

Micro-ordinateur

Micro-ordinateur moderne *Exemple*

Boîtier

Avec alimentation 300W +5V, +3.3V, +12V, -12V

Carte mère

Chipset Intel ou AMD Bus système à 800MHz

Deux slots PCI 64 bits et 3 slots PCI 32 bits

Slot PCI Express x 16

1 Carte graphique Media Accelerator 900

2 ports IEEE 1394 (Firewire)

2 ports USB 2.0

1 port parallèle pour imprimante

1 port série RS232

Micro-ordinateur moderne *Exemple*

Processeur

Processeur Intel Pentium 4 à 3 GHz / 1Mo de cache
ou AMD Athlon XP 3,2 GHz / 640 Ko de cache

Mémoire

1Go DDR RAM Bicanal 400 MHz

Carte graphique

Carte graphique PCI Express ATI TM X300 SE
VGA/DVI (Bi-écran)

Carte réseau

Carte réseau Ethernet Gigabits

Micro-ordinateur moderne *Exemple*

Carte son

Intégrée sur la carte mère

Ou avec chipset audio intégré Analog Devices

Disque dur

160Go SATA IDE 7200Tpm

Lecteur de disquette

Lecteur de disquettes 3.5" / 1.44 Mo

Ecran

Ecran Plat 15" FP E151 VGA - TCO 95

Micro-ordinateur moderne *Exemple*

Lecteur / graveur de DVD

Graveur de DVD DVD+RW (double couche)

Enceintes

Intégrées au moniteur
ou Enceintes Stéréo

Modem

Carte Modem PCI Data/Fax V.90 56K

Micro-ordinateur moderne *Exemple*

Clavier

CLAVIER AZERTY

Souris

SOURIS Optique 2 boutons + molette

Système d'exploitation

Microsoft Windows XP
ou Linux

Antivirus

Antivirus V7.6

2^{ème} partie
Programmation en Turbo
Pascal

2^{ème} partie
Programmation en Pascal

- 1) Notions de langage informatique
- 2) Conception d'un programme informatique
- 3) Un premier programme en Pascal
- 4) Données et types de données
- 5) Opérations et opérateurs
- 6) Fonctions et procédures usuelles
- 7) Instructions d'entrée-sortie
- 8) Instructions conditionnelles
- 9) Boucles
- 10) Sous-programmes

Chapitre 1
Notion de langage
informatique

Programme

Ensemble d'instructions permettant à un ordinateur d'exécuter une suite d'opérations déterminées.

Analogie : Ensemble de trous sur la bande perforée d'un orgue de Barbarie.

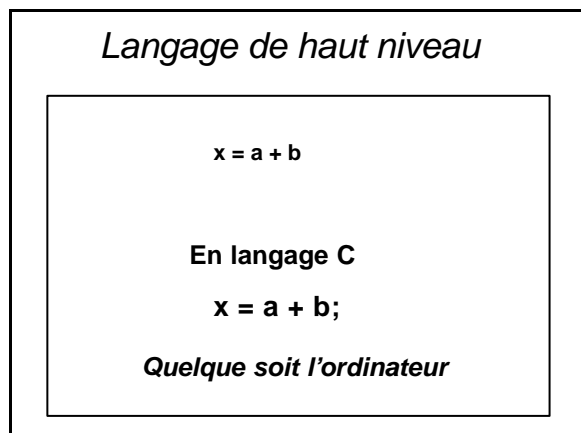
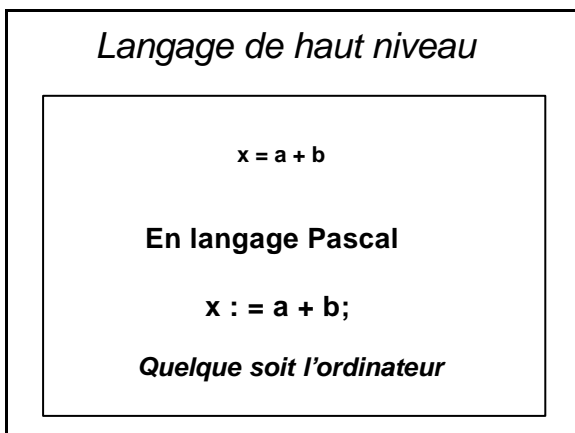
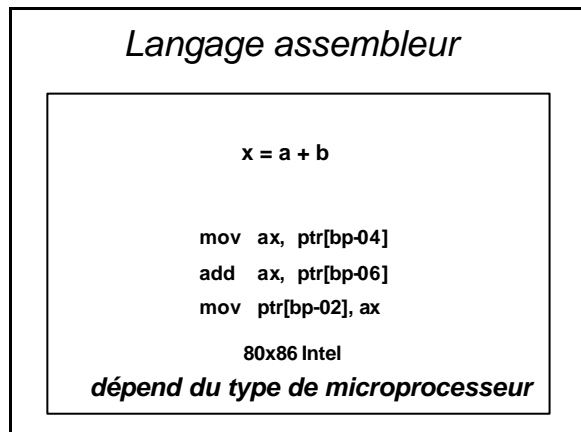
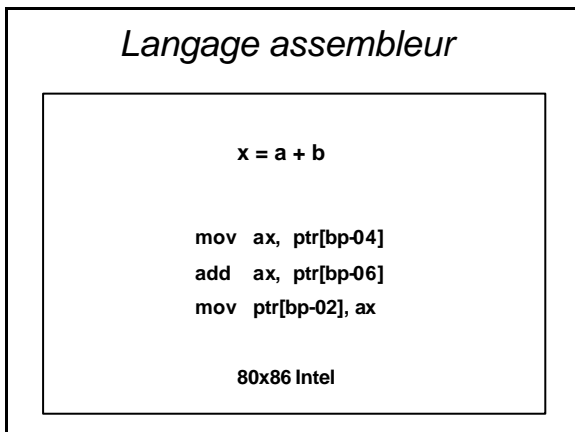
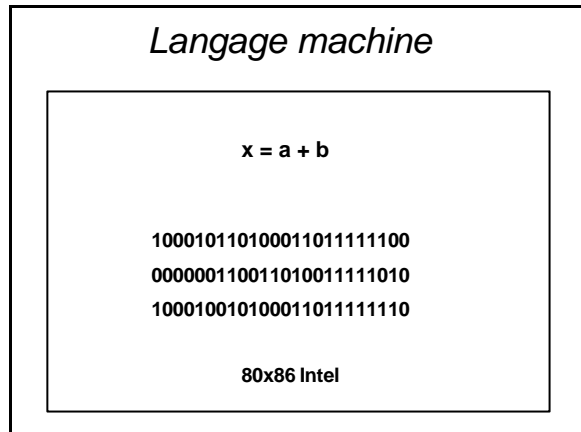
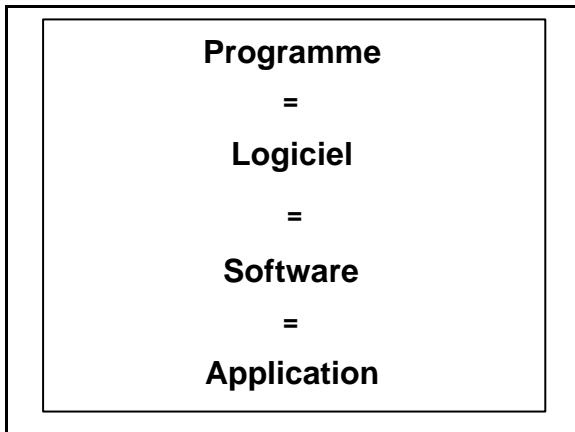
Programmation

Conception et écriture de programmes informatiques

Langage informatique

Approximation :

Langue écrite compréhensible par le programmeur et l'ordinateur



Langage de haut niveau

Pascal
C
Fortran
Cobol
Java

Langage de haut niveau

Pascal
C
Fortran
Cobol

Langage structurés

Langage de haut niveau

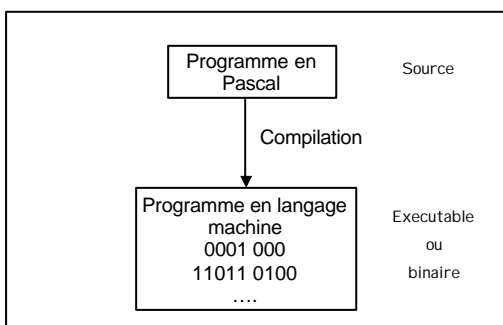
Pascal objet
Delphi
C++
Java

Langage à objets

Langage informatique

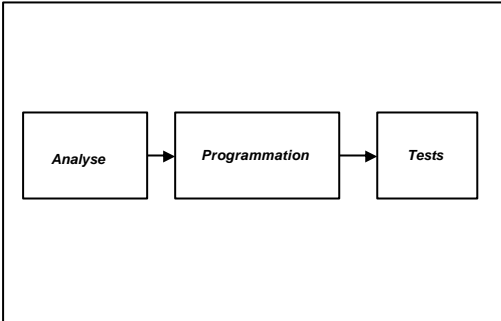
*Langue écrite compréhensible
par le programmeur et
l'ordinateur après traduction*

Compilation

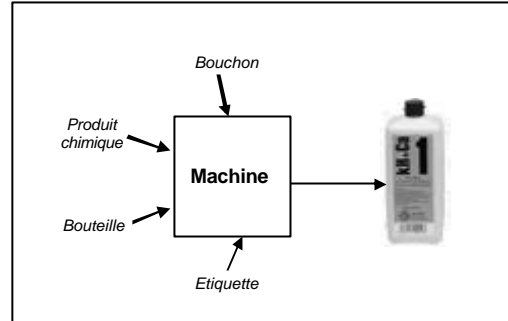


Chapitre 2
Conception d'un programme informatique

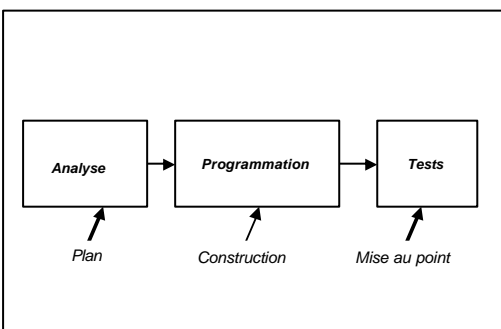
Conception d'un programme



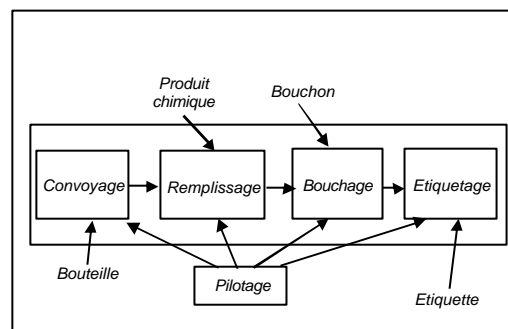
Conception



Conception d'un programme



Sous-programmes



Chapitre 3

Un premier programme en langage Pascal

Un premier programme

a = 2

b = 5

c = a + b

Afficher c

Un premier programme

Programme simple, pas besoin de sous-programme

Uniquement un programme principal

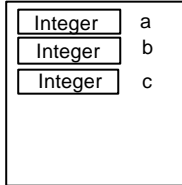
Un premier programme

program Exemple1;

Zone mémoire contenant un programme

Un premier programme

program Exemple1;

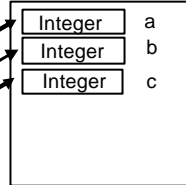


Créer 3 boîtes pour des entiers

Un premier programme

program Exemple1;

VAR a : INTEGER;
VAR b : INTEGER;
VAR c : INTEGER;

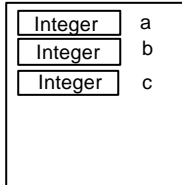


Créer 3 boîtes pour des entiers

Un premier programme

program Exemple1;

VAR a,b,c : INTEGER;

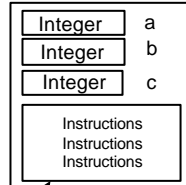


Ecriture plus compacte

Un premier programme

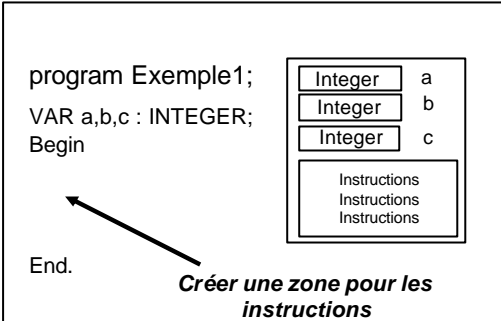
program Exemple1;

VAR a,b,c : INTEGER;

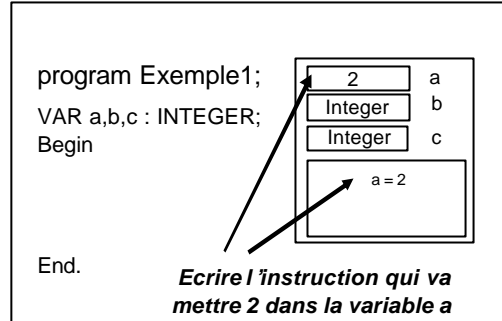


Créer une zone pour les instructions

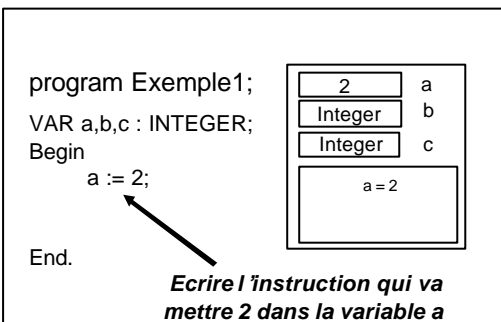
Un premier programme



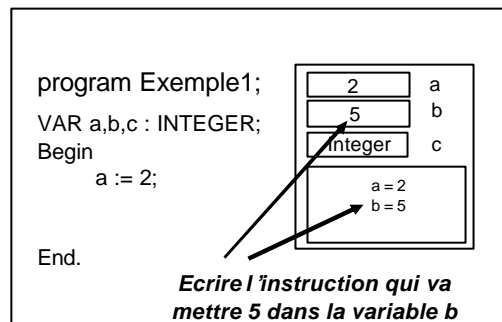
Un premier programme



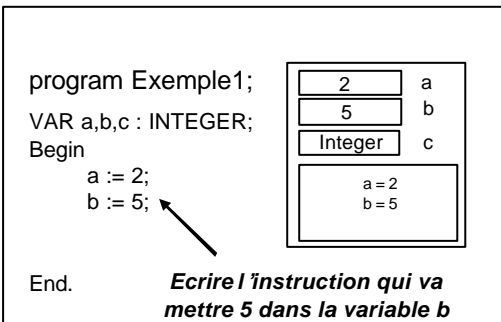
Un premier programme



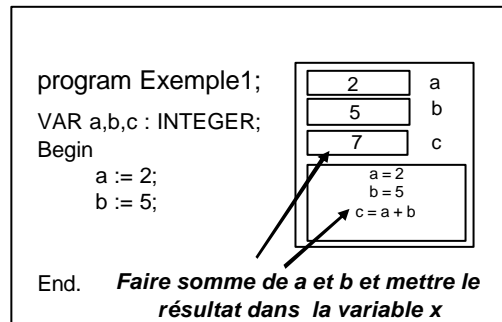
Un premier programme



Un premier programme



Un premier programme



Un premier programme

```
program Exemple1;  
VAR a,b,c : INTEGER;  
Begin  
  a := 2;  
  b := 5;  
  c := a+b;  
End.
```

2	a
5	b
7	c
a = 2 b = 5 c = a + b	

Faire somme de a et b et mettre le résultat dans la variable c

Un premier programme

```
program Exemple1;  
VAR a,b,c : INTEGER;  
Begin  
  a := 2;  
  b := 5;  
  c := a+b;  
End.
```

2	a
5	b
7	c
a = 2 b = 5 c = a + b afficher c	

Afficher la valeur de c

Un premier programme

```
program Exemple1;  
VAR a,b,c : INTEGER;  
Begin  
  a := 2;  
  b := 5;  
  c := a+b;  
  writeln(c);  
End.
```

2	a
5	b
7	c
a = 2 b = 5 c = a + b afficher c	

Afficher la valeur de c

Un premier programme

```
{Premier programme} ← Commentaire  
program Exemple1;  
VAR a,b,c : INTEGER;  
Begin  
  a := 2;  
  b := 5;  
  c := a+b;  
  writeln(c);  
End.
```

2	a
5	b
7	c
a = 2 b = 5 c = a + b afficher c	

Commentaires

```
{ Premier programme }  
  
(* Premier programme *)  
  
Non traduits en instructions
```

Chapitre 4 Données et types de données

Données

```

program Exemple1;
VAR a,b,c : INTEGER;
Begin
  a := 2;
  b := 5;
  c := a+b;
  writeln(c);
End.

```

2	a
5	b
7	c

a = 2
b = 5
c = a + b
afficher c

Variables

```

VAR surface : INTEGER;

```

↑ ↑

Identificateur Type

Surface Entier de -32.768 à +32.767

2 octets

Variables

Identificateur
63 caractères maximum

Surface du rectangle **INTERDIT**

Surface_du_rectangle **CORRECT**

6s **INTERDIT**

s6 **CORRECT**

Variables

Identificateur
Minuscules = MAJUSCULE

SURFACE_DU_RECTANGLE

Surface_du_rectangle

SuRFaCe_Du_recTANGLE

Type

```

VAR a, b, c : INTEGER;
Begin
  a := 2;
  b := 5;
  c := a + b;
End.

```

↑ Type

c vaut 7

Type

```

VAR a, b, c : INTEGER;
Begin
  a := 2;
  b := 5;
  c := a + b;
End.

```

Integer	a
Integer	b
Integer	c

↑

3 cases mémoire de 2 octets

c vaut 7

Types entiers

Type	Portée	Format
byte	0 à 255	8 bits, non signé
word	0 à 65535	16 bits, non signés
shortint	-128 à +127	8 bits, signé
integer	-32768 à + 32767	16 bits, signé
longint	-2147483668 à +2147483647	32 bits, signé

Type

```

VAR x, y, z : DOUBLE;

Begin
  x := 2.0;
  y := 5.0;
  z := x + y;
End.

z vaut 7.0
    
```

Diagram: An arrow points from the word "Type" to the declaration "DOUBLE;" in the code above.

Type

```

VAR x, y, z : DOUBLE;

Begin
  x := 2.0;
  y := 5.0;
  z := a + b;
End.

z vaut 7.0
    
```

Diagram: Three boxes labeled "Double" are stacked vertically and labeled x, y, and z. An arrow points from the text "3 cases mémoire de 8 octets" below to the boxes.

Types réels

Type	Domaine	Nombre de chiffres significatifs	Taille en octets
Single	$\pm 1,5 \cdot 10^{-45}$ à $\pm 3,4 \cdot 10^{+38}$	7..8	4
Real	$\pm 2,9 \cdot 10^{-39}$ à $\pm 1,7 \cdot 10^{+38}$	11..12	6
Double	$\pm 5,0 \cdot 10^{-324}$ à $\pm 1,7 \cdot 10^{+308}$	15..16	8
Extended	$\pm 1,9 \cdot 10^{+4951}$ à $\pm 1,1 \cdot 10^{+4032}$	19..20	10

Mélange de types

```

VAR a : INTEGER;
    x : DOUBLE;

Begin
  a := 2;
  x := 2;
  writeln(a);
  writeln(x);
End.
    
```

Diagram: An arrow points from the text "Entier dans un réel" to the assignment "x := 2;". A registered trademark symbol (®) is placed before "Conversion en 2.0".

Mélange de types

```

VAR a : INTEGER;
    x : DOUBLE;

Begin
  a := 2.0;
  x := 2.0;
  writeln(a);
  writeln(x);
End.
    
```

Diagram: An arrow points from the text "Error : Incompatible types" to the assignment "a := 2.0;".

Mélange de types

```
VAR a : INTEGER;  
    x : DOUBLE;  
  
Begin  
  x := 2.0;  
  a := x; ← Error :  
          Incompatible types  
  writeln(a);  
  writeln(x);  
End.
```

Mélange de types

Entier → Réel
Réel ✗ Entier

Type boolean

```
VAR chaise_couleur_verte : BOOLEAN; Boolean  
  
Begin  
  chaise_couleur_verte := true;  
  writeln (chaise_couleur_verte);  
End.
```

↑
1 case
mémoire de
1 octet

Type boolean

```
VAR chaise_couleur_verte : BOOLEAN;  
  
Begin  
  chaise_couleur_verte := true;  
  writeln (chaise_couleur_verte);  
End.
```

↙
True
ou
False

Type boolean

```
VAR chaise_couleur_verte : BOOLEAN;  
  
Begin  
  chaise_couleur_verte := 0; ← Error :  
                             Incompatible types  
  writeln (chaise_couleur_verte);  
End.
```

Type string

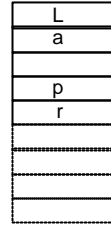
```
VAR u, v, w : STRING; ↖ Maximum 255  
caractères  
  
Begin  
  u := 'La programmation';  
  v := 'en Pascal';  
  w := u + v;  
End.
```

Type string

```
VAR u, v, w : STRING;  
  
Begin  
u := 'La programmation';  
v := 'en Pascal';  
w := u + v;  
End.
```

Type string

```
VAR u, v, w : STRING;  
  
Begin  
u := 'La programmation';  
v := 'en Pascal';  
w := u + v;  
End.
```



255 cases
mémoire de
1 octet

Type string

```
VAR u, v, w : STRING;  
  
Begin  
u := 'La programmation';  
v := 'en Pascal';  
w := u + v;  
End.
```

Ce n'est pas une addition
mais une concaténation

w vaut 'La programmation en Pascal'

Type string

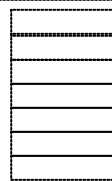
```
VAR a, b, c : INTEGER;  
VAR u, v, w : STRING;  
  
Begin  
u := 'La programmation';  
a := 5;  
c := a + u;  
End.
```

Types

Erreur : Incompatible types

Type string

```
VAR u : STRING[7];
```

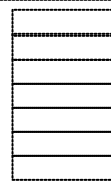


7 cases mémoire
de
1 octet

Type string

```
VAR u : STRING[7];
```

Maximum : 255



Type string

```

VAR u : STRING[7];

Begin
u := 'La programmation';
writeln(u);
End.

```

L
a
p
r
o
g

↖

Pas d'erreur

Type string

```

VAR u : STRING[7];

Begin
u := 'La programmation';
writeln(u);
End.

```

L
a
p
r
o
g

A l'écran :

La prog

Type char

```

VAR f : CHAR;

Begin
f := 'A';
writeln(f);
End.

```

Char

↑

1 case mémoire de 1 octet

Type char

```

VAR f : CHAR;

Begin
f := 'A';
writeln(f);
End.

```

0100 0001

↑

65 en mémoire

Type char

```

VAR f : CHAR; ↔ STRING[1]

Begin
f := 'A';
writeln(f);
End.

```

Type Tableau

Déclaration

```

VAR tableau:ARRAY[1 .. 100 ] OF INTEGER;

```

Integer
Integer
Integer
Integer
Integer
Integer
Integer

↑

100 cases d'integers = 200 octets

Type Tableau

Déclaration

```
VAR tableau : ARRAY[ 1 .. 3 , 1 .. 6 ] OF INTEGER;
```

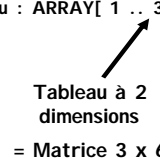


 Tableau à 2 dimensions
 = Matrice 3 x 6

Type Tableau

Déclaration

```
VAR tableau : ARRAY[ 1 .. 3 , 1 .. 6 , 1 .. 5 ] OF INTEGER;
```

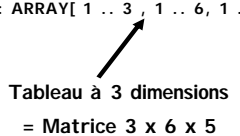


 Tableau à 3 dimensions
 = Matrice 3 x 6 x 5

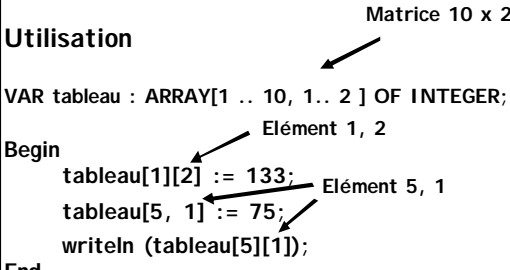
Type Tableau

Utilisation

```
VAR tableau : ARRAY[1 .. 10, 1.. 2 ] OF INTEGER;
```

Matrice 10 x 2

```
Begin
  tableau[1][2] := 133;
  tableau[5, 1] := 75;
  writeln (tableau[5][1]);
End.
```



 Élément 1, 2
 Élément 5, 1

Notations équivalentes

Type RECORD

C'est un type que l'on crée sur mesure

Etudiant

- Nom
- Prénom
- Note en chimie
- Note en physique
- Moyenne
- Classement

Type RECORD

Etudiant

- Nom
- Prénom
- Note en chimie
- Note en physique
- Moyenne
- Classement

String
String
Double
Double
Double
Integer

Aspect de la variable souhaitée

Type RECORD

On crée le type

```
TYPE ETUDIANT = RECORD
  nom : STRING;
  prenom : STRING;
  chimie : DOUBLE;
  physique : DOUBLE;
  moyenne : DOUBLE;
  classement : INTEGER;
END;
```

Type sur mesure

Description de la variable
Elle n'est pas encore créée

Type RECORD

Déclaration = Création de la zone en mémoire

```
VAR etudiant_1sp : ETUDIANT;
```

String
String
Double
Double
Integer

↑
La variable est créée en mémoire

Type RECORD

Zoom sur la variable créée

Variable : etudiant_1sp

etudiant_1sp.nom	String (255 octets)
etudiant_1sp.prenom	String (255 octets)
etudiant_1sp.chimie	Double (4o)
etudiant_1sp.physique	Double (4o)
etudiant_1sp.moyenne	Double (4o)
etudiant_1sp.classement	Integer (2o)

↑
Eléments de la variable

Type RECORD

Utilisation dans le programme

```
Begin
etudiant_1sp.nom := 'DUPONT';
etudiant_1sp.prenom := 'Jules';
etudiant_1sp.chimie := 16.0;
etudiant_1sp.physique := 15.0;
etudiant_1sp.moyenne := (etudiant_1sp.chimie +
etudiant_1sp.physique)/2;
etudiant_1sp.classement := 3;
writeln(etudiant_1sp.nom);
writeln(etudiant_1sp.moyenne);
End.
```

Type RECORD

Zoom sur la variable créée

Variable : etudiant_1sp

etudiant_1sp.nom	DUPONT
etudiant_1sp.prenom	Jules
etudiant_1sp.chimie	16.0
etudiant_1sp.physique	15.0
etudiant_1sp.moyenne	15.5
etudiant_1sp.classement	3

↑
Eléments de la variable

```
Program Notes;
TYPE ETUDIANT = RECORD
  nom : STRING;
  prenom : STRING
  chimie : DOUBLE;
  physique : DOUBLE;
  moyenne : DOUBLE;
  classement : INTEGER;
END;
```

Type

```
VAR etudiant_1sp : ETUDIANT;
```

Déclaration

```
Begin
etudiant_1sp.nom := 'DUPONT';
etudiant_1sp.prenom := 'Jules';
etudiant_1sp.chimie := 16.0;
```

Utilisation

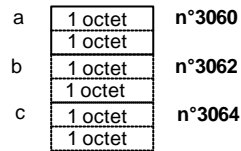
```
etudiant_1sp.physique := 15.0;
etudiant_1sp.moyenne := (etudiant_1sp.chimie +
etudiant_1sp.physique)/2;
etudiant_1sp.classement := 3;
writeln(etudiant_1sp.nom);
writeln(etudiant_1sp.moyenne);

End.
```


Type Pointeur

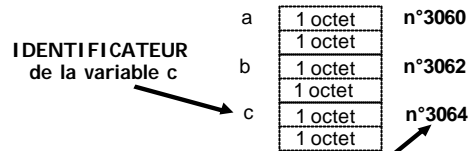
C'est un numéro de case mémoire

VAR a, b, c : INTEGER;



Type Pointeur

VAR a, b, c : INTEGER;



IDENTIFICATEUR
de la variable c

ADRESSE de la variable c

Type Pointeur

VAR a, b, c : INTEGER;

Begin

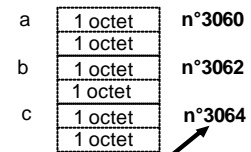
c := 5; Accès à la variable c
writeln(c); par son identificateur

End.

Type Pointeur

VAR a, b, c : INTEGER;

Comment accéder à la
variable c grâce à son
adresse ?

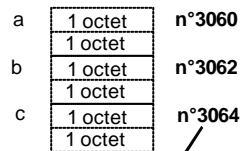


ADRESSE de la variable c

Type Pointeur

VAR a, b, c : INTEGER;

En stockant le n° de
case dans une variable
spéciale de type
pointeur



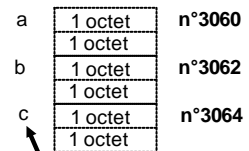
Variable de type pointeur



Type Pointeur

VAR a, b, c : INTEGER;

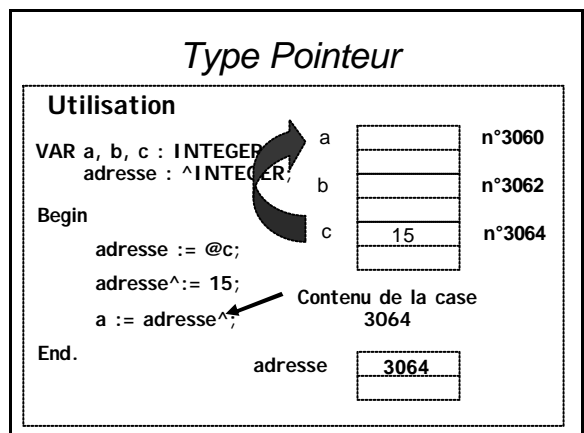
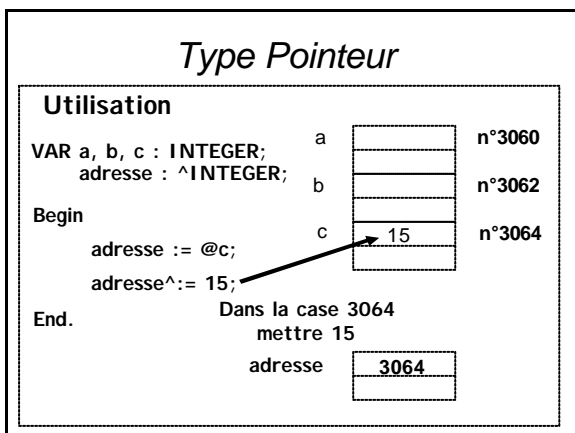
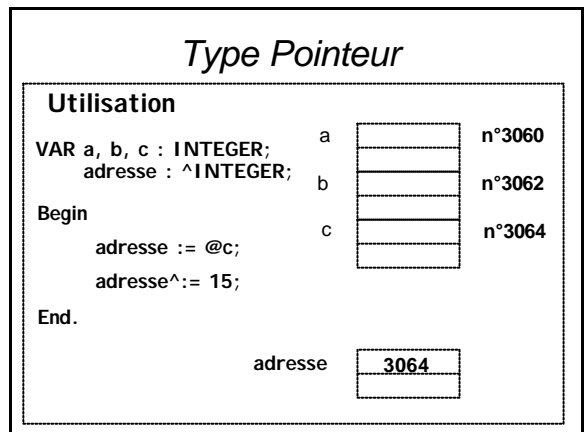
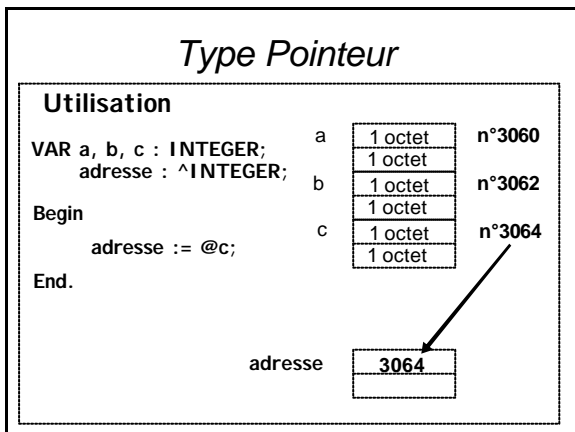
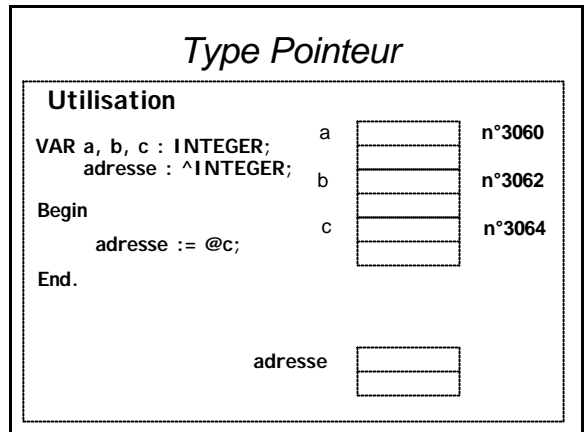
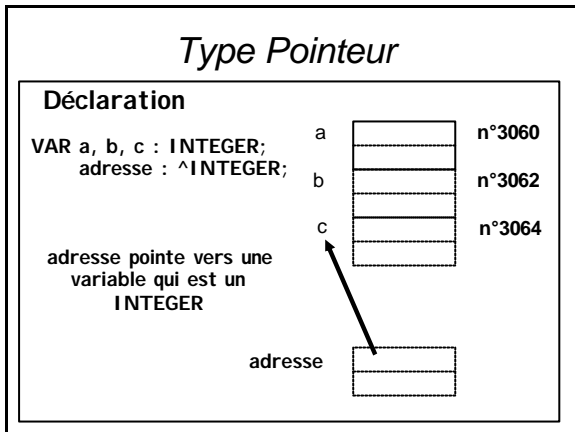
Pointeur : La valeur de
cette variable est le n°
de case d'une autre
variable



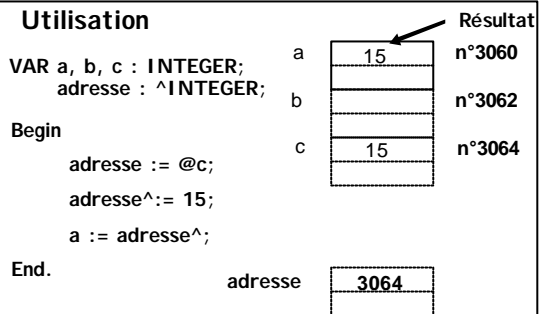
Pointe vers la variable c

adresse





Type Pointeur



Remarque sur les variables

Toujours déclarer les variables avant de les utiliser

Sinon pas de place en mémoire attribuée à cette donnée

Remarque sur les variables

```
program test;
Begin
    a := 5;
End.
```

Il manque
VAR a : INTEGER;

Erreur à la compilation :
Unknown identifier

Remarque sur les variables

```
program test;
VAR diamètre : DOUBLE;
Begin
    diamètre := 5.0;
End.
```

Caractères accentués interdits dans les identificateurs

Erreur à la compilation :
Illegal character

Remarque sur les integers

Avec la version de Pascal utilisée en TP

Integer sur 4 octets – ® LongInt

- 2.147.483.668 à + 2.147.483.647



Uniquement avec cette version

Chapitre 5

Opérations et opérateurs

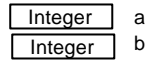
Affectation

VAR a, b : INTEGER;

Begin

a := 3;
b := 5;

End.



:= signifie mettre la
valeur de droite dans la
case de gauche

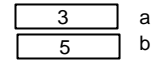
Affectation

VAR a, b : INTEGER;

Begin

a := 3;
b := 5;

End.



Résultat

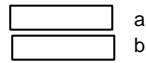
Affectation

VAR a, b : INTEGER;

Begin

3 := a;

End.



INTERDIT

Il n'y pas de
case pour 3

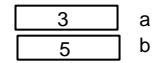
Affectation

VAR a, b : INTEGER;

Begin

a := 3;
b := 5;
a:=b;

End.



Etat de départ

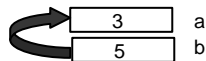
Affectation

VAR a, b : INTEGER;

Begin

a := 3;
b := 5;
a:= b;

End.



Mettre la valeur
de b dans a

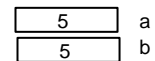
Affectation

VAR a, b : INTEGER;

Begin

a := 3;
b := 5;
a:= b;

End.



Résultat

Affectation

VAR a, b : INTEGER;

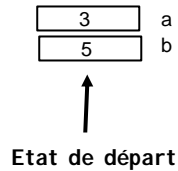
Begin

a := 3;

b := 5;

b:=a;

End.



Affectation

VAR a, b : INTEGER;

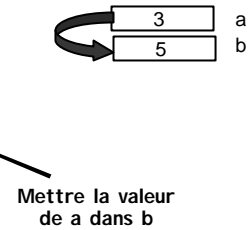
Begin

a := 3;

b := 5;

b:= a;

End.



Affectation

VAR a, b : INTEGER;

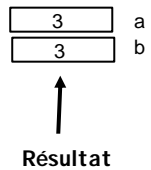
Begin

a := 3;

b := 5;

b:= a;

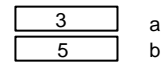
End.



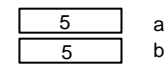
Affectation

a := 3;

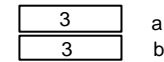
b := 5;



a:= b;



b:= a;



Affectation ® opération non commutative

Affectation

a:= b; \neq a = b

affectation \neq égalité

Opérateur arithmétiques

+ addition
- soustraction
* multiplication
/ division

Ne s'appliquent qu'aux
nombres (entiers ou réels)

Addition

$c := a + b$

←

De la droite vers la gauche
b ajouté à a
puis
affectation du résultat à c

Addition

```
VAR a, b, c : INTEGER;
Begin
  a := 3;
  b := 5;
  c:= a + b;
End.
```

Addition

$a := a + b$ **AUTORISÉ**

←

b ajouté à a
puis
affectation du résultat à a

Addition

```
VAR a, b, c : INTEGER;
Begin
  a := 3;
  b := 5;
  a:= a + b;
End.
```

Addition

```
VAR a, b, c : INTEGER;
Begin
  a := 3;
  b := 5;
  a:= a + b;
End.
```

Opérateur arithmétiques

+ - *

Opérandes	Résultat
Tous entiers	Entier
Au moins un réel	Réel

Opérateur arithmétiques

/

Opérandes	Résultat
Entiers ou réel	Réel

Division entière

```
VAR a, b, c : INTEGER;
```

```
Begin
```

```
  a := 1;
```

```
  b := 3;
```

```
  c := a div b;
```

```
End.
```

↖ Partie entière
du quotient de
a par b

Division entière

1 div 3 ® 0

3 div 3 ® 1

4 div 3 ® 1

Opérandes	Résultat
Tous entiers	Entier

Division entière

Si on fait

reel := 1 div 3;

⊘ Entier dans un réel

reel vaut 0.00000

Modulo

```
VAR a, b, c : INTEGER;
```

```
Begin
```

```
  a := 1;
```

```
  b := 3;
```

```
  c := a mod b;
```

```
End.
```

↖ Reste de la
division entière

Modulo

1 mod 3 ® 1

2 mod 3 ® 2

3 mod 3 ® 0

Opérandes	Résultat
Tous entiers	Entier

Modulo

Si on fait

reel := 1 mod 3;

⊆ Entier dans un réel

reel vaut 1.00000

Moins unaire

$a := - b$

Ne s'applique qu'à l'opérande b

® unaire

Moins unaire

$a := - b$

←

De la droite vers la gauche

- appliqué à b

puis

affectation du résultat à a

Moins unaire

VAR a, b : INTEGER;

Begin

b := 5;

a := - b;

End.

Moins unaire

VAR a, b : INTEGER;

Begin

b := 5;

a := - b;

End.

La valeur de b n'est pas modifiée

Moins unaire

VAR a : INTEGER;

Begin

a := 27;

a := - a;

End.

AUTORISÉ

Moins unaire

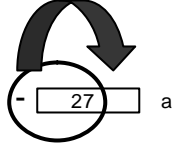
VAR a: INTEGER;

Begin

a := 27;

a := - a;

End.



Moins unaire

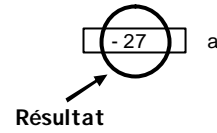
VAR a: INTEGER;

Begin

a := 27;

a := - a;

End.



Moins unaire

Opérande	Résultat
Entier	Entier
Réel	Réel

Opérateurs binaires

- NON binaire unaire
- ET binaire
- OU binaire
- Ou exclusif binaire
- Décalage vers la droite
- Décalage vers la gauche

Opérateurs binaires

Agissent au niveau bit

Opérande	Résultat
Entier	Entier

Non binaire unaire

```

0 0 0 0  0 1 0 1
↓ ↓ ↓ ↓  ↓ ↓ ↓ ↓
1 1 1 1  1 0 1 0
    
```

Inversion de tous les bits

Non binaire unaire

```

VAR a, b : BYTE;

Begin
  a := 5;
  b := NOT a;
End.

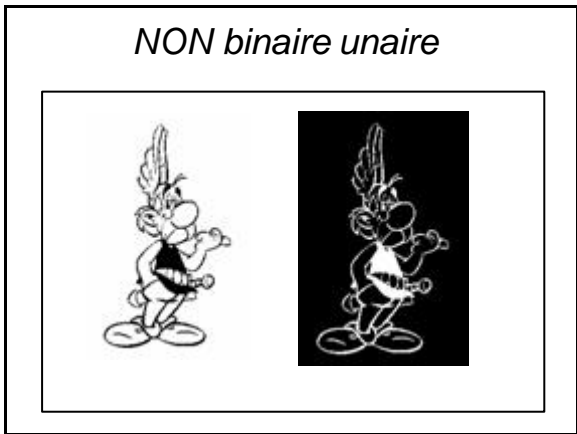
```

NON binaire unaire

```

a  0 0 0 0  0 1 0 1  =  5
      ↓
b  1 1 1 1  1 0 1 0  = 250

```



ET binaire

```

a  0 0 0 0  0 0 1 1  =  3
b  0 0 0 0  0 1 0 1  =  5
-----
c  0 0 0 0  0 0 0 1  =  1

```

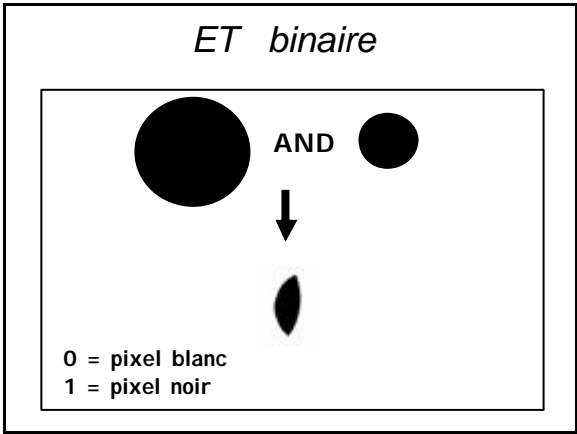
ET binaire

```

VAR a, b, c : BYTE;

Begin
  a := 3;
  b := 5;
  c := a AND b;
End.

```



OU binaire

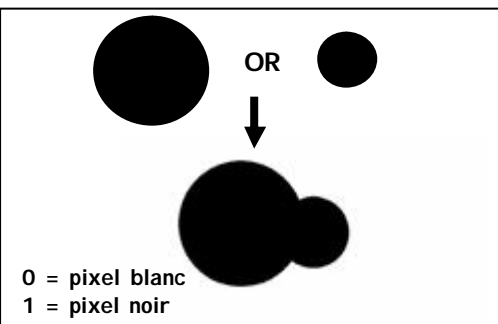
```
a    0 0 0 0  0 0 1 1  =  3
b    0 0 0 0  0 1 0 1  =  5
-----
c    0 0 0 0  0 1 1 1  =  7
```

OU binaire

```
VAR a, b, c : BYTE;

Begin
  a := 3;
  b := 5;
  c := a OR b;
End.
```

OU binaire



OU exclusif binaire

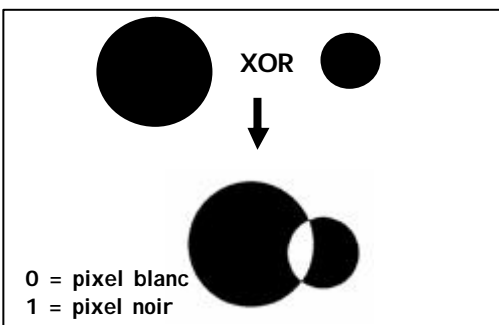
```
a    0 0 0 0  0 0 1 1  =  3
b    0 0 0 0  0 1 0 1  =  5
-----
c    0 0 0 0  0 1 1 0  =  6
```

OU exclusif binaire


```
VAR a, b, c : BYTE;

Begin
  a := 3;
  b := 5;
  c := a XOR b;
End.
```

OU exclusif binaire



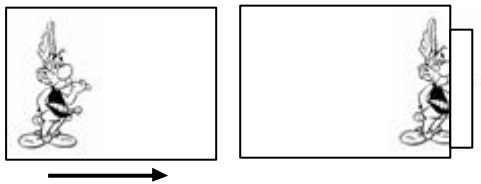
Décalage vers la droite

```
VAR a, b, c : BYTE;  
  
Begin  
  a := 5;  Shift Right  
  b := a SHR 1;  
  c := a SHR 3;  
End.
```


Décalage vers la droite

```
a    0 0 0 0 0 1 0 1 = 5  
      ────────────>  
b    0 0 0 0 0 0 1 0 = 2  
c    0 0 0 0 0 0 0 0 = 0
```

Décalage vers la droite



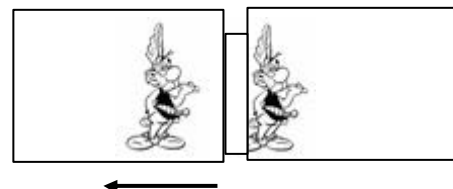
Décalage vers la gauche

```
VAR a, b, c : BYTE;  
  
Begin  
  a := 5;  Shift Left  
  b := b SHL 1;  
  c := b SHL 3;  
End.
```

Décalage vers la gauche

```
a    0 0 0 0 0 1 0 1 = 5  
      <──────────  
b    0 0 0 0 1 0 1 0 = 10  
c    0 0 1 0 1 0 0 0 = 40
```

Décalage vers la gauche



Opérateurs relationnels

Permettent de comparer
2 valeurs
et fournissent
un résultat booléen
(true ou false)

Opérateurs relationnels

>	supérieur
>=	supérieur ou égal
<	inférieur
<=	inférieur ou égal
=	égal
<>	différent

Opérateurs relationnels

```
VAR moyenne : DOUBLE;  
    admis : BOOLEAN;  
  
Begin  
    moyenne := 15.0;  
    admis := (moyenne >= 10.0);  
    writeln(admis);  
End.  
  
True
```

Opérateurs relationnels

```
VAR moyenne : DOUBLE;  
    admis : BOOLEAN;  
  
Begin  
    moyenne := 6.0;  
    admis := (moyenne >= 10.0);  
    writeln(admis);  
End.  
  
False
```

Opérateurs relationnels



Ne jamais utiliser l'opérateur
relationnel
=
avec un nombre réel

Opérateurs relationnels

Toujours tester la valeur d'un
réel entre 2 bornes

Valeur - e < réel < Valeur + e

Opérateurs booléens

Opérande	Résultat
Booléen	Booléen

Opérateurs booléens

moyenne à l'écrit \geq 10
admis

8 \leq moyenne à l'écrit < 10
oral

moyenne à l'écrit < 8
recalé

Opérateurs booléens

```
admis := moyenne_ecrit >= 10 ;  
recalé := moyenne_ecrit < 8 ;
```

Opérateurs booléens

```
Oral := (moyenne_ecrit < 10) AND (moyenne_ecrit >= 8);
```

↑ ↑ ↑
Booléen Booléen Booléen

Opérateurs booléens

moyenne à l'écrit < 8
redouble

moyenne à l'oral < 10
redouble

Opérateurs booléens

```
redouble := (moyenne_ecrit < 8) OR (moyenne_oral < 10);
```

↑ ↑ ↑
Booléen Booléen Booléen

Chapitre 6

Fonctions et procédures usuelles

Fonctions mathématiques

Fonction	Signification
abs	Valeur absolue
exp	Exponentielle
ln	Logarithme népérien
sqr	Élévation au carré
sqrt	Racine carrée
pi	Pi : 3.14159 ...
sin	Sinus
cos	Cosinus
arctan	Arc tangente

Fonction	Signification
round	Arrondi à l'entier le plus proche
trunc	Troncature
frac	Partie fractionnelle d'un réel
int	Partie entière d'un réel
inc	Incrémente
dec	Décrémente
power	Puissance

Abs

```

VAR i : INTEGER;
      r : DOUBLE;

Begin
  i := abs(-21);
  r := abs(- 3.9);
End.

  i      21
  r      3.9
  
```

Exp

```

VAR x, y : DOUBLE;

Begin
  x := 1.7;
  y := exp(x);
End.

  y      5.4739473917272
  
```

Ln

```

VAR x, y : DOUBLE;

Begin
  x := 1.7;
  y := Ln(x);
End.

  y      0.5306282510621
  
```

Ln

```
Begin
  writeln(Ln(-1.7));
End.
```

Erreur à la compilation

Ln

```
VAR x: DOUBLE;
Begin
  x := -1.7;
  writeln ( Ln(x) );
End.
```

Erreur à L'EXECUTION

Sqr

```
VAR x, y : DOUBLE;
Begin
  x := -1.7;
  y := sqr(x);
End.
```

y vaut 2.8899999

Sqrt

```
VAR x, y : DOUBLE;
Begin
  x := 25.0;
  y := sqrt(x);
End.
```

y vaut 5.0

Pi

```
VAR x : DOUBLE;
Begin
  x := Pi;
End.
```

x vaut 3.141592653589793

Sin Cos

```
y := sin (x);
z := cos(x);
```

en radians

Tangente

```
VAR x, y : DOUBLE;  
  
Begin  
  x := Pi/4;  
  y := sin(x)/cos(x);  
End.  
  
y vaut 0.999999999  
Problème d'arrondi
```

Opérateurs relationnels



Ne jamais utiliser l'opérateur
relationnel
=
avec un nombre réel

Opérateurs relationnels

Toujours tester la valeur d'un
réel entre 2 bornes

Valeur - e < réel < Valeur + e

Arctan

```
VAR x, y : DOUBLE;  
  
Begin  
  x := 1;  
  y := 4 * arctan(x)/Pi;  
End.  
  
y vaut 1.0
```

Round

```
VAR x : DOUBLE;  
  k : LONGINT;  
Begin  
  x := 3.4;  
  k := round(x);  
End.  
  
3.4 → 3   entier le plus  
3.6 → 4   proche  
3.5 → 4   le plus grand
```

Trunc

```
VAR x : DOUBLE;  
  k : LONGINT;  
Begin  
  x := 3.4;  
  k := trunc(x);  
End.  
  
3.4 → 3   partie  
3.6 → 3   entière d'un  
3.5 → 3   nombre réel
```

Round Trunc

Les résultats de
**Round
Trunc**
sont des entiers

Frac Int

Les résultats de
**Frac
Int**
sont des réels

Frac

```
VAR x, y : DOUBLE;  
  
Begin  
  x := 3.35;  
  y := frac(x);  
End.  
  
3.35 → 0.35  
3.6 → 0.6
```

Int

```
VAR x, y : DOUBLE;  
  
Begin  
  x := 3.35;  
  y := Int(x);  
End.  
  
3.35 → 3.0  
3.60 → 3.0
```

Nombres réels

Inc

```
VAR i : INTEGER;  
  
Begin  
  i := 1;            i vaut 2  
  Inc(i);            i vaut 7  
  Inc(i, 5);  
End.
```

Dec

```
VAR i : INTEGER;  
  
Begin  
  i := 1;            i vaut 0  
  Dec(i);            i vaut -5  
  Dec(i, 5);  
End.
```

Power

```

Program essai;
uses math; ← Pas nécessaire avec
              la version de TP

VAR x : DOUBLE;

Begin
  x := power(5, 2);
End.

      x ← 25.0
  
```

Unités

```

Program essai;
uses math; ← Bibliothèque de
              fonctions non
              standard

VAR x,y : DOUBLE;

Begin
  x := 180.0;
  y := degtorad(x);
End.

      y ← 3.1415926535897
  
```

- Unités*
- CRT : Fonctions d'utilisation de l'affichage en mode texte**
- Effacement d'écran
 - Fenêtres en mode texte
 - Changement de taille et de couleur d'affichage
- GRAPH : Fonctions graphiques**
- Traçage de courbes
 - Traçage de formes géométriques

- Unités*
- MATH : Fonctions mathématiques supplémentaires**
- Fonctions trigonométriques supplémentaires
 - Conversion d'angles
 - Fonction hyperboliques
 - Fonction statistiques
- GRAPH : Fonctions graphiques**
- Traçage de courbes
 - Traçage de formes géométriques

- Unités*
- SYSUTILS : Fonctions de gestion de l'ordinateur**
- Lecture et changement d'heure
 - Création de fichiers et de répertoires
 - Changement de nom, effacement de fichiers

Fonctions sur les chaînes

Fonction	Signification
chr	Code ASCII → caractère
ord	Caractère → code ASCII
val	Convertit un nombre en chaîne
str	Convertit une chaîne en nombre
length	Longueur d'une chaîne
copy	Copie une partie d'une chaîne
pos	Position d'une chaîne dans une autre
concat	Concaténation de 2 chaînes

Chr

```
VAR a : INTEGER;  
    c : CHAR;  
Begin  
  a := 65;  
  c := a;  
  c := chr(a);  
End.  
  
c := chr(65);  <<  c := 'A';
```

Ord

```
VAR a : INTEGER;  
    c : CHAR;  
Begin  
  c := 'A';  
  a := c;  
  a := ord(c);  
End.  
  
a := ord('A');  <<  a := 65;
```

Val

```
VAR x : DOUBLE;  
    s : STRING;  
    erreur : INTEGER;  
Begin  
  s := '4.18';  
  x := s;  
  val (s, x, erreur);  
End.  
  
x ← 4.18
```

Str

```
VAR x : DOUBLE;  
    s : STRING;  
Begin  
  x := 3.14;  
  s := x;  
  str (x, s);  
End.  
  
s ← '3.14'
```

Length

```
VAR i : INTEGER;  
    s : STRING;  
Begin  
  s := 'TOTO';  
  i := length(s);  
End.  
  
i ← 4
```

Copy

```
VAR s, t : STRING;  
Begin  
  s := 'La programmation';  
  t := copy(s,4,7);  
End.  
  
t ← 'program'
```

Pos

```
VAR s : STRING;  
    i : INTEGER;  
Begin  
    s := 'La programmation';  
    i := pos( 'gram', s);  
End.  
  
    i ← 7
```

Concat

```
VAR s,t,u : STRING;  
  
Begin  
    s := 'La programmation';  
    t := ' en Pascal';  
    u := concat(s, t);  
End.  
u ← 'La programmation en Pascal'
```

Concat

```
VAR s,t,u : STRING;  
  
Begin  
    s := 'La programmation';  
    t := ' en Pascal';  
    u := s + t; ← Plus simple  
End.  
u ← 'La programmation en Pascal'
```

Fonctions de l'opérating system

Fonction	Signification
getdir	Valeur du répertoire de travail
chdir	Change de répertoire de travail
mkdir	Créer un répertoire
rmdir	Supprimer un répertoire
halt	Arrêter l'exécution du programme

Getdir Chdir

```
VAR s,t : STRING;  
  
Begin  
    getdir(0,t);  
    writeln(t); ← C:\Dev-Pascal  
    s := '..';  
    chdir(s);  
    getdir(0,t);  
    writeln(t); ← C:\  
End.
```

Mkdir

```
VAR s : STRING;  
  
Begin  
    s := 'Totodir'  
    mkdir(s);  
End.  
  
Création du répertoire Totodir,  
sous le répertoire de travail
```

Rmdir

```
VAR s : STRING;
```

```
Begin
```

```
  s := 'Totodir'  
  rmdir(s);
```

```
End.
```

Suppression du répertoire
Totodir

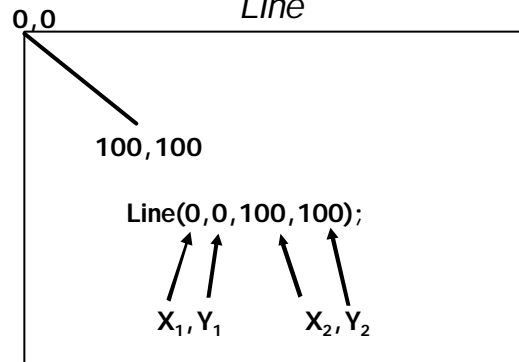
Fonctions de l'unité GRAPH ou WINGRAPH

Fonction	Signification
initgraph	Initialisation du mode graphique
closegraph	Termine le mode graphique
line	Trace une ligne entre deux points
circle	Trace un cercle
rectangle	Trace un rectangle

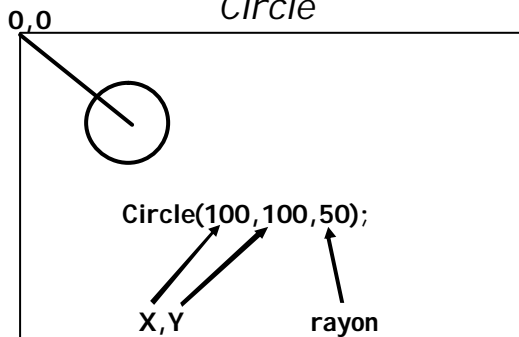
Initgraph Closegraph

```
program essai;  
Uses graph; ← Unité GRAPH  
var  
  gd, gm : smallint; ← Integer sur  
  PathToDriver : string; ← 16 bits dans la  
                           version des TP  
begin  
  gd:=detect;  
  gm:=0;  
  PathToDriver:='C:\Dev-Pascal\BGI';  
  InitGraph(gd, gm, PathToDriver);  
  if GraphResult<>grok then  
    halt;  
  CloseGraph;  
end.
```

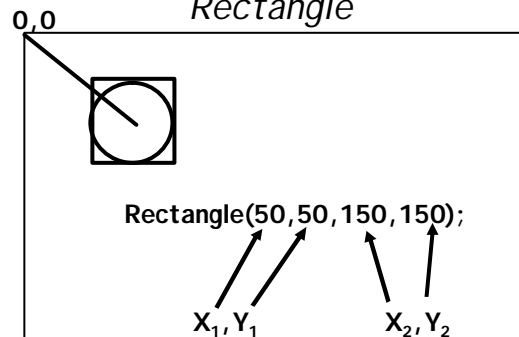
Line



Circle



Rectangle



Chapitre 7

Instructions d'entrées-sorties

Fonctions d'entrées-sorties

Fonction	Signification
writeln	Ecriture, puis à la ligne
write	Ecriture
readln	Lecture, puis à la ligne
read	Lecture
assign	Désigner un fichier par son nom
reset	Ouvrir un fichier en lecture
rewrite	Ouvrir un fichier en lecture-écriture
close	Fermer un fichier

Writeln

```

VAR x : DOUBLE;
Begin
  x := 1/3;
  writeln(x);
  writeln(x : 5 : 4);
End.
3.333333333333333E-001
0.3333      5 chiffres 4 décimales
  
```

Writeln

```

VAR x : DOUBLE;
Begin
  x := 1/3;
  writeln(x);
  writeln(x : 5 : 4);
End.
3.333333333333333E-001
0.3333      5 chiffres 4 décimales
  
```

A la ligne

Formatage
uniquement pour les
réels

Write

```

VAR x, y, z : DOUBLE;
Begin
  x := 1/3;
  y := 0.25
  z := 1/7
  write(x : 3 : 5);
  write(y : 5 : 4);
  write(z : 5 : 4);
End.
0.33330.25000.1429
  
```

Write

```

VAR x, y, z : DOUBLE;
Begin
  x := 1/3;
  y := 0.25
  z := 1/7
  write(x : 5 : 4);
  write(y : 5 : 4);
  write(z : 5 : 4);
End.
0.33330.25000.1429
  x | y | z
  
```

Write

```
VAR x, y, z : DOUBLE;  
Begin  
  x := 1/3;  
  y := 0.25  
  z := 1/7  
  write(x : 5 : 4);      0.33330.2500  
  writeln(y : 5 : 4);   0.1429  
  write(z : 5 : 4);  
End.
```

Write

```
VAR x, y, z : DOUBLE;  
Begin  
  x := 1/3;  
  y := 0.25  
  z := 1/7  
  write(x : 3 : 5);      0.33330.2500  
  writeln(y : 5 : 4);   0.1429  
  write(z : 5 : 4);      z  
End.
```

A la ligne APRES l'écriture de y

Readln

```
VAR a, b, c : INTEGER;  
Begin  
  readln(a);  
  writeln(a);  
End.
```

Le programme S'ARRETE ET ATTEND que l'on ait tapé la valeur de a et RETURN

Read

```
VAR a, b, c : INTEGER;  
Begin  
  read(a);  
  writeln(a);  
End.
```

Le programme S'ARRETE ET ATTEND que l'on ait tapé la valeur de a et RETURN

Au clavier pas de différence entre READ et READLN (mais différence pour les fichiers)

Writeln dans un fichier

```
VAR a, b, c : INTEGER;  
    F : Text;  
Begin  
  Assign (F, 'test.txt');  
  Rewrite (F);  
  a := 3;  
  b := 4;  
  c := 10;  
  writeln(F, a);  
  writeln(F, b);  
  writeln(F, c);  
End.
```

Writeln dans un fichier

```
VAR a, b, c : INTEGER;  
    F : Text; ← Type FICHER TEXTE  
Begin  
  Assign (F, 'test.txt');  
  Rewrite (F);  
  a := 3;  
  b := 4;  
  c := 10;  
  writeln(F, a);  
  writeln(F, b);  
  writeln(F, c);  
End.
```


Writeln dans un fichier

```
VAR a, b, c : INTEGER;
    F : text;
Begin
  Assign (F, 'test.txt');
  Rewrite (F);
  a := 3;
  b := 4;
  c := 10;
  writeln(F, a);
  writeln(F, b);
  writeln(F, c);
End.
```

Dans le fichier texte
3
4
10
Writeln uniquement dans un fichier texte

Write dans un fichier

```
VAR a, b, c : INTEGER;
    F : File of INTEGER;
Begin
  Assign (F, 'test.tmp');
  Rewrite (F);
  a := 3;
  b := 4;
  c := 10;
  write(F, a);
  write(F, b);
  write(F, c);
End.
```

Type FICHER D'ENTIERES

Write dans un fichier

```
VAR a, b, c : INTEGER;
    F : File of INTEGER;
Begin
  Assign (F, 'test.tmp');
  Rewrite (F);
  a := 3;
  b := 4;
  c := 10;
  write(F, a);
  write(F, b);
  write(F, c);
End.
```

Writeln interdits en dehors d'un fichier texte

Write dans un fichier

```
VAR a, b, c : INTEGER;
    F : File of INTEGER;
Begin
  Assign (F, 'test.tmp');
  Rewrite (F);
  a := 3;
  b := 4;
  c := 10;
  write(F, a);
  write(F, b);
  write(F, c);
End.
```

Enregistrement des valeurs en représentation en complément à 2 dans le fichier

Readln d'un fichier

```
VAR a, b, c : INTEGER;
    F : Text;
Begin
  Assign (F, 'test.txt');
  Reset (F);
  readln(F, a);
  readln(F, b);
  readln(F, c);
End.
```

Type FICHER TEXTE
On lit les nombres comme s'ils étaient tapés au clavier
Chaque nombre est à la ligne dans le fichier

Read d'un fichier

```
VAR a, b, c : INTEGER;
    F : File of INTEGER;
Begin
  Assign (F, 'test.tmp');
  Reset (F);
  read(F, a);
  read(F, b);
  read(F, c);
End.
```

Type FICHER D'ENTIERES
Lecture des entiers directement à partir de leur représentation en complément à 2

Chapitre 8

Instructions conditionnelles

If

```

VAR moyenne : DOUBLE;
    resultat : STRING;

Begin
    resultat := 'RECALE';
    readln(moyenne);
    if ( moyenne >= 10.0 ) then
        begin
            resultat := 'RECU';
        end;
    writeln(resultat);
End.

```

If

```

VAR moyenne : DOUBLE;
    resultat : STRING;

Begin
    resultat := 'RECALE';
    readln(moyenne);
    if ( moyenne >= 10.0 ) then
        resultat := 'RECU';
    writeln(resultat);
End.

```

Begin et end pas obligatoire si une seule instruction

If

```

if ( condition ) then
    begin
        executé si la condition est vraie
        exécuté si la condition est vraie
        ... ..
    end;
suite exécutée dans tous les cas
... ..

```

If Else

```

if ( condition ) then
    begin
        executé si la condition est VRAIE
        exécuté si la condition est VRAIE
    end;
Else
    begin
        executé si la condition est FAUSSE
        exécuté si la condition est FAUSSE
    end;
suite exécutée DANS TOUS LES CAS

```

If Else

```

VAR moyenne : DOUBLE;
    resultat : STRING;

Begin
    readln(moyenne);
    if ( moyenne >= 10 ) then
        resultat := 'RECU';
    else
        resultat := 'RECALE';
    writeln(resultat);
End.

```

Begin et end pas obligatoire si une seule instruction

Case

```
VAR i : INTEGER;  
Begin  
  readln( i );  
  case i of  
    0,2,4,6,8,10: writeln(' i est un nombre pair');  
    1,3,5,7,9:   writeln(' i est un nombre impair');  
  else  
    writeln(' i est negatif ou > a 10');  
End.
```