de deux personnes identiques

4. Ecrire une procédure de conversion de réels en caractères

5. Réaliser le programme en entier, en utilisant les procédures définies précédemment. Bien identifier les types de données, la partie lecture/écriture, ainsi que la fusion.

Problème n°2 (6 points) : récursivité ***(environ 30 mn)***

1. Ecrire une procédure RECURSIVE `concat_deux_chaines` qui aura trois paramètres en arguments:

`chaine1, chaine2` de type `string` et `longueur` de type `integer`

La procedure devra concaténer `chaine1` et `chaine2` dans `chaine2`.
`Longueur` aura comme valeur avant et après l'appel, la longueur de la chaine1
(`Longueur` évoluera en fonction de `chaine1`).
Pour cela on devra IMPERATIVEMENT utiliser la procédure `ajout` supposée déjà définie :

avant appel

```
           chaine1 = "aaaa"
chaine2 = "bbbb"

ajout(chaine1,chaine2)
```

après appel

```
           chaine1 = "aaaab"
chaine2 = "bbb"
```

2. Ecrire un exemple d'utilisation avec :
   chaine1 = "impec" et
   chaine2 = "cable"

Problème n°3 (7 points) : Voyage ***(environ 45 mn)***

Un voyageur, devant visiter N villes (N≤20), désire le faire au moindre coût. Ce voyageur averti utilise l'algorithme suivant : à partir d'une ville, il se rend à la ville la plus proche qu'il n'ait pas encore visitée. On veut écrire un programme PASCAL qui mette en œuvre cet algorithme.

Les distances entre villes, prises deux à deux, sont enregistrées dans un tableau à deux dimensions. On pourra à cet effet adopter la déclaration commune suivante :

```
var RESEAU : array [1 . . 20, 1 . . 20] of real ;
```

1. Faire une procédure d'initialisation de RESEAU où l'utilisateur introduit lui-même les distances entre ces N villes. On supposera que le graphe est entièrement connecté, toute ville étant reliée à toute ville. On remarque que la matrice ainsi obtenue est ***symétrique***, car la distance d'une ville A à une ville B est identique à celle entre la ville B et la ville A.
3. Faire une procédure qui détermine, en appliquant l'algorithme énoncé ci-dessus, une solution au problème en énonçant successivement les villes visitées et qui précise les distances cumulées. On supposera qu'il part de la ville n°1.

Problème n°4 (7 points) : Où est passé la 7ème Compagnie ? ***(environ 45 mn)***

Lors de la seconde guerre mondiale, la 7ème Compagnie avait eu de gros problèmes d'orientation lorsqu'elle s'est retrouvée au milieu des lignes allemandes. De façon à obvier à ce genre de désagrément, le Colonel Urbain Théodore Constant aimerait que vous informatisiez un certain nombre de tâches inhérentes à la localisation des compagnies de son régiment et notamment de la 7ème Compagnie.

La 7ème Compagnie est commandée par le Capitaine Théophile Perron assisté de son fidèle Lieutenant Frédéric Dominici (Commandant Adjoint) et du Sous-lieutenant Christophe Hamel (Officier d'instruction). Cette compagnie est composée de quatre sections dirigées respectivement par les Aspirants Ghislaine Monchy-Humière et Bruno François, et les Adjudants Laurent Salant et Didier Von Moeschler. Au sein d'un régiment, chaque compagnie possède un numéro, un commandant, un commandant adjoint, un officier d'instruction, différentes sections dont le nombre varie (au maximum : huit ; pour la 7ème Compagnie : quatre), et différentes positions en fonction de dates. Chaque position est repérée par une coordonnée (latitude, longitude). On ne conserve en mémoire que les dix dernières positions d'une compagnie, ceci au sein d'un tableau contenant pour chaque position sa date et ses coordonnées. Par exemple, le 04/06/96 la 7ème Compagnie se trouvait à la position 45.3° (latitude), 5.1° (longitude). Chaque section comporte un numéro, un chef, un adjoint et vingt soldats. Par exemple, la section n°1 est dirigée par l'Aspirant Ghislaine Monchy-Humière assistée du Chef Thibault Monestier, son adjoint, et comporte vingt soldats. Chaque soldat possède un numéro matricule, un nom, un prénom et un grade sur vingt caractères chacun, ainsi qu'une date de naissance qui se compose d'un jour compris entre 1 et 31, d'un mois entre 1 et 12 et d'une année. Par exemple, le Soldat de 1ère classe Raymond Noël a le numéro matricule 20007 et est né le 20/01/72. Quant au Caporal Stéphane Simonin, il est né le 25/11/71 et a le numéro matricule 12045.

1) Définissez, en langage Pascal, la structure de données correspondante comprenant le nom du programme, ainsi que les zones des constantes, des types et des variables. Dans ce but, spécifiez notamment :

a) Le nombre maximum de sections ;
b) La structure permettant d'enregistrer une date (`type date`) ;
c) La structure permettant d'enregistrer des coordonnées (latitude, longitude) (`type coordonnées`) ;
d) La structure permettant d'enregistrer un soldat (`type soldat`) ;
e) La structure permettant d'enregistrer une section (`type section`) ;
f) La structure permettant d'enregistrer une compagnie (`type compagnie`) ;
g) Le fichier des compagnies appartenant au régiment (`fichier RÉGIMENT`).

2) Réalisez la fonction `Ajout` permettant d'ajouter une compagnie passée en paramètre dans le fichier (`regiment.doc`) des compagnies du régiment. Cette fonction renverra le résultat de la fonction prédéfinie `IORESULT`. La fonction `IORESULT` renvoie `"0"` si l'opération d'écriture s'est bien passée, un autre nombre sinon. L'ajout de la compagnie s'effectuera en fin de fichier et toutes les opérations inhérentes à la gestion du fichier s'effectueront dans cette fonction. Aucune variable globale ne sera utilisée, hormis la variable `"rgt"` de type `RÉGIMENT` (le type `RÉGIMENT` correspond au type du fichier `"regiment.doc"`).

3) Réalisez la procédure `Saisie_position` permettant de saisir une position (latitude, longitude) ainsi que la date d'une compagnie donnée dont on affichera le numéro sur deux caractères, ceci pour un emplacement `"i"` entre 1 et 10, dans le tableau des positions de la compagnie. Cette procédure n'utilisera pas de variables globales.

Si vous réussissez cette mission, le Capitaine Théophile Perron soumettra sûrement au Commandant Walter Piget votre nomination au grade supérieur. De plus, si vous persévérez dans la voie de la réussite, vous serez certainement décoré par le Général Philippe Ferrer d'ici cinq ans.

NB : Toute ressemblance avec des personnes ou des événements existant ou ayant existé ne serait que pure coïncidence.

Problème n°5 (6 points) : Qu'ai-je retenu de mon cours ? ***(environ 30 mn)***

a- Quel est l'intérêt d'un fichier ?
b- Pourquoi utiliser un tableau (type `ARRAY`) ? Quand doit-on utiliser ce type de données ?
c- A quoi sert l'instruction `VAR` placée devant un paramètre dans une procédure ?
d- Pourquoi choisir l'instruction `CASE` plutôt que `IF` ?
e- Quand utiliser `REPEAT` plutôt que `WHILE` ?
f- Quel est l'intérêt d'écrire un algorithme avant de commencer à programmer ?
g-Qu'est-ce-que la récursivité ?
h-Quels sont les points communs et les différences entre un tableau et un fichier ?
i- A quoi sert le type `RECORD` ?
j- Quelle est la différence entre variable locale et variable globale ?
k-Quelle est la différence entre nom logique et nom physique ?
l- Quand utiliser l'instruction `FOR` ?

Problème n°6 (7 points) : Mystère … ***(environ 45 mn)***

a) Écrire un programme Pascal qui réalise la fonction récursive suivante :
`MYSTÈRE(X,Y)` vaut `X` si `Y=1`, et vaut `X+MYSTÈRE(X,Y-1)` sinon.
b) Préciser ce que fait cette fonction (X et Y entiers naturels). Pour confirmer, une simulation effectuée sur un exemple simple sera la bienvenue.
c) Indiquez les affichages successifs (au nombre de 11....) du programme ci-dessous :

```
program BINARY_TREE ;
    type tableau = array[1 . . 100] of integer ;
    var        tab : tableau ;
               root, index : integer ;
    procedure TREE_PRINT(t : tableau ; sub_root : integer) ;
    var i, j : integer ;
    begin
         i := 2 * sub_root ; j := 2 * sub_root + 1 ;
         if  ((t[i] <> 0) and (t[j] <> 0))
```

```
            then begin
                   TREE_PRINT(t, i) ; TREE_PRINT(t, j) ;
                 end ;
        writeln(t[sub_root]) ;
        t[sub_root] := 0 ;
end ;
begin
  for index := 1 to 100 do tab[index] := 0 ;
      root := 1 ;
      tab[1]  := 50 ; tab[2]  := 60 ; tab[3]  := 70 ; tab[6]  := 35 ;
      tab[7]  := 80 ; tab[12]  := 1 ; tab[13]  := 2 ; tab[14]  := 34 ;
      tab[15]  := 36 ; tab[28]  := 4 ; tab[29]  := 7 ;
      TREE_PRINT(tab, root) ;
end.
```

Problème n°7 (7 points) :

On dispose de trois fichiers contenant des informations concernant des personnes.

Fichier 1 : "age.dat" contient le nom et l'age
(nom : string, age : integer)

Fichier 2 : "coordonnees.dat" contient le nom, le prénom et l'adresse
(nom, prenom, adresse : string)

Fichier 3 : "secu.dat" contient le nom et le numéro de sécurite sociale
(nom : string, secu :integer)

On considère que chacun des trois fichiers possède un enregistrement sur chacune des personnes.

On se propose de créer un seul fichier regroupant les informations des trois autres fichiers.

1 - Ecrire un fonction `recherche_age` qui renvoie l'âge d'une personne à partir de son nom.

2 - Ecrire une procédure `recherche_prenom_adresse` qui recherche le prénom et l'adresse d'une personne donnée.

3 - A l'aide de `recherche_age` et `recherche_prenom_adresse`, écrire le programme principal qui fusionne toutes les informations des trois fichiers en un seul : "personne.dat" .

Problème n°8 (7 points)

On désire programmer un agenda, dans le but de mémoriser une liste de rendez-vous (100 rendez-vous maximum). Un rendez-vous est défini par sa date, ses horaires de début et de fin (à la minute près) et sa description (message de 200 caractères au plus).

Le programme demande d'abord de saisir tous les rendez-vous. L'utilisateur entre ensuite la date d'un jour (exemple : 16 11 1997), et le programme affiche alors tous les rendez-vous de ce jour.
1 - Indiquer les structures de données qu'utilisera le programme, c'est-à-dire les déclarations Pascal des constantes, types et variables du programme principal. Commenter chaque déclaration pour en expliquer la signification.

2 - Ecrire le programme principal, qui appellera les procédures `saisir_agenda` et `afficher_rdv_du_jour`, définies dans les questions suivantes.

3 - Ecrire la procédure `saisir_agenda`, qui demande à l'utilisateur d'entrer les caractéristiques de tous ses rendez-vous. La procédure retourne l'agenda lu en paramètre de sortie. Suggestion : utiliser une sous-procédure `saisir_rdv` qui lit un seul rendez-vous et le retourne en paramètre de sortie.

4 - Ecrire la procédure `afficher_rdv_du_jour`, qui prend comme paramètres d'entrée la date d'un jour et une liste de rendez-vous, et affiche à l'écran les rendez-vous du jour spécifié. Exemple d'affichage :

```
       Journée du 16 11 1997
 8h15 - 10h15   réunion équipe logicielle
17h23 - 17h45   prendre le bus !!
18h30 - 19h30   tennis jp
3 rendez-vous pour cette journée
```

Problème n°9 (6 points)

Voici le programme suivant.

```
program croise ;
{===========}

var nb, res : integer;

function f (n:integer) : integer ;
{-----------------------------------}

function g(n:integer) : integer ;
begin
    g := f(n div 2) + 1
end ;

begin
if n = 1
   then f := 0
   else f := g (n)
end;

begin
   readln(nb);
   res := f(nb);
   writeln ('f (', nb, ') = ', res)
end.
```

1) Faire une simulation pour nb=23.

2) Que représente la variable globale *res* ?

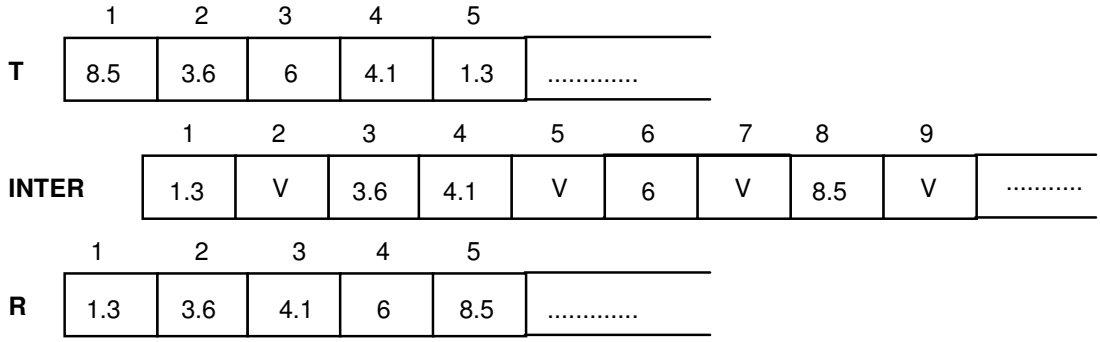
3) A quoi sert ce programme ? Que fait-il ?

Problème n°10 (6 points) : TRIS

Pour trier un tableau T contenant des nombres réels *compris entre 1 et 1000* et ayant des *parties entières différentes,* on opère de façon suivante :

- on place les nombres dans un tableau intermédiaire INTER, chaque nombre X étant placé à l'indice ENT(X) (ENT désigne la fonction "partie entière"). Cette opération fournit un tableau trié contenant des "cases vides".
- on transfère ensuite les nombres de INTER vers un tableau de résultat R, pour obtenir un tableau trié sans "case vide".

Exemple : "V" signifie VIDE



On veut écrire un programme réalisant ce tri de nombres, que l'on qualifiera de TRI DIRECT.

1. a) Préciser les structures de données utilisées pour l'ensemble du programme.

b) Ecrire en Pascal une procédure qui effectue le transfert de T vers INTER. [2 pts]

2. Ecrire en Pascal une procédure qui effectue le transfert du tableau INTER vers le tableau R.

3. a) Ecrire en Pascal l'ensemble du programme. On se contentera d'insérer l'en-tête des procédures précédentes sans les réécrire in extenso.

b) Pourquoi cette méthode de tri est-elle intéressante ? A contrario, en quoi pêche-t-elle ?

Problème n°11 (7 points) : Calculette avec noms et unités

On aimerait disposer d'un programme qui permettrait à l'utilisateur
- de stocker des nombres dans une liste, en associant un nom et une unité à chaque valeur mémorisée ;
- de faire des opérations simples sur ces nombres et de mémoriser le résultat dans la liste.

Cette liste contiendra au maximum 100 nombres, chaque nombre étant décrit par son nom (au plus 20 caractères), sa valeur (un réel), et son unité (au plus 10 caractères).

Au lancement du programme, on commence avec une liste vide. On propose ensuite de façon répétitive un menu principal où l'utilisateur peut choisir entre 4 types d'action :

• s'il entre **s**, on lui demande de saisir un nouveau nom (qui ne doit pas encore exister dans la liste) puis on lui demande de saisir la valeur à mémoriser et son unité ; le nombre ainsi entré est ajouté à la fin de la liste ; exemple de saisie :

```
--- Saisie d'un nouveau nombre ---
Nom      : distance
Valeur   : 120
Unité    : km
```

• s'il entre **a**, on affiche la liste des nombres déjà entrés, chaque nombre sur une ligne (son nom, sa valeur et son unité) ; exemple d'affichage après la saisie précédente :

```
--- Liste des nombres mémorisés ---
temps      2.00    h
distance 120.00   km
```

• s'il entre **+**, **-**, ***** ou **/**, on demande de saisir le nom de deux nombres (qui doivent déjà exister dans la liste) et le nom du nombre résultat (qui doit être différent des noms existants dans la liste) ; on effectue alors l'opération demandée entre les deux nombres désignés, et on stocke le résultat dans le nouveau nombre, celui-ci étant ajouté à la fin de la liste (son unité est déduite de l'unité des opérandes); exemple d'opération :

```
--- Opération / ---
Nom du premier opérande    : distance
Nom du second opérande     : temps
Nom du résultat            : vitesse
Résultat          : vitesse = distance / temps = 60.00 km/h
```

• s'il entre **q**, on quitte le programme.

1- Quelles informations ce programme prend-il en entrée ? Quel résultat produit-il en sortie ? Définir les principales variables utilisées dans le programme et leur type.

2- Décrire l'architecture globale du programme. Pour traiter les choix **a**, **s**, et [**+ - *** ou **/**] du menu principal, on appelle respectivement les procédures `afficher_liste`, `saisir_nombre` et `calculer_operation` (voir question suivante).

3- Définir les paramètres des procédures `afficher_liste`, `saisir_nombre` et `calculer_operation`, et donner un algorithme grossier pour chacune d'entre elles.

4- Donner un algorithme détaillé de la procédure `calculer_operation`. Cette procédure appelle la fonction `cherche_nom` (voir question suivante).

5- Ecrire en Pascal la fonction `cherche_nom`, qui prend comme paramètres un nom et une liste de valeurs nommées ; si le nom donné existe dans la liste, cette fonction retourne la *position* du nombre correspondant, et 0 sinon.

N.B.: Pour les questions 1 à 3, décrivez la solution proposée de façon informelle, de façon à en faire ressortir les grandes lignes - comme si vous deviez expliquer le fonctionnement du programme à un interlocuteur non-informaticien. Pour cela, n'utilisez ni du code Pascal, ni une traduction littérale du Pascal en français, mais des phrases rédigées.

Problème n°12(6 points)

Indiquez, dans les cases correspondantes de la feuille fournie pour répondre à cet exercice, la nature, le type et la valeur des identificateurs utilisés dans le programme suivant aux points indiqués. Les instructions de la ligne où est posé le point sont exécutées avant de déterminer la nature, le type et la valeur des identificateurs quand le programme s'exécute.

La nature est soit une variable globale (notée VG), soit une variable locale (notée VL), soit un paramètre formel (noté PF). Le type sera à déterminer parmi les types simples du langage PASCAL, soit integer (I), soit real (R), soit boolean (B), soit char (C). La valeur sera à déterminer. Par conséquent une réponse sera un triplet, par exemple (VG, I, 34), autre exemple (PF, C, 'g').

```
program asile( input, output );

var
    a,b : integer;  c : real;  d : boolean;

function F1( b,d,c : integer):real;
var
    a : boolean;
begin
    a := true;     { point 1 }
    F1 := sqrt(sqr(d)-(4*b*c));
end;

function F2( d, c : integer):boolean;
var
    a : char;
begin
    a := 'a';
    d:= 0;     { point 2 }
    F2 := (a ='a') or odd(c);
end;

procedure F3( var a : integer ; b : integer );
var
    c : char ;
begin
```

```
    c := chr(b) ;
    c := chr((2 * a) + ord(c));
    a := ord('A') ;   { point 3 }
end;

begin      { et voici le programme principal }
    a := 9 ;
    b := 6 ;
    d := true ;
    c := F1( a, b, 1 ) ;
    d := F2( 7, 3 ) ;
    F3( b , 53) ; { point 4 }
end.
```

Voici un extrait de la table des codes ASCII qui peut vous être utile pour répondre à la question.

code base 10	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70
caractère	2	3	4	5	6	7	8	9	:	;	<	=	>	?	@	A	B	C	D	E	F

Voici le tableau que vous devez remplir pour répondre à la question numéro **3** :

	a	**b**	**c**	**d**
point 1				
point 2				
point 3				
point 4				

Problème n°13 (5 points) : Fichiers

Je désire gérer ma cave à vin. Une bouteille de vin est caractérisée par :

- son nom
- son type (rouge, blanc, rose)
- son annee (1800 .. 1998)
- sa note (0 .. 20)

1 – Définir le type vin.

2 – Ecrire une procédure *creeCave(nom-du-fichier)*, qui créer le fichier *nom-du-fichier* et enregistre dans celui-ci l'ensemble des vins constituant la cave.

3 – Ecrire une procédure *selectUneBouteille(nom-de-bouteille,nom-du-fichier)* permettant d'afficher les caractéristiques d'une bouteille (nom, type, annee et note).

4 – Ecrire une procédure *eclate(nom-du-fichier)* qui, à partir du fichier principal nom-du-fichier, va créer 3 fichiers : rouge, blanc et rose contenant uniquement les vins de même nature.

Problème n°14 (5points) : Domino

On considère un jeu de dominos, comportant bien sûr 28 dominos. On rappelle que chaque domino présente sur l'une des faces (l'autre étant pleine) une paire de chiffres {i,j}
avec $0 \leq i,j \leq 6$. On considère dans cet exercice des séquences de dominos, le principe de constitution d'une séquence étant le suivant : deux dominos d1 = {i1,j1} et d2 = {i2,j2} peuvent être accolés l'un à l'autre à condition d'avoir un chiffre en commun : ainsi, si j1 = j2, alors d1 pourra être suivi de d2, formant au total la séquence (i1,j1,j2,i2).

1) Proposer deux modes de représentation d'une séquence de dominos.

2) Choisir l'une de ces représentations. Soit alors D une séquence de dominos déjà existante. Concevoir une fonction qui admettant D en argument renvoie *vrai* si D respecte bien le principe de formation, et *faux* sinon.

3) Soit J un joueur qui va puiser successivement (*on dit piocher*) dans le tas des 28 dominos. On demande ici de réaliser une procédure qui permette à J, en effectuant des tirages successifs, de mener jusqu'à son terme la constitution d'une séquence de dominos. Pour simplifier , on admettra que le tirage d'un domino est simplement réalisé par une lecture de deux chiffres compris entre 0 et 6. Une fois le domino tiré, si l'application du principe de chaînage est possible, ce domino sera ajouté à la séquence, là où il convient (nécessairement en début ou en fin de séquence), sinon il sera définitivement éliminé. Ce processus s'arrêtera lorsqu'il n'y aura plus rien à piocher.

Problème n°15 (5 points) : Les Pascaloks

Sur la planète Infok les Pascalok jouent à un jeu très populaire. Mais dans chaque région d'Infok où ce joue un match, les Pascaloks ont la mauvaise habitude de ne pas utiliser les mêmes techniques d'affichage des scores. Néanmoins tous les scores doivent apparaîtrent sur toute la planète. Aidez les Pascaloks chargés des affichages à s'y retrouver :

- Les Binoks n'utilisent que des lampes éteintes ou allumées pour afficher les résultats. Ils sont donc en base 2.
- Les Octoks, toujours étourdis, ont malencontreusement perdu les chiffres 8 et 9. Ils sont donc en base 8.
- Les Hexadécimoks, réputés pour leur avarice, ne veulent utiliser que deux caractères pour afficher les scores, ainsi la base 16 leur suffit.
- Enfin les Décimoks, qui sont la honte des Pascaloks, n'utilisent que des chiffres dans une bête base 10.

SCORES	Binoks Affichage Binaire	Octoks Affichage Octal	Héxadécimoks Affichage Hexadécimal	Décimoks Affichage Décimal
Chez les Binoks (base 2) 10110 à 111101				
Chez les Octoks (base 8) 53 à 102				
Chez les Hexadécimoks (base 16) 1E à 39				
Chez les Décimoks (base 10) 172 à 240				

Problème n°16 (5 points)

Soir le programme suivant :

```
program pro_c_est_dur;

Var    i,j,k : integer;

Procedure AFFICHE (n,x,y,z : integer) ;
Begin
   Writeln(n,x,y,z) ;
End;

Procedure TRUC(var a,b,c : integer) ;
Var d : integer ;
Begin
   d := a ;
   a := b ;
   b := d ;
End ;


Procedure BIDULE(var x,y,z : integer) ;

Procedure BAZAR ;
Begin
   x := x*2 ;
   y := y*10 ;
   z := z*100 ;
   AFFICHE(8,x,y,z) ;
End ;

Procedure MACHIN(x,y,z : integer) ;
Begin
   z := x*2 ;
   x := y ;
   y := z+3 ;
End ;

Begin  {début de la partie principale de BIDULE}
   AFFICHE(2,x,y,z) ;
   If x>y
      then
       TRUC(x,y,z) ;
   AFFICHE(3,x,y,z) ;
   MACHIN(x,y,z) ;
   AFFICHE(4,x,y,z) ;
   If y>z
     Then
   Begin
       TRUC(x,y,z) ;
       AFFICHE(5,x,y,z) ;
       MACHIN(x,y,z) ;
       If x>y
         Then
       Begin
           TRUC(x,y,z) ;
           AFFICHE(6,x,y,z) ;
           MACHIN(x,y,z) ;
       End ;
   End ;
   AFFICHE(7,x,y,z) ;
   BAZAR ;
   AFFICHE(9,x,y,z) ;
   TRUC(z,y,x) ;
End ;

Begin      { et voici le programme principal }
   i := 100 ;
   j := 50 ;
```

```
    k := 25 ;
    AFFICHE(1,i,j,k);
    BIDULE(i,j,k);
    AFFICHE(10,i,j,k);
End.
```

Remplir le tableau de la page suivante, en respectant bien les valeurs données pour le premier paramètre (n) de la procédure AFFICHE (ces valeurs de n indiquent l'ordre d'appel de la procédure). Par exemple, lorsque n vaut 1, x a pour valeur 100, y vaut 50 et z vaut 25. Compléter le tableau pour les autres valeurs de n.

Voici le tableau que vous devez remplir pour répondre à la question numéro 4 . Il correspond à l'affichage effectué avec l'instruction `writeln(n,x,y,z)` dans la procédure AFFICHE, selon la valeur du paramètre n (premier paramètre de la procédure AFFICHE). Par exemple, lorsque n vaut 1, x a pour valeur 100, y vaut 50 et z vaut 25. L'instruction writeln(n,x,y,z) donne alors : 1 100 50 25 *(ce qui correspond à la première ligne du tableau).*
Compléter le tableau pour les autres valeurs de n.

n	**x**	**y**	**z**
1	100	50	25
2			
3			
4			
5			
6			
7			
8			
9			
10			

Problème n°17 (7 points) : La Vie des Hauts

Le responsable d'une vidéothèque désire connaître chaque jour la liste de toutes les cassettes qui sont louées, avec le nom du client emprunteur. Chaque cassette est identifiée par un numéro unique. Il dispose de trois fichiers :

Le fichier 1 «Film» fournit le numéro de la cassette, le titre du film, le réalisateur, l'année et le producteur.

Le fichier 2 «Emprunt» contient le numéro de la cassette, la date de début, la date de fin de l'emprunt, le nom de l'emprunteur (Chaque emprunt de cassette est enregistré et conservé ; la date de fin de l'emprunt vaut 00 00 0000 tant que la cassette n'a pas été rendue).

Fichier 3 : «Client» contient le nom du client, son prénom, son adresse, son numéro de téléphone (On suppose que deux clients ne portent pas le même nom).

1pt 1 – Indiquez les structures de données qu'utilisera le programme : c'est à dire les déclarations des constantes, des types et des variables du programme principal.

2pts 2 – Ecrire une procédure ***Recherche_Dernier_Emprunt*** qui renvoie à partir d'un numéro de cassette, le dernier emprunt de la cassette : c'est à dire la date de début, la date de fin de l'emprunt et le nom de l'emprunteur.

2pts 3- Ecrire une procédure ***Recherche_Client*** qui recherche le prénom, l'adresse et le numéro de téléphone à partir du nom du client.

2pts 4- A l'aide des deux procédures précédentes, proposer un algorithme permettant de créer un nouveau fichier des cassettes louées à partir des trois fichiers existants ; ce fichier contiendra le numéro de la cassette, le titre du film, la date de début de l'emprunt, le nom, le prénom, l'adresse et le numéro de téléphone de l'emprunteur.

Problème n°18 (7 points) : Les 'Strings' de l'été

On cherche à réaliser un petit logiciel de traitement de texte (Mon Mini-Word). Ecrire un programme qui justifie une phrase, c'est-à-dire qui l'étend depuis la marge gauche jusqu'à la marge droite de la feuille. Ce programme commence par demander la largeur de la feuille.

1. Ecrire un programme qui commence par demander une largeur, puis demande une chaîne de longueur inférieure à cette largeur qui se termine par un point, et compte le nombre de mots de cette chaîne.

2. Ecrire le programme qui décompose la chaîne de façon à répartir "au mieux" les blancs qui s'y trouvent.

Exemple :

largeur : 27

La chaîne

t	o	t	o		n'	e	s	t		p	a	s		c	o	n	t	e	n	t	.				

devient :

t	o	t	o			n'	e	s	t				p	a	s			c	o	n	t	e	n	t	.

Problème n°19 (6 points) : Var ou pas Var, Local ou pas ? telle est la question...

Voici un programme qui ne sert à rien mais qui affiche plein de choses.

```
---------------------------------------------------------------------------------------------------------
PROGRAM TRUC
CONST C=30;
VAR A,B,D,E : integer;

PROCEDURE Affiche(D, E : integer);
VAR A : integer;
begin
      A:=40; E:=50;
      writeln('Affiche');
      writeln('A=',A); writeln('B=',B); writeln('C=',C); writeln('D=',D); writeln('E=',E);
```

```
end;

FUNCTION Calcul(D, E : integer):integer;
VAR B,C : integer;
begin
      B:=60; C:=50; D:=80;
      Calcul := E;
      writeln('Calcul');
      writeln('A=',A); writeln('B=',B); writeln('C=',C); writeln('D=',D); writeln('E=',E);
end;

begin
      A:=10; B:=20; E:=70;
      Affiche(A,B);
      D:=Calcul(A,B);
      writeln('Truc');
      writeln('A=',A); writeln('B=',B); writeln('C=',C); writeln('D=',D); writeln('E=',E);
end.
```

Q1 - Complétez le tableau Q1 avec les valeurs affichées lors de l'exécution de ce programme.

Q2 - On modifie les paramètres de la fonction -Calcul- et de la procédure -Affiche- comme suit : PROCEDURE Affiche(D : integer; **var** E : integer);
FUNCTION Calcul(**var** D : integer; E : integer);
Complétez le tableau Q2 avec les valeurs affichées lors de l'exécution après modifications.

Complétez le tableau Q1 avec les valeurs affichées lors de l'exécution de ce programme.

Complétez le tableau Q2 avec les valeurs affichées lors de l'exécution après modifications.

Q1 **avant modifications**
Affiche
A=
B=
C=
D=
E=
Calcul
A=
B=
C=
D=
E=
Truc
A=
B=
C=
D=
E=

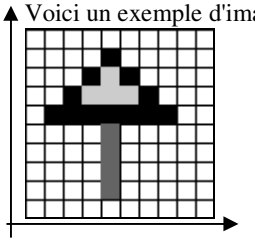
Q2 **après modifications**
Affiche
A=
B=
C=
D=
E=
Calcul
A=
B=
C=
D=
E=
Truc
A=
B=
C=
D=
E=

Problème n°20 (6 points) : Traitement d'images

Afin de représenter des images en informatique on utilise des tableaux de points (ou *pixels*). Dans le cas d'images en noir et blanc chaque point est doté d'une luminosité plus ou moins forte entre le noir et le blanc, on

appelle cela les niveaux de gris. Dans le cadre de cet exercice on travaillera avec des images en 256 niveaux de gris. Par convention on adopte que le noir vaut 0 et le blanc 255.

Voici un exemple d'image:



Cette image a une résolution de 10 points x 10 points. Chaque point du tableau, repéré par une abscisse sur l'intervalle [1..10], une ordonnée sur l'intervalle [1..10], prend donc une valeur de luminosité comprise entre 0 et 255.
Exemple: le point (5,1) à une valeur 255 ; le point (5,2)=58 ; le point (5,8)=195; le point (5,9)=0.

Question 1 : Proposer un type IMAGE permettant de représenter une image de résolution 640 x 480 points.

Question 2 : Proposer une procédure qui éclaircit une IMAGE passée en paramètre. L'éclaircissement consiste à augmenter la luminosité de chaque pixel d'une intensité comprise entre 1 et 255. Ce nombre sera également passé en paramètre à la procédure. Attention la luminosité maximum est 255 !

Question 3 : Proposer une procédure qui contraste une IMAGE passée en paramètre. Cette procédure de contraste consiste à éclaircir les points clairs (de luminosité supérieure à 127) et à assombrir les points sombres (de 0 à 127). Comme dans la question précédente l'intensité du contraste sera passée en paramètre.

Question 4 : Proposer une fonction booléenne qui détermine si deux IMAGES passées en paramètre sont identiques.

Note: Toutes les questions devront être traitées en Pascal.
Indication : Chaque procédure tient en moins de 15 lignes.

Problème n°21 : La récursivité peut être vitale

Certains romanciers ont imaginé les *ravages* qu'occasionnerait la disparition de l'électricité dans notre monde. Imaginons ceux que causerait la disparition des boucles dans un monde un peu particulier … Dans ce monde un programme permet de calculer la durée de vie d'un être vivant, selon la loi suivante :

Chaque être a (150-x)% de chance de survie chaque année.
ou x représente l'âge courant de l'être concerné.

Le programme actuellement utilisé pour faire fonctionner cette loi est le suivant:

```
program avec_boucles;
function age_deces (age_naissance:integer):integer;
 var age_courant:integer;
 begin
age_courant:=age_naissance;
randomize;
while random(100) < (150-age_courant) do
   age_courant:=age_courant+1;
age_deces:=age_courant;
 end;

begin
 writeln(age_deces(0));
end.
```

Note : La fonction *random(x)* renvoi un nombre entier aléatoire entre 0 et x-1. La procédure *randomize* permet de réinitialiser le générateur de nombre aléatoire pour obtenir un nombre différent à chaque appel de *random*.

Question : Réécrivez la fonction *age_deces* de façon à la rendre récurssive (et donc sans utiliser de boucle *while*).

Problème n°22 : Gestion d'un stock de sucreries

Le gérant d'une petite confiserie ne proposant que 5 types de bonbons et 5 types de chocolats s'est amusé durant ses loisirs à concocter son propre programme Pascal de facturation. Cette première version ne lui permet pour l'instant que de calculer le montant de la facture de chaque client. Ce programme demande tout d'abord de rentrer le code de l'article acheté puis la quantité, ce processus se répétant jusqu'à ce que l'utilisateur tape le code –1 qui a pour conséquence l'arrêt de la saisie et l'affichage du montant global des achats. Ce programme permet également de rejeter la prise en compte de codes incorrects ne correspondant à aucune marchandise. Le listing de ce programme vous est donné en annexe (feuille séparée). Prenez-en d'abord connaissance.

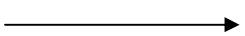
Question 1 : Dans ce programme, le gérant a déclaré en constante les différents types de marchandises dont il dispose. Une malencontreuse erreur de frappe a fait disparaître certaines déclarations de types utilisés dans le programme. Remédiez à cela en complétant les zones correspondantes sur le listing fourni, sur la feuille détachable.

Question 2 : Deux clients se sont présentés successivement au magasin.

Complétez toutes les cases des tableaux fournis en annexe (feuille séparée) en spécifiant les valeurs prises par les différentes variables en question après le traitement de chaque client.

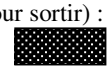
Voilà les saisies réalisées et les affichages qui en découlent :

```
**********************
* Nouveau client                *
**********************
Référence de l'article (-1 pour sortir) : 0010
Quantité : 4
Référence de l'article (-1 pour sortir) : 1001
Quantité : 4
Référence de l'article (-1 pour sortir) : 1111
Quantité : 3
Cet article n'existe pas
Référence de l'article (-1 pour sortir) : 0001
Quantité : 4
Référence de l'article (-1 pour sortir) : 0101
Quantité : 5
Référence de l'article (-1 pour sortir) : -1
Somme dûe :
```



qte ?, emplacement ?, somme ?, j ?

```
(Nouveau client : « n » ; Quitter : « q »)
n

**********************
* Nouveau client          *
**********************
Référence de l'article (-1 pour sortir) : 1010
Quantité : 3
Référence de l'article (-1 pour sortir) : 1001
Quantité : 3
Référence de l'article (-1 pour sortir) : 0110
Quantité : 3
```

Référence de l'article (-1 pour sortir) : -1
Somme due : 

(Nouveau client : « n » ; Quitter : « q »)
n

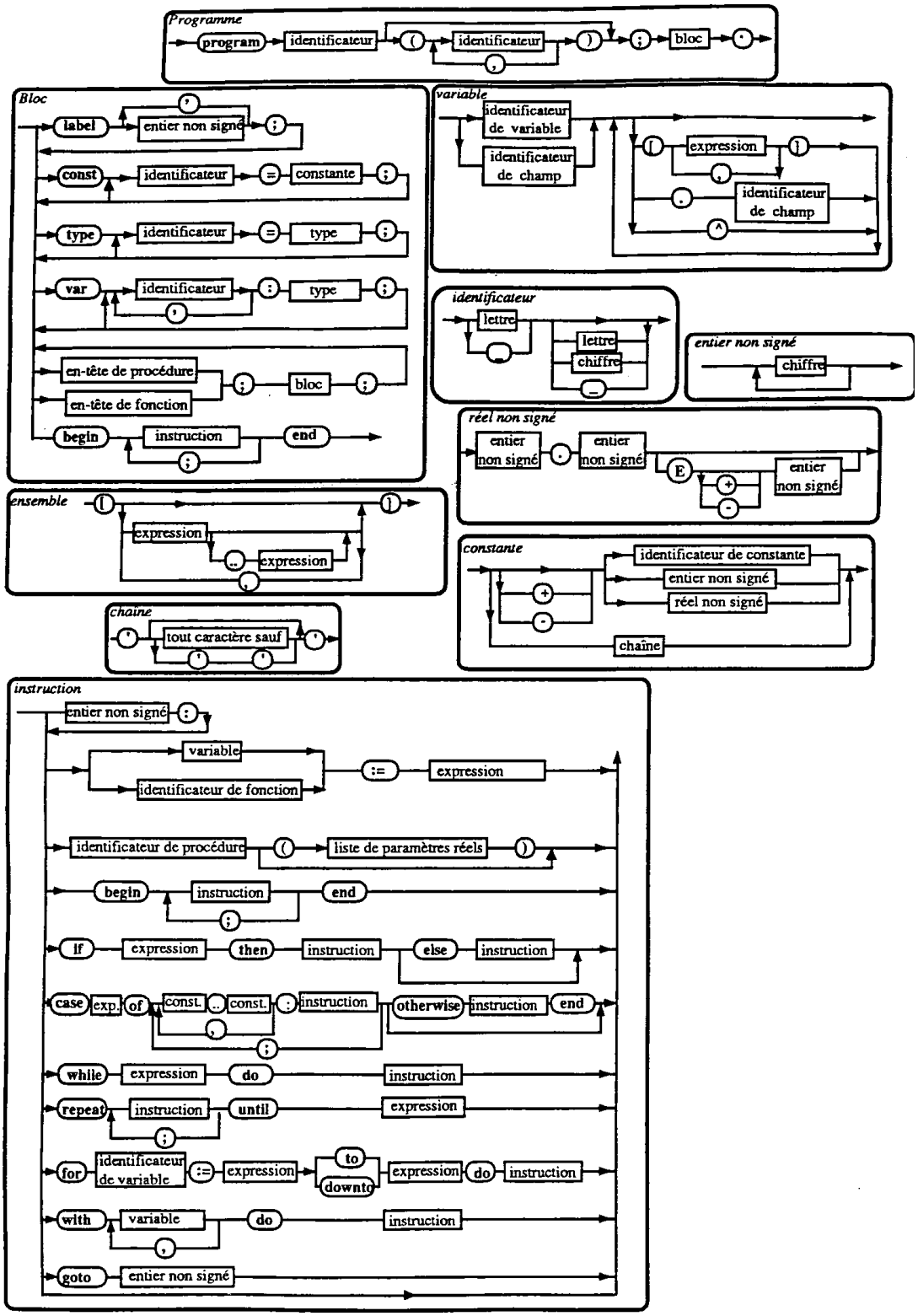
…

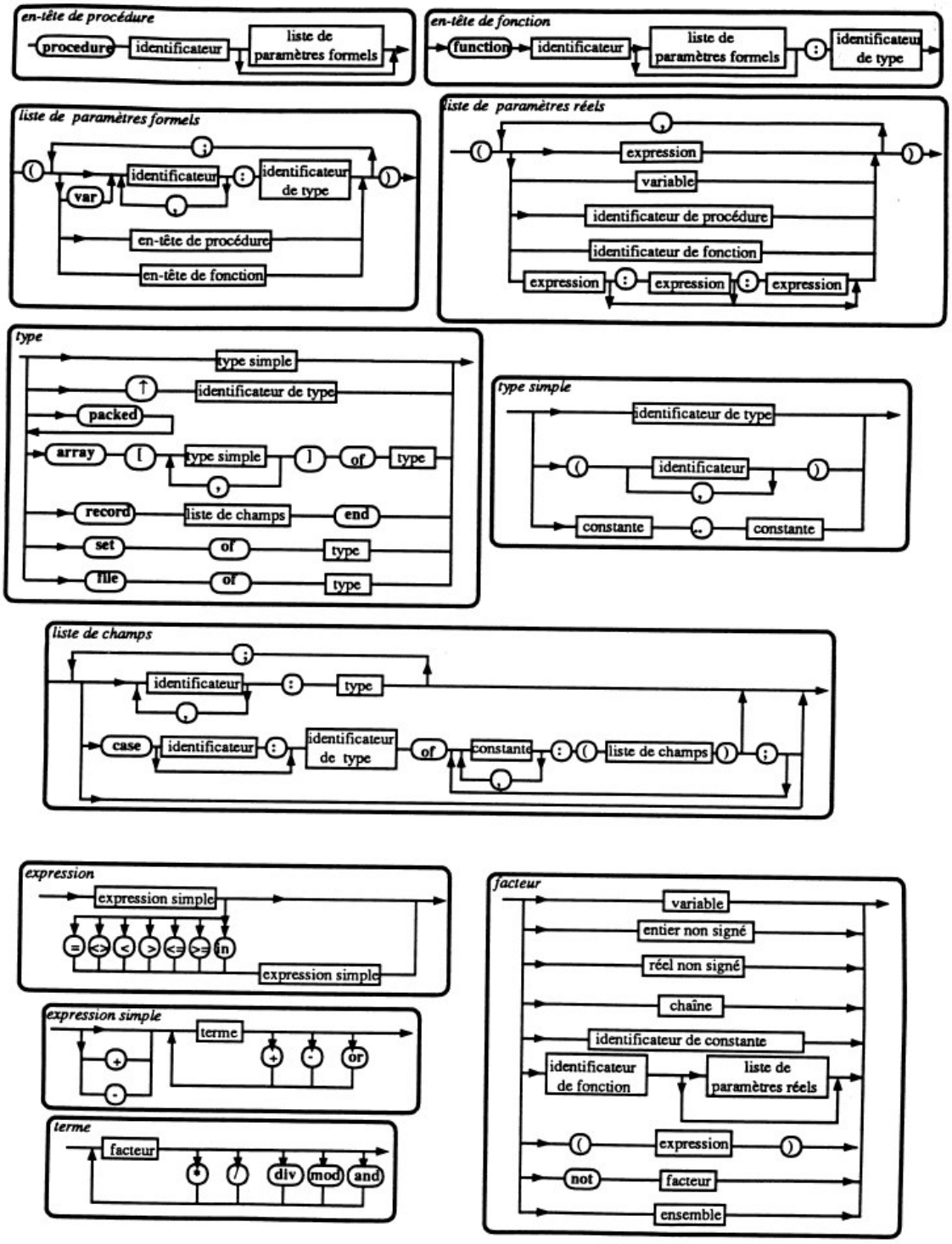
qte ?, emplacement ?, somme ?, j ?

Question 3 : Le gérant souhaite de plus afficher une facture détaillée des achats de chaque client en présentant tout d'abord les bonbons puis les chocolats. Un exemple de la présentation souhaitée est donné ci-dessous. Créez une procédure qui vous permette d'obtenir un affichage de ce type. Placez également dans le programme principal la ligne de commande faisant appel à cette procédure.

```
FACTURE
4     bonbons    « Ourson »            :  0.8 F
4     bonbons    « Frite               :  0.4 F

4     chocolats  « Noir et orange »    :  18.8 F

            TOTAL  :                   20 F
```

16 Syntaxe du langage Pascal




en-tête de procédure
procedure
identificateur
liste de paramètres formels
en-tête de fonction
function
identificateur
liste de paramètres formels
identificateur de type
liste de paramètres formels
var
identificateur
identificateur de type
en-tête de procédure
en-tête de fonction
liste de paramètres réels
expression
variable
identificateur de procédure
identificateur de fonction
expression
expression
expression
type
type simple
identificateur de type
packed
array
type simple
of
type
record
liste de champs
end
set
of
type
file
of
type
type simple
identificateur de type
identificateur
constante
constante
liste de champs
identificateur
type
case
identificateur
identificateur de type
of
constante
liste de champs
expression
expression simple
expression simple
expression simple
terme
terme
facteur
facteur
variable
entier non signé
réel non signé
chaîne
identificateur de constante
identificateur de fonction
liste de paramètres réels
expression
not
facteur
ensemble