

Informatique : principes et méthodes de programmation

Cours dispensé par
Thomas DUFRASNE

Introduction

Organisation

- Théorie : 32 heures
 - Laboratoire : 64 heures
 - Autonomie : 24 heures
-
- Total : 120 heures

Objectifs

- Mise en œuvre et justification d'une méthodologie rigoureuse et cohérente de résolution de problèmes (observation, résolution, expérimentation, validation) ;
- Appliquer à ces situations les fonctionnalités essentielles d'un langage procédural choisi ;
- Concevoir et construire des algorithmes en utilisant de manière pertinente les éléments suivants :
 - o Les types de données élémentaires ;
 - o Les vecteurs et matrices ;
 - o Les figures algorithmiques de base (séquence, alternative et répétitive) ;
 - o Les expressions et instructions ;
 - o Entrées et sorties conversationnelles ;
 - o Les fonctions et procédures ;
 - o La récursivité ;
 - o Les fichiers.
- Traduire de manière adéquate des algorithmes en respectant les spécificités du langage utilisé ;
- Recourir à bon escient à la documentation disponible.

Langage

Utilisation du langage PASCAL

- TURBO PASCAL
- BORLAND PASCAL

- Dérivé → DELPHI

Remarque générale

Vu que la table des matières évolue en même temps que la mise à jour du document, vous trouverez cette table en fin d'ouvrage. Elle sera donc distribuée avec la dernière partie du présent syllabus.

Concepts informatiques

Un programme est une succession d'instructions destinées à être exécutées par un ordinateur, dans un certain ordre, afin de produire un résultat spécifique.

Les commandes comprises par un ordinateur sont, d'une manière interne, exprimées dans un code binaire (succession de 0 et de 1).

Un langage de programmation est un langage qui permet au programmeur de donner des commandes non ambiguës à l'ordinateur.

Il existe différents types de langages :

- Le langage machine : c'est le seul langage compris par la machine. Son usage est complexe et spécifique à chaque ordinateur. Il est constitué de symboles binaires.
- Le langage d'assemblage (ou Assembleur) : il est directement inspiré du langage de la machine. Il en est très proche mais est plus clair. Il est constitué de texte et non de symboles binaires.
- Les langages évolués : ils sont proches du langage de l'homme. Le compilateur se charge de traduire ce langage en langage machine. Quelques exemples : COBOL, PASCAL, FORTRAN, C.

La séquence des étapes ou opérations qui doivent être suivies par le programme est l'algorithme du programme. L'algorithme est la spécification pas à pas du processus qui doit résoudre le problème posé. Il doit être clair et non ambigu. Cet algorithme est transmis à l'ordinateur au travers du langage de programmation.

Le langage PASCAL appartient à la famille des langages structurés. Ceux-ci sont des langages qui permettent l'élaboration des programmes à partir d'un ensemble de blocs d'instructions. Les programmes écrits au moyen d'un tel langage peuvent être réalisés à partir de quatre structures de base : la séquence, la décision, la structure répétitive, la procédure.

- La **séquence** est un groupe d'instructions exécutées les unes après les autres ;
- Une **décision** est une structure permettant à une action du programme d'être influencée par des données ;
- La **structure répétitive** est utilisée pour exécuter une instruction ou une suite d'instructions plusieurs fois. Alors que les instructions sont les mêmes chaque fois que la boucle est exécutée, les données sur lesquelles elles agissent changent d'une fois à l'autre.
- La **procédure** (ou **fonction**) permet de remplacer un groupe d'instructions par un nom qui identifie ce groupe. Les procédures permettent de donner aux programmes une structure modulaire, ce qui rend l'écriture plus facile et plus dense.

Exemple simple

Le code source

```
{ *****  
 * Programme de calcul de racines carrees *  
 *      fournies par l'utilisateur      *  
 ***** }  
program prg_premier ;  
const Nombre_fois = 5 ;  
var i      : integer ;  
    nombre : real ;  
    racine  : real ;  
begin  
  writeln('bonjour') ;  
  writeln('je vais vous calculer ',Nombre_fois, ' racines  
carrees :') ;  
  for i :=1 to Nombre_fois do begin  
    write('donnez un nombre : ') ;  
    readln(nombre) ;  
    if nombre < 0 then writeln('Pas de racine pour ce  
nombre !')  
    else begin  
      racine := sqrt (nombre) ;  
      writeln(nombre,' a pour racine ',racine)  
    end ;  
  end ;  
  writeln('Merci') ;  
end.
```

Explications

- En-tête :

Il s'agit de la première ligne du programme, identifiant le programme en lui donnant un nom. Un nom identifiant le rôle du programme sera choisi de préférence. Vous éviterez d'attribuer à une variable le nom donné dans cette section. Si vous tentez cela, vous obtiendrez un message d'erreur à la compilation.

Attention : " ; " en fin de ligne.

- Partie déclaration :

Elle s'étend de la fin de l'en-tête jusqu'au premier mot BEGIN. Elle comporte dans ce cas deux sortes de déclarations (CONST et VAR).

```
const Nombre_fois = 5 ;
```

Dans tout le programme, l'identificateur désignera la valeur 5.

Avantages : Meilleure visibilité
Meilleure adaptation

```
var i      : integer ;  
    nombre : real ;
```

```
racine      : real ;
```

Déclaration de "type" de variables. i prendra des valeurs différentes lors de l'exécution du programme.

```
i          → type entier  
nombre,  
racine    → type réel
```

- Partie exécutable

Observation du corps du programme (BEGIN...END, WRITELN, WRITE, READLN, FOR...TO, READ, IF...THEN...ELSE et ponctuation.

- Documentation

Soit par des commentaires (cf. exemple) et/ou par la documentation accompagnant le programme. Toujours documenter ses programmes.

- Lisibilité

Remarquez que le code a été écrit de telle manière que les blocs d'instructions sont clairement délimités. Il en résulte que le code source sera plus clair. Par exemple, nous verrons tout de suite si nous avons oublié un "end". Cet agencement d'écriture n'est pas un luxe, il est obligatoire !

Édition et compilation

Édition	Saisir le code source et l'enregistrer. Se fait via l'éditeur de TURBO PASCAL. Enregistrement du fichier mon_programme.pas. Prenez l'habitude d'enregistrer votre programme AVANT la première compilation.
Compilation	Traduction du code source en langage machine. Le résultat est mon_programme.exe.

☒ **Exercice 1**

Lancer l'éditeur, enregistrer le code source de l'exemple, compiler et exécuter le programme.

Règles générales du TURBO PASCAL

Éléments du langage : identificateurs, identificateurs prédéfinis et mots clés

Les identificateurs sont des noms choisis par le programmeur (Nombre_fois, i, racine, nombre). Ce sont les noms de variables, de constantes, de procédures, de fonctions et de types.

Dans l'exemple se trouvent également des mots imposés par TURBO PASCAL (program, integer, const, begin...) Ces mots se classent en deux catégories : les mots clés et les identificateurs prédéfinis.

Des mots clés ne peuvent jamais être utilisés comme identificateurs. Ils constituent le "vocabulaire" du langage PASCAL. Leur rôle et leur signification sont parfaitement définis.

absolute	end	near (2) (4)	string (1)
and	external (1)	nil	then
asm (1)	far (2) (4)	not	to
array	file	object (3)	type
assembler (1)	for	of	unit (1)
(2)	forward (2)	or	until
begin	function	packed	uses (1)
case	goto	private (2) (4)	var
const	if	procedure	virtual (1) (2)
constructor (3)	implementation (1)	program	while
destructor (3)	in	record	with
div	inline (1)	repeat	xor (1)
do	interrupt (1) (2)	set	
downto	label	shl (1)	
else	mod	shr (1)	

- (1) : Mot clé propre à TURBO PASCAL, n'existe pas en PASCAL standard
(2) : Depuis la version 6, traité comme identificateur prédéfini (cf. infra)
(3) : depuis la version 5.5
(4) : depuis la version 6.

Les identificateurs prédéfinis sont des mots possédant une signification par défaut (Integer, real, writeln...). Ils peuvent être utilisés comme identificateur par le programmeur, mais cela prête à confusion.

Il est donc possible d'appeler une variable Integer → à éviter !

Les règles d'écriture des identificateurs

Les identificateurs sont des noms d'entités définis par l'utilisateur (constantes, variables, procédures, fonctions, types...). Le programmeur définit ces identificateurs dans une partie spécifique du programme, le bloc des déclarations.

Un identificateur devra respecter les règles suivantes :

- Il ne doit contenir que des lettres ou des chiffres ou le caractère souligné '_'. Parmi les lettres, seules sont acceptées les 26 lettres de l'alphabet. Les lettres accentuées ne sont pas acceptées par le compilateur.
- Le premier caractère d'un identificateur doit être une lettre ou le caractère '_'.
- Seuls les 63 premiers caractères d'un identificateur sont significatifs et sa longueur ne peut dépasser une ligne de l'éditeur. On veillera cependant à garder une longueur "lisible".
- ATTENTION : Pascal est un des rares programmes qui ne font pas la différence entre les minuscules et les majuscules.

☒ Exercice 2

Quels sont les identificateurs corrects ou incorrects ?

```
A
A1
1b
n
n_1
racine_carree
racine_carrée
Racine-carree
Taxe_sur_la_valeur_ajoutee
Nombre 1
```

Un espace est obligatoire pour séparer les différents mots, sauf lorsque deux mots sont séparés par ':' ou ';'. Toutefois, le programmeur s'imposera de la clarté dans ses codes sources. Par ailleurs, une chaîne ne peut être écrite sur plusieurs lignes.

```
program Racines_carrees ; const Nombre_fois = 5 ; var i : integer ;
nombre : real ; racine : real ; begin writeln ('bonjour ') ;
writeln('je vais vous calculer ',Nombre_fois, ' racines carrees :') ;
for i :=1 to Nombre_fois do begin write('donnez un nombre : ') ;
readln(nombre) ; if nombre < 0 then writeln
('Pas de racine pour ce nombre !') ; else begin
racine := sqrt (nombre) ; writeln
(nombre,' a pour racine ',racine) end ; end ;
writeln('Au revoir ') ; end.
```

Même si la mise en page d'un programme est libre ou presque, le programmeur évitera de saisir le code d'une manière qui complique tout travail de maintenance sur ce code.

Les commentaires

Les commentaires, signalés au compilateur par les symboles ' { ' et ' } ' ou ' * ' et ' * } ' pourront être placés partout où un espace ou un saut de ligne est permis.

```
{ un exemple de commentaire }  
  
{* un autre exemple de commentaire *}
```

Il est indispensable de commenter ses programmes. Il est erroné de penser que les codes sources sont lisibles et compréhensibles, même pour celui qui l'encode. Il faut garder à l'esprit que les programmes sont susceptibles d'être modifiés longtemps après leur encodage, et les commentaires sont indispensables dans ce cas.

Organisation de la partie déclaration

Dans la partie déclaration doivent être déclarés les constantes (const), les variables (var), les labels (label), les types (type), les procédures (procedure) et les fonctions (function).

Attention : cela doit se faire dans cet ordre en PASCAL STANDARD (l'ordre importe peu en TURBO PASCAL).

Structure générale d'un programme PASCAL

PROGRAM	monProgramme ;
LABEL	déclaration ;
CONSTANT	définition ;
TYPE	définition ;
VARIABLE	déclaration ;
PROCEDURE	déclaration ;
FUNCTION	déclaration ;
BEGIN	
	Instructions ;
END.	

Les types scalaires prédéfinis

Il s'agit des types INTEGER, REAL, CHAR et BOOLEAN

Les types variables

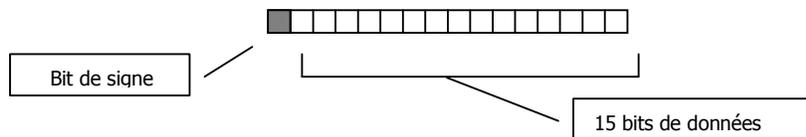
Le type INTEGER (entier)

Il correspond à des entiers et se déclare grâce à l'identificateur integer.

Exemples corrects : 12 +25 -324

Les opérateurs applicables au type entier

- + Addition
- Soustraction
- Opposé
- * Multiplication
- div Division entière
- mod Modulo (reste de la division entière)



Exemples :

```
0    0000000000000000
1    0000000000000001
2    0000000000000010
16   0000000000010000
127  0000000001111111
256  0000000011111111
-1   1111111111111111
-2   1111111111111110
-3   1111111111111101
-256 1111111100000000
```

Limites

Une variable de type Integer s'étend de -32768 à 32767.

Fonctions relatives

Fonction	Rôle
Abs(i)	Retourne la valeur absolue de i
sqr(i)	Retourne i élevé au carré ATTENTION au dépassement → En cas de doute, le type longint sera utilisé

☒ Exercice 3 , Exercice 4, Exercice 5

Le type REAL (réel)

Correspond de manière relativement bien approchée aux nombres réels et se déclare grâce à l'identificateur `real`. Il peut s'écrire sous la forme décimale ou exponentielle.

Exemples d'écriture :

12.43	OK
-0.453	OK
+1.0	OK
.58	NOK (pas de chiffre avant le point)

PASCAL nous permet également d'écrire ces nombres au format exponentiel

4.25E+4	OK
4.25E4	OK
5.48E-3	OK
28e3	OK
-45e-13	OK
5e0.2	NOK (exposant non entier)
-.25e3	NOK (pas de chiffre avant le point)
1.25 E 3	NOK (espaces)

Les opérateurs applicables au type réel

+	Addition
-	Soustraction
-	Opposé
*	Multiplication
/	Division exacte

Limites

Les réels sont codés sur 6 octets (40 bits pour la mantisse, 8 pour l'exposant). Une variable de type `Real` s'étend donc (en valeur absolue) de $2.9 \cdot 10^{-39}$ à $1.7 \cdot 10^{38}$.

Fonctions relatives

Fonction	Rôle
abs(i) sqr(i) sqrt(i)	valeur absolue de i élévation au carré racine carrée
Sin Cos Arctan	sinus (argument en radian) cosinus (argument en radian) arctg (résultat en radian)
ln exp	logarithme népérien exponentielle
frac int	partie fractionnaire frac(1.35) = 0.35 partie entière int(-4.75) = -4

Fonctions de conversion de réel → entier

Permet de convertir un réel en entier

Fonction	Rôle
round(n)	arrondi à l'entier le plus proche round(2.85) = 3 round(2.32) = 2
trunc(n)	partie entière de n trunc(2.32) = 2 trunc(-2.85) = -2

☒ Exercice 6

ATTENTION :

```

program prg_01_attention;
uses crt;
const x = 1e37 ;
      y = 1.0 ;
var valeur1 : real ;
    valeur2 : real ;

begin
  clrscr;
  valeur1 := (x + y - x)/y ;
  valeur2 := (x - x + y)/y ;
  writeln('Valeur 1 : ', valeur1);
  writeln('Valeur 2 : ', valeur2);
  readkey;
end.

```

→ Nous reviendrons plus loin sur la priorité des opérateurs.

Le type CHAR (caractère)

Exemple introductif :

```
program prg_02_char ;  
  
const mon_choix = 'e';  
var votre_choix : char;  
  
begin  
  write('Choisissez un caractère : ');  
  readln(votre_choix);  
  writeln('Merci pour      : ', votre_choix);  
  writeln('Mon choix est : ', mon_choix);  
  readln ;  
end.
```

Le type char est codé sur un octet et représente un caractère quelconque. Il inclut les 26 caractères de l'alphabet (minuscules et majuscules), les chiffres de 0 à 9 et certains caractères spéciaux (symboles et caractères de contrôle).

Écriture :

Les caractères imprimables (32 à 126) s'écrivent de la manière :

'a' 'C' '1' '+' ':'

Les caractères de contrôle s'écrivent de cette manière :

cloche := ^G (Attention : pas d'apostrophe)

Fonctions relatives

Fonction	Argument passé	Résultat	Rôle
Ord	Char	Integer	Valeur numérique du code du caractère
Chr	Integer (entre 0 et 255)	Char	Caractère ayant le code indiqué

☒ Exercice 7, Exercice 8

Le type BOOLEAN (booléen)

Il s'agit d'un type logique qui n'apparaît pas de manière aussi intuitive que les autres précédemment vus.

Exemple introductif :

```
program prg_booleen1 ;
var n, p : integer ;
begin
  writeln('donnez 2 entiers :') ;
  readln(n) ;
  readln(p) ;
  if n < p then
    writeln('croissant !')
  else
    writeln('non croissant ou égal !') ;
end.
```

Dans ce programme, la condition ($n < p$) est vérifiée. Si la condition est vraie, il y a exécution de l'instruction affichage 'croissant !'. Sinon, il y a affichage de 'non croissant ou égal !'.

La condition ($n < p$) est en fait une expression **booléenne**, pouvant prendre la valeur TRUE (vraie) ou FALSE (fausse).

Le programme ci-dessus peut se réécrire en introduisant une variable de type **booléen** :

```
program prg_03_boolean ;
var n, p : integer ;
    result : boolean ;
begin
  writeln('donnez 2 entiers :') ;
  readln(n) ;
  readln(p) ;
  result := n < p ;
  if result = true then
    writeln('croissants !')
  else
    writeln('Décroissant ou égaux !');
  readln;
end.
```

Il est possible d'écrire ce type de variable et donc de visualiser si une variable de type booléen est vraie (TRUE) ou fausse (FALSE). Voici ci-dessous un exemple qui affiche le résultat d'une variable, dont la valeur dépend des nombres introduits par l'utilisateur :

```
program prg_booleen3 ;
var n, p : integer ;
    result : boolean ;
begin
  writeln('donnez 2 entiers :') ;
  readln(n) ;
  readln(p) ;
  result := n < p ;
  writeln('Résultat : ',result) ;
  if result = true then
```

```

        writeln('croissant !')
    else
        writeln('non croissant ou égal !')
end.

```

Si nous voulons traiter les 3 cas de figures possibles – croissants, décroissants ou égaux – nous devons ajouter une instruction `if` supplémentaire. Le programme se présenterait de cette manière :

```

program prg_booleen4 ;
uses crt;
var n, p      : integer ;
begin
    write('donnez 2 entiers : ') ;
    readln(n) ;
    write('          : ');
    readln(p) ;
    if n<p then
        writeln('Croissants !')
    else if n>p then
        writeln('Décroissants !')
    else
        writeln('Egaux !');
    readkey;
end.

```

Opérateur de comparaison :

Opérateur	Signification numérique	Signification caractère
=	égal	égal
<	inférieur	De code inférieur
<=	inférieur ou égal	De code inférieur ou égal
>	supérieur	De code supérieur
>=	supérieur ou égal	De code supérieur ou égal
<>	différent	différent

Lorsque PASCAL compare deux données scalaires, il parvient toujours à les ordonner (pour autant que les deux valeurs soient du même type.) Il est donc possible de comparer 2 réels, 2 caractères, 2 entiers. PASCAL se base sur la valeur ASCII.

☒ Exercice 9

Opérateurs booléens

Opérateur AND

X	Y	X and Y
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

- $(a < b) \text{ and } (b < c)$

Opérateur OR

X	Y	X or Y
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

- $(a < b) \text{ or } (b < c)$

Opérateur XOR

X	Y	X xor Y
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

- $(a < b) \text{ xor } (b < c)$

Opérateur NOT

X	not X
TRUE	FALSE
FALSE	TRUE

- ☒ Exercice 10, Exercice 11, Exercice 12, Exercice 13, Exercice 14, Exercice 15

Le type CONST (constante)

Comme nous avons eu l'occasion de le voir précédemment, la déclaration de type n'est pas nécessaire lorsqu'une constante est déclarée :

Exemple :

```
const constRelle = 2.6 ;  
constEntiere = 3 ;  
constLettre = 'A' ;
```

Les entrées et sorties conversationnelles

Dans l'exemple introductif et dans les différents exercices réalisés jusqu'ici, nous avons eu l'occasion d'utiliser ce qui est appelé "les entrées et sorties conversationnelles". L'utilisateur entre un nombre au clavier, le programme quant à lui affiche à l'écran divers messages et le résultat du calcul de la racine carrée.

Dans ce chapitre, nous verrons les règles qui s'imposent lorsque l'utilisateur doit entrer des informations et lorsque le programmeur veut les afficher.

Les instructions de lecture

Il est peut-être utile de rappeler que lorsque nous parlons de lecture, il s'agit de la lecture par le programme de données introduites par l'utilisateur. Jusqu'ici, les seules données en entrée d'un programme l'ont été via le clavier. Nous avons eu l'occasion de rencontrer quelques exemples et nous savons maintenant que l'instruction de lecture est donnée par **read()** ou **readln()**.

Pour les différents exemples que nous rencontrerons par après, nous utiliserons à chaque fois le même type de programme : un petit programme destiné à lire une ou plusieurs données introduites par l'utilisateur et à afficher ensuite à l'écran ces mêmes données. Cela nous permettra de voir de quelle manière le compilateur interprète – ou comprend – les données qui lui ont été données.

Exemples introductifs

```
program prgInOut_1 ;
uses crt ;
var n : integer ;
begin
  write('Donnez un entier') ;
  read(n);
  writeln('Merci pour ', n) ;
  readkey ;
end.
```

Nous partirons d'exemples simples, dans lesquels n et p sont des variables entières, x et y sont des variables réelles et c1 et c2 sont des variables de type char. L'espace tapé au clavier est représenté par le symbole ^, alors que l'appui de la touche enter est représenté par le caractère : ↵

Lecture d'un entier :

Valeurs introduites	Résultat de "n"
-12 ↵	-12
^^25^^ ↵	25
12. ↵	Erreur
45bonjour ↵	Erreur
45 Bonjour ↵	45

On constate que si le nombre est entouré d'espaces, cela ne pose pas de problèmes. Par contre, si les données numériques sont "mélangées" avec des données alphabétiques (sauf le signe + et -), cela entraîne une erreur

Lecture de deux entiers :

Pour ce faire, nous utilisons le petit programme suivant :

```

program prgInOut_2 ;
  uses crt ;
  var
    n,p  : integer ;
begin
  writeln('Donnez deux entiers') ;
  read(n,p);
  writeln('Merci pour ', n, ' et ',p) ;
  readkey ;
end.

```

Valeurs introduites	Résultat de "n"	Résultat de "p"
^13^^48 ↵	13	48
15^-7 ↵	15	-7
15-7 ↵	Erreur	
15^^↵^-7 ↵	15	-7

Ici aussi, si nous introduisons deux nombres séparés par un espace ou par un saut de ligne, tout se passe bien. Si nous introduisons les deux nombres à la suite l'un de l'autre, il semble naturel de penser que le compilateur ne comprendra pas ce que nous avons voulu introduire.

Lecture d'un réel :

Nous utilisons dans ce cas le programme suivant :

```

program prgInOut_3 ;
  uses crt ;
  var
    x  : real ;
begin
  write('Donnez un nombre réel : ') ;
  read(x);
  writeln('Merci pour ', x) ;
  readkey ;
end.

```

Valeurs introduites	Résultat de "x"
-12.25 ↵	-1.2250000000000000E+001
1e3 ↵	1.0000000000000000E+003
48 ↵	4.8000000000000000E+001
.5 ↵	5.0000000000000000E-001
3. ↵	3.0000000000000000E+001
1,12 ↵	Erreur

Il y a deux nouveautés par rapport à ce que nous avons vu précédemment. Nous devons taper le point "." pour séparer les décimales de la partie entière du nombre. Si nous tentons

d'introduire la virgule "," comme séparateur, une erreur se produit. L'autre nouveauté concerne le caractère "e" qui indique la notation scientifique de la manière suivante :

$$1e3 \Leftrightarrow 1 * 10^3 \Leftrightarrow 1000$$

Lecture d'un caractère :

Nous utilisons ici le programme suivant :

```
program prgInOut_4 ;
  uses crt ;
  var
    c1 : char ;
begin
  writeln('Donnez un caractère') ;
  read(c1);
  write('Merci pour ', c1) ;
  readkey ;
end.
```

Valeurs introduites	Résultat de "c1"
A ↵	A
AA ↵	A
^V ↵	^ (attention)
1a	1

Ici, le compilateur est sensé lire un caractère. Il ne peut bien sûr pas faire la différence entre un caractère qui pour nous signifie quelque chose (la lettre 'A' par exemple) d'un caractère insignifiant (l'espace). Donc, ici, nous devons faire attention à ce que nous tapons.

Lecture de deux caractères :

Nous utiliserons enfin ce programme :

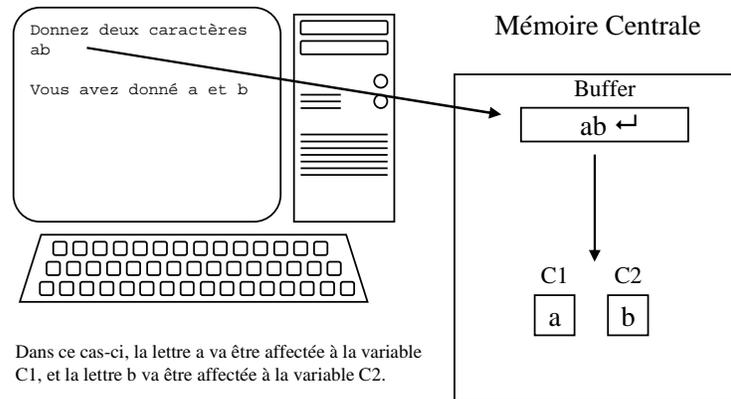
```
program prgInOut_5 ;
  uses crt ;
  var
    c1,c2 : char ;
begin
  writeln('Donnez deux caractères') ;
  read(c1,c2);
  writeln('Merci pour ', c1, ' et ',c2) ;
  readkey ;
end.
```

Valeurs introduites	Résultat de "c1"	Résultat de "c2"
AB ↵	A	B
A^B ↵	A	
12^^a ↵	1	2
^^AZ		

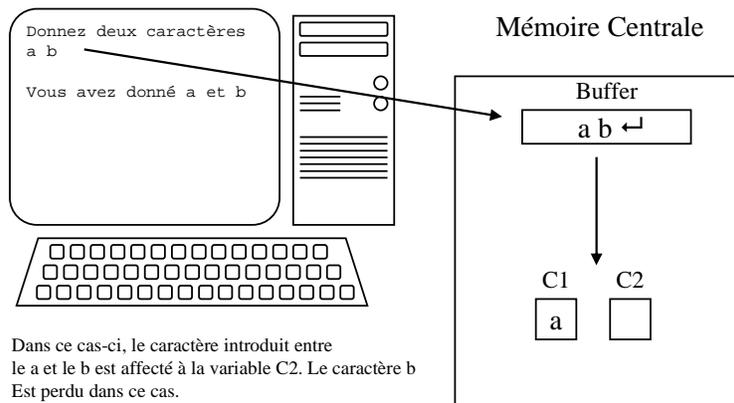
La remarque faite précédemment s'applique ici aussi : si nous introduisons un espace parmi nos deux caractères, le compilateur prendra cet espace comme caractère introduit. La solution dans ce cas – pour limiter le risque d'erreurs – est de faire la saisie des deux caractères au moyen de deux instructions distinctes.

Il résulte de ceci qu'il faut faire attention lorsque nous créons un programme. En effet, en pensant à l'instruction permettant l'affichage "Donnez deux caractères :", nous savons très bien ce que nous voulons dire, mais nous devons être certain que l'utilisateur comprendra la phrase de la même façon et pensera à ne pas mettre d'espaces entre les deux caractères introduits. La façon la plus sécurisante de travailler est d'effectuer l'opération au moyen de deux instructions distinctes. Nous limiterons ainsi le risque d'erreur. Toutefois, la sécurité n'est pas maximale : si nous voulons que l'utilisateur introduise une des 26 lettres de l'alphabet, ce sera à nous à réaliser la vérification dans notre code.

Règles générales d'une instruction de lecture



Lorsque l'utilisateur introduit des données via le clavier, le compilateur n'affecte pas directement les valeurs introduites aux variables concernées ; il utilise un "buffer"¹ pour stocker de manière temporaire chaque caractère frappé. Ensuite, lorsque l'utilisateur va presser sur la touche <enter> (symbolisée par le caractère "↵"), un pointeur va explorer ce tampon caractère par caractère jusqu'à ce que les variables soient garnies. Dans le cas de la saisie de deux caractères, le pointeur va rencontrer la lettre "a" qu'il va affecter à la variable C1, va rencontrer ensuite la lettre "b" qu'il va affecter à la variable C2. Nous comprenons mieux maintenant pourquoi, si nous introduisons un espace entre les deux lettres, le pointeur va affecter le caractère espace à la variable C2.

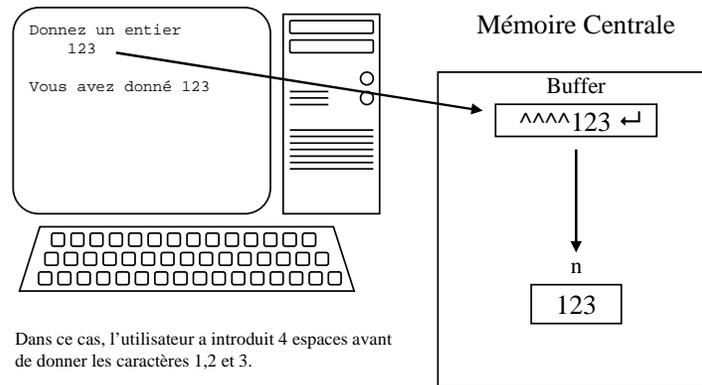


¹ buffer : mémoire tampon

Il est clair que selon ce principe, le type de variable va avoir un rôle important dans ce processus.

Pour une variable numérique

Si le caractère introduit est un espace, le pointeur se déplace jusqu'à ce qu'il rencontre un caractère différent. Ce caractère est alors marqué et le pointeur se déplace jusqu'à ce qu'il rencontre un nouvel espace ou le saut de ligne (représenté dans nos exemples par le caractère "↵"). Si les caractères forment une valeur acceptable, cette suite de caractères va être convertie en une donnée binaire (sur 16 bits pour un entier, sur 48 bits pour un réel).



Remarque : les caractères ^^^^ représentent
Le fait que le buffer est garni avec 4 caractères
Espaces avant les nombres 123.

Pour une variable de type caractère

Le caractère introduit va simplement être affecté à la variable de type caractère. Donc, il est important de ne pas taper d'espaces avant le caractère que l'on veut réellement affecter à la variable.

Mélange de types lors d'une saisie

Si nous mélangeons différents types lors d'une saisie, ce qui a été vu ci-dessus reste bien sûr d'application. La prudence est de rigueur pour éviter toute mauvaise interprétation. Nous reprenons un exemple simple qui nous permettra de réaliser les petits tests suivants

```
program prgInOut_6 ;
  uses crt ;
  var
    c1 : char ;
    n   : integer ;
begin
  writeln('Donnez un caractère et un nombre') ;
  read(c1,n);
  writeln('Merci pour ', c1, ' et ',n) ;
  readkey ;
end.
```

Valeurs introduites	Résultat de "c1"	Résultat de "n"
A25 ↵	A	25
A^^25 ↵	A	25
15^^A ↵	1	5
^A^12	Erreur	

La seule contrainte est de ne pas commencer par un espace, auquel cas une erreur se produit. Si l'utilisateur introduit d'abord 15 et ensuite A, le compilateur affecte le 1 à la variable C1, et il lui reste dès lors le 5 qu'il affecte à la variable n.

```

program prgInOut_7 ;
  uses crt ;
  var
    c1 : char ;
    n   : integer ;
begin
  writeln('Donnez un nombre et un caractère') ;
  read(n,c1);
  writeln('Merci pour ', n, ' et ',c1) ;
  readkey ;
end.

```

Valeurs introduites	Résultat de "n"	Résultat de "c1"
25A ↵	Erreur	
25A^ ↵	Erreur	
25^A ↵	25	^

Dans ce cas-ci, il est impossible de réaliser l'opération, à moins que de passer par une variable inutile, qui nous permettrait de saisir l'espace. Le programme suivant réalise cette opération.

```

program prgInOut_8 ;
  uses crt ;
  var
    c1,c : char ;
    n    : integer ;
begin
  writeln('Donnez un nombre et un caractère') ;
  read(n,c,c1);
  writeln('Merci pour ', n, ' et ',c1) ;
  readkey
end.

```

Syntaxe des instructions de lecture

```

read  (liste_de_variables);
readln (liste_de_variables);
readln ;

readln employé seul provoque simplement un saut de ligne.

```

Les instructions d'affichage

Exemple introductif

```

program prgInOut_9 ;
uses crt;
var n, p : integer ;
    x, y : real ;
    c1, c2 : char ;
    ok : boolean ;
begin
    n := 3 ;
    p := 125 ;
    x := 3.45e6 ;
    y := 2.0 ;
    c1 := 'e' ;
    c2 := 'i' ;
    ok := false ;
    writeln ('nombre', n) ;
    writeln ('nombre ', n) ;
    writeln (n, p) ;
    writeln (n, ' ', p) ;
    writeln (c1, c2) ;
    writeln (x, y) ;
    writeln ('cela est ', ok);
    readkey;
end.

```

Instruction	Affichage
writeln ('Nombre', n);	Nombre3
writeln ('Nombre ', n);	Nombre 3
writeln (n, p);	3125
writeln (n, ' ', p);	3 125
writeln (C1, C2);	ei
writeln (x, y);	3.4500000000E+006 2.0000000000E+000
writeln ('cela est ', ok)	cela est FALSE

Règles générales d'affichage par défaut

Nature de l'expression	Gabarit utilisé par défaut
Constante chaîne ('...')	Sa propre longueur
Entière (INTEGER)	Le minimum nécessaire à son affichage, compte tenu de sa valeur effective
Réelle (REAL)	17 (18 dans les versions inférieures à la 4.0)
Caractère (CHAR)	1
Booléenne (BOOLEAN)	4 (true) ou 5 (false)

Il est toutefois possible d'outrepasser ces règles par défaut et d'imposer son propre gabarit.

Comment modifier les gabarit par défaut

Exemple

```
program prgInOut_10 ;
uses crt;
var n, p : integer ;
    x : real ;
    c1, c2: char ;
    ok : boolean ;
begin
  n := 3 ;
  p := 125 ;
  x := 1.23456e2 ;
  c1 := 'e' ;
  c2 := 'i' ;
  ok := false ;

  writeln ('nombre', n :4);
  writeln (n:3, p:5);
  writeln (c1:3, c2:6, ok:7);
  writeln (p:2);
  writeln ('bonjour':3, ok:2);
  writeln (x:20);
  writeln (x:10);
  writeln (x:2);
  writeln (x:12:4);
  writeln (x:10:1);
  writeln (x:8:5);
  readkey;
end.
```

Instruction	Affichage
writeln ('Nombre', n :4);	Nombre 3
writeln (n:3, p:5);	3 125
writeln (c1:3, c2:6, ok:7);	e i FALSE
writeln (p:2);	125
writeln ('bonjour':3, ok:2);	bonjourFALSE
writeln (x:20);	1.23456000000E+02
writeln (x:10);	1.2346E+02
writeln (x:2);	1.2E+02
writeln (x:12:4);	123.4560
writeln (x:10:1);	123.5
writeln (x:8:5);	123.45600

Règles générales

Lorsque le gabarit défini est trop petit pour afficher la variable complète, PASCAL outrepassa le gabarit défini par l'utilisateur et impose un gabarit minimum, selon les règles suivantes :

Nature de l'expression	Gabarit minimum
Constante chaîne ('...')	Sa propre longueur
Entière (INTEGER)	Son nombre de chiffres, +1 pour les valeurs négatives
Réelle (REAL)	7 (8 pour un réel négatif)
Caractère (CHAR)	1
Booléenne (BOOLEAN)	4 (true) ou 5 (false)

Syntaxe des instructions d'affichage

```
write (liste_de_variables) ;  
writeln (liste_de_variables) ;  
writeln ;
```