Module PROG & ALGORITHMIQUE
TD 1
Les bases

Hervé Owsinski
Herve.Owsinski@uphf.fr

2026-2027

1 Présentation

Module Les bases de l'algorithmique Semestre 1
Hervé Owsinski
Herve.Owsinski@uphf.fr

Ce module comporte

- 16 cours/TD (1h30) [ALGO]
- 4 TP (3h) [ALGO et PROG]

2 Que va-t-on y faire ?

Définitions

Définitions 1 : ALGORITHME (Abstraction)
Un **algorithme** est une suite **finie** et **non-ambiguë** d'opérations ou d'instructions permettant de résoudre un problème.

(Wikipédia) L'**algorithmique** est l'ensemble des règles et techniques qui sont impliquées dans la définition et la conception d'algorithmes.

Définition 2 : PROGRAMME (Implémentation)
Un **programme** (informatique) est la traduction d'un algorithme dans un langage de programmation donné.

Il s'agit donc d'implémenter l'idée abstraite de l'algorithme en la transformant en réalisation concrète.

Définitions

Distinction entre pseudo-langage et langage de programmation
Un **algorithme** peut être écrit dans un langage proche du langage naturel : c'est ce qu'on appelle le **pseudo-langage**. En TD on utilise uniquement le pseudo-langage.

Un **programme** doit être écrit dans un langage compris par la machine sur laquelle il est exécuté. C'est le **langage de programmation**. Voir TP.

Définition 3 : Programmation IMPERATIVE
Dans ce module, nous allons aborder la **programmation impérative**. Il s'agit d'un **paradigme de programmation** (grossièrement une façon de programmer) dans laquelle l'ordinateur exécute les instructions séquentiellement, l'une après l'autre. Cela veut dire que l'ordre dans lequel sont écrites les instructions est (très) important.

Définitions

Qu'est-ce qu'une DECLARATION ?
Tout **objet** manipulé dans un algorithme doit être déclaré avant de pouvoir être utilisé.

Un **objet** est caractérisé par un **identifiant** (c'est-à-dire un nom) unique.

Il existe deux catégories d'objets manipulés dans les algorithmes : les **constantes** et les **variables**.

Définitions

Définition 4 : CONSTANTE

Une **constante** correspond à un objet dont la valeur ne change pas pendant le déroulement de l'algorithme.

La constante peut être implicite (par exemple 1 ou 2.5) ou explicite, c'est-à-dire définie par le programmeur (par exemple MAX = 100). L'identifiant d'une constante est généralement composé uniquement de **majuscules**. Lorsque l'identifiant d'une constante contient plusieurs mots, on les sépare par le caractère '_'.

Exemple : PRIX_PAQUET = 20

Définitions

Définition 5 : VARIABLE

Une **variable** est un objet dont la valeur peut changer en cours de traitement.

La valeur de la variable appartient à un ensemble ou à un domaine de définition : c'est la notion de **type**. L'identifiant d'une variable commence généralement par une **minuscule**. Lorsque l'identifiant contient plusieurs mots, on les sépare par le caractère '_'.

Exemple : total_a_payer

Définitions

Définition 6 : TYPE

Le **type** d'une variable définit l'ensemble des valeurs que peut prendre la variable et l'ensemble des actions (ou opérations) que l'on peut effectuer sur la variable.

Les types simples :

- le type **Booléen** : {faux, vrai}
- le type **Caractère** : { 'a','b','z','A','B','Z','0','1','9',' ',' ?',...}
- le type **Entier** : c'est l'équivalent de l'ensemble Z en mathématique.
- le type **Réel** : c'est l'équivalent de l'ensemble R en mathématique.

Définitions

Définitions 7 : EN-TETE et CORPS

Un algorithme est toujours composé d'un **en-tête** et d'un **corps**.

L'**en-tête** regroupe le nom de l'algorithme et les données qu'il utilise. [L2-3]

Le **corps** est la suite d'instructions à exécuter séquentiellement. Il correspond au traitement à réaliser. Il commence (en pseudo-langage) par le mot clé **début** et se termine par le mot clé **fin**. [L04-08]

```
1 CONST K = 8 ;
2 Programme exo0
3     VAR a, b : Entier ;
4 début
5     a ← K ;
6     b ← a * 2 ;
7     Écrire(b) ;
8 fin
```

Définitions

Définitions 8 : PRIMITIVE 1/3

Une **primitive** est une action de base du langage algorithmique, une action qui peut être exécutée directement par l'ordinateur.

Les 3 primitives de ce module sont : **affectation, lecture, écriture**.

L'**affectation** permet d'attribuer (ou affecter) une valeur, ou le résultat d'une expression, à une variable. Elle a pour but de modifier la valeur de la variable.

L'affectation est représentée par le symbole ←

```
a ← 12 ;
```

(Attention : le = ne correspond pas à une affectation en pseudo-langage, ce n'est pas du Python)

Définitions

Définitions 8 : PRIMITIVE 2/3

Une **primitive** est une action de base du langage algorithmique. Elle correspond à une action qui peut être exécutée directement par l'ordinateur.

Les 3 primitives de ce module sont : **affectation, lecture, écriture**.

La **lecture** permet à l'utilisateur de saisir une donnée (valeur, texte) au clavier. La donnée est récupérée dans l'algorithme et elle est affectée à une variable déjà déclarée et de même type que ce que l'on va lire.

```
Lire(a) ;
```

On stoppe et on demande à l'utilisateur de fournir une valeur. L'algorithme ne continue pas tant que la variable a n'est pas remplie.

Définitions

Définitions 8 : PRIMITIVE 3/3

Une **primitive** est une action de base du langage algorithmique. Elle correspond à une action qui peut être exécutée directement par l'ordinateur.

Les 3 primitives de ce module sont : **affectation, lecture, écriture**.

L'**écriture** permet à l'algorithme d'afficher une donnée (valeur, texte) à l'écran.

```
Ecrire(a) ;
```

On considère que l'algorithme va Écrire le contenu de a sur l'écran (imaginaire).

Définitions

Nos choix sur le pseudo-langage

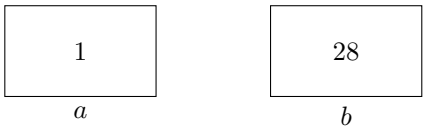
Un algorithme écrit en pseudo-langage respecte un ensemble de règles. Celles-ci reposent en partie sur la définition de quelques mots clés.

- la déclaration d'une constante se fait avant **début** via le mot clé **Programme**
- la première ligne de l'algorithme est **Programme nom_du_programme**
- la déclaration d'une variable se fait grâce au mot clé **VAR**
- le corps de l'algorithme commence avec le mot clé **début** et se termine avec le mot clé **fin**
- l'affectation est représentée par le symbole ←
- l'instruction de lecture est traduite par le mot clé **Lire**
- l'instruction d'écriture est traduite par le mot clé **Écrire**
- La fin d'une instruction est marquée par un point-virgule ;

3 Un exerice corrigé

Exo 0 en pseudo-code - déroulé complet

```
Programme exo_0_a
VAR a, b : Entier ;
début
    a ← 8 ;          /* a référence 8 */
    b ← a × 4 ;      /* b référence 8*4 = 32 */
    a ← 1 ;          /* a référence maintenant 1 */
    b ← b - 4 × a;   /* b référence 32-4*1=32-4=28 */
    Écrire(a, b) ;   /* On affiche 1 et 28 */
fin
```



a b

Exo 0 en Python

```python
# programme principal


a = 8               # a référence 8
b = a * 4           # b référence 8*4 = 32
a = 1               # a référence maintenant 1
b = b - 4*a         # b référence 32-4*1=32-4=28
print(a, " ", b)    # On affiche 1 et 28
```

Exo 0 en "Python-algo" version 1

```python
# programme principal
a:int               # a défini comme int
b:int               # b défini comme int
a = 8               # a référence 8
b = a * 4           # b référence 8*4 = 32
a = 1               # a référence maintenant 1
b = b - 4*a         # b référence 32-4*1=32-4=28
print(a, " ", b)    # On affiche 1 et 28
```

Exo 0 en "Python-algo" version 2

```python
# programme principal


a:int = 8           # a référence 8
b:int = a * 4       # b référence 8*4 = 32
a = 1               # a référence maintenant 1
b = b - 4*a         # b référence 32-4*1=32-4=28
print(a, " ", b)    # On affiche 1 et 28
```

Exo 0 en Python version 3

```python
# programme principal


a:int = 8 ;         # a référence 8
b:int = a * 4 ;     # b référence 8*4 = 32
a = 1 ;             # a référence maintenant 1
b = b - 4*a ;       # b référence 32-4*1=32-4=28
print(a, " ", b) ; # On affiche 1 et 28
```

Exo 0

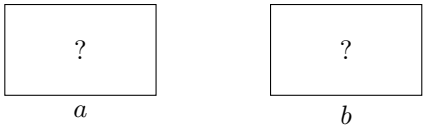
DEUX REMARQUES IMPORTANTES

1. Lors d'une affectation, on commence par évaluer ce qu'il y a à **droite PUIS on l'affecte à gauche**.

2. A cause de la **séquentialité des instructions**, un changement de valeur de variable ne modifie pas une autre variable déjà calculée : l'affectation ne définit pas une formule constamment vraie. Pas de recalcul automatique.

Exo 0 bis

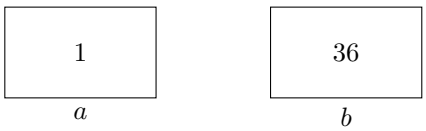
```
Programme exo_0_b
VAR a, b : Entier ;
début
    Lire(a) ;
    b ← a × 4 ;
    a ← 1 ;
    b ← b - 4 × a;
    Écrire(a, b) ;
fin
```



Que se passe-t-il si on tape 10 au départ ?

Exo 0 bis en pseudo-code, déroulé total

```
Programme exo_0_b
VAR a, b : Entier ;
début
    Lire(a) ;           /* a référence 10 */
    b ← a × 4 ;         /* b référence 10*4 = 40 */
    a ← 1 ;             /* a référence 1 */
    b ← b - 4 × a;      /* b référence 40 - 4*1 = 36 */
    Écrire(a, b) ;      /* On affiche 1 et 36 */
fin
```



Exo 0 bis en Python

```
# programme principal


a = int(input())      # a référence 10
b = a * 4             # b référence 10*4 = 40
a = 1                 # a référence maintenant 1
b = b - 4*a           # b référence 40-4*1=36
print(a, " ", b)      # On affiche 1 et 36
```

Important : en Python, **Lire(a)** s'implémente en trois temps :

1. D'abord, on récupère la saisie clavier de l'utilisateur avec **input()**
2. Ensuite, on la transtype en entier (integer en anglais) avec **int(input())**
3. Enfin, on affecte à gauche ce qu'on vient d'évaluer à droite avec **a = ...**

4 Exercices

Exo 1

```
Programme exo1
  VAR A, B, C : Entier ;
début
  Écrire("Entrer un entier : ") ; Lire(A) ;
  B ← 3 ;
  C ← -1 ;
  A ← A × B;
  B ← A + B ;
  A ← 2×A - B ;
  B ← B × -C ;
  C ← -B + C ;
  B ← B - A ;
  Écrire(A, B, C) ;
fin
```

?	?	?
A	B	C

Exo 1 - pseudo-code, déroulé final

```
Programme exo1
  VAR A, B, C : Entier ;
début
  Écrire("Entrer un entier : ");Lire(A) ;  /* A référence Ao */
  B ← 3 ;                                  /* B référence 3 */
  C ← -1 ;                                 /* C référence -1 */
  A ← A × B;       /* A référence Ao*3 = 3Ao */
  B ← A + B ;      /* B référence 3Ao + 3 */
  A ← 2×A - B ;    /* A réf 2*(3Ao) - (3Ao + 3) = 3Ao - 3 */
  B ← B × -C ;     /* B réf  (3Ao + 3) * 1 = 3 Ao + 3 */
  C ← -B + C ;     /* C réf -3Ao - 3 - 1 = -3Ao -4 */
  B ← B - A ;      /* B réf 3Ao + 3 - (3Ao - 3) = 6 */
  Écrire(A, B, C); /* Affiche (3Ao - 3), 6 et (-3Ao - 4) */
fin
```

$3A_0 - 3$	6	$-3A_0 - 4$
A	B	C

Exo 1 - test en "Algo-Python"

```
# programme principal
print("Entrer un entier")
a:int = int(input())          # Ao vaut 10 sur l'exemple
b:int = 3
c:int = -1
a = a * b
b = a + b
a = 2*a - b                   # 3*Ao - 3
b = b * -c
c = -b + c                    # -3*Ao - 4
b = b - a                     # 6
print(a, " ", b, " ", c)
```

Si on lance ce programme, qu'on tape 10, on obtient :

```
Entrer un entier
10
27   6   -34
```

Remarque

Convention : minuscule au début pour une variable
Habituellement en informatique, on utilise uniquement un nom entièrement composé de MAJUSCULES pour les CONSTANTES et un nom commençant par une minuscule pour les variables.

Exo 2

Exercice 2 : Que fait l'algorithme suivant :

```
Programme exo2
    VAR x, y : Réel ;
début
    Lire(x) ; Lire(y) ;   /* x réf X et y réf Y          */
    Écrire(x, y) ;
    x ← x + y;
    y ← x - y;
    x ← x - y;
    Écrire(x, y) ;
fin
```

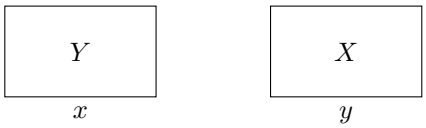
Proposer (Écrire) un autre algorithme pour faire la même chose.

Remarque : nous ne pouvons pas savoir ce que l'utilisateur va taper sur le clavier et on ne peut pas réfléchir sur les valeurs réelles attribuées pour les variables x et y. On symbolisera ces valeurs par X et Y. X représente l'entier qui sera lu depuis le clavier et qu'on stockera dans x.

Exo 2

Exercice 2 : **Que fait l'algorithme suivant** :

```
Programme permuter
    VAR x, y : Réel ;
début
    Lire(x) ; Lire(y) ;   /* x réf X et y réf Y          */
    Écrire(x, y) ;        /* Affiche X et Y              */
    x ← x + y;            /* 1 - x ref X + Y              */
    y ← x - y;            /* 2 - y ref X+Y-Y = X          */
    x ← x - y;            /* 3 - x ref X+Y-X = Y          */
    Écrire(x, y) ;        /* Affiche Y et X              */
fin
```



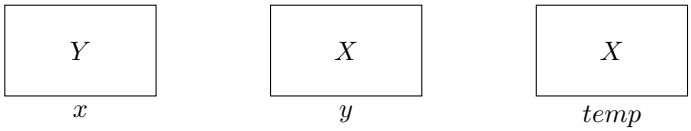
Cet algorithme devrait donc se nommer **permuter**, c'est à dire **inverser les contenus**.

Exo 2 - déroulé complet du deuxième algorithme

Exercice 2 : **Proposer (Écrire) un autre algorithme pour faire la même chose.**
Il existe un autre algorithme en 3 temps qui nécessite une **variable temporaire supplémentaire**.

```
Programme exo2
    VAR x, y, temp : Réel ;
début
    Lire(x) ; Lire(y) ;   /* x réf X et y réf Y     */
    Écrire(x, y) ;        /* Affiche X et Y         */
    temp ← x ;            /* 1 - temp réf X         */
    x ← y;                /* 2 - x réf Y            */
    y ← temp;             /* 3 - y réf X            */
    Écrire(x, y) ;        /* Affiche Y et X         */
```

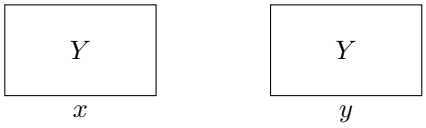
```
10 fin
```



Exo 2 - remarques

Notez bien que l'**algorithme suivant est FAUX** : sans la variable temp, on ferait disparaître la valeur X dès la ligne 7.

```
1  Programme exo2faux
2      VAR x, y       : Réel ;
3  début
4      Lire(x) ; Lire(y) ;   /* x réf X et y réf Y     */
5      Écrire(x, y) ;        /* Affiche X et Y         */
6
7      x ← y;                /* x réf Y                */
8      y ← x;                /* y réf Y                */
9      Écrire(x, y) ;        /* Affiche Y et Y         */
10 fin
```



Exo 2 - remarques

Par contre, les rôles de x et y sont interchangeables : on peut récupérer soit la valeur de x dans temp, soit la valeur de y dans temp. Il suffit de bien modifier la suite du programme. Par exemple :

```
1  Programme exo2
2      VAR x, y, temp : Réel ;
3  début
4      Lire(x) ; Lire(y) ;   /* x réf X et y réf Y     */
5      Écrire(x, y) ;        /* Affiche X et Y         */
6      temp ← y ;            /* 1 - temp réf Y         */
7      y ← x;                /* 2 - y réf X            */
8      x ← temp;             /* 3 - x réf Y            */
9      Écrire(x, y) ;        /* Affiche Y et X         */
10 fin
```

Exo 2 - version 1 en "algo-Python"

```
1  # programme principal version 1
2  x:int ;
3  y:int ;
4  temp:int ;
5  print("Entrer un entier") ; x = int(input()) ;
6  print("Entrer un entier") ; y = int(input()) ;
7  print(x, " ", y) ;
8
9  temp = x ;
10 x = y ;
11 y = temp ;
12 print(x, " ", y) ;
```

Exo 2 - version 2 en "algo-Python"

```
1  # programme principal version 1
2  print("Entrer un entier") ; x:int = int(input()) ;
3  print("Entrer un entier") ; y:int = int(input()) ;
4  print(x, " ", y) ;
```

```

temp:int = x ;
x = y ;
y = temp ;
print(x, " ", y) ;
```

Exo 2 - version 3 en Python

```
# programme principal version 2
print("Entrer un entier") ; x = int(input())
print("Entrer un entier") ; y = int(input())
print(x, " ", y)

temp = x
x = y
y = temp
print(x, " ", y)
```

En Python, le saut à la ligne indique qu'on change d'instructions.

Le point-virgule n'est donc obligatoire en Python qu'aux lignes 2 et 3. Mais on préférera les mettre en TP pour ne pas les oublier le jour du DS papier.

Exo 2 - version 3 en Python

```
# programme principal version 2
x = int(input("Entrer un entier"))
y = int(input("Entrer un entier"))
print(x, " ", y)

temp = x
x = y
y = temp
print(x, " ", y)
```

En Python, on peut transmettre le texte à afficher pour la question directement dans les parenthèses de input().

D'ailleurs, il faudra plutôt faire cela si vous utilisez en TP une version de Python qui tourne sur un navigateur Web.

Exo 2 - version 4 en Python

```
# programme principal version 2
x = int(input("Entrer un entier"))
y = int(input("Entrer un entier"))
print(x, " ", y)

x, y = y, x
print(x, " ", y)
```

La permutation est une opération très courante en informatique. Python permet de le faire assez simplement à l'aide de l'utilisation des tuples mais **cette utilisation n'est pas autorisée sur ce module d'algorithmique car tous les langages ne comportent pas cette syntaxe.**

En DS, une telle réponse ne serait pas acceptée en tant que réponse algorithmique valide. Attention, si vous avez l'habitude de faire cela.

Exo 3

Exercice 3 :

Écrire un algorithme qui calcule et affiche la moyenne de deux notes (lues) d'un étudiant, chaque note ayant un coefficient (lu également).

```
Programme exo3
    VAR n1, n2, c1, c2, moyenne : Réel ;
début
    Écrire("Donnez la note 1 puis son coefficient.") ;
```

```
    Lire(n1) ; Lire(c1) ;
    Écrire("Donnez la note 2 puis son coefficient.") ;
    Lire(n2) ; Lire(c2) ;
    moyenne ← (n1×c1 + n2×c2) / (c1 + c2) ;
    Écrire(moyenne) ;
fin
```

Exo 3

Gestion visuelle des priorités

Normalement, on espace les opérateurs et les opérandes.
On peut tenter d'aider le lecteur humain en supprimant ces espaces autour de l'opération à faire en priorité :

Correct sans aide visuelle

```
moyenne ← (n1 × c1 + n2 × c2) / (c1 + c2) ;
```

Correct avec aide visuelle sur les priorités

```
moyenne ← (n1×c1 + n2×c2) / (c1 + c2) ;
```

Nommez correctement vos variables !

Attention, aux noms de variables.
$note1$ et $coef1$ seraient beaucoup plus **explicites** que $n1$ et $c1$.
Par la suite, on utilisera donc plutôt des noms permettant de comprendre ce que les variables sont censées référencer.

Exo 4

Exercice 4 :
On considère la fonction mathématique f définie par l'expression :

$$f(x) = (2 - x) \times (-4x^2 - 1)$$

Écrire un algorithme qui **calcule** et **affiche** la valeur de l'image d'un nombre x, lu au clavier, par f (attention au calcul de x^2 en algorithmique).

Exo 4

Exercice 4 :
Le type de s est ici à choisir car non spécifié directement ou indirectement dans l'énoncé. Prenons des entiers par exemple.

$$f(x) = (2 - x) \times (-4x^2 - 1)$$

```
Programme exo4
    VAR x, image : Entier ;
début
    Écrire("Donnez la valeur voulue pour x") ;
    Lire(x) ;
    image ← (2 - x) × (-4 × x × x - 1 ) ;
    Écrire(image) ;
fin
```

Complément : division euclidienne

```
  23 | 3
     |---
 -21 | 7 -> QUOTIENT
 = 2
RESTE
```

Quotient et reste en pseudo-code : DIV et MOD
On obtient ces résultats avec les opérateurs de pseudo-code suivants :

- a DIV b pour obtenir le quotient de a / b
- a MOD b pour obtenir le reste de a / b

```
q ← 23 DIV 3 ;   q référence 7
r ← 23 MOD 3 ;   r référence 2
```

Exo 5

Exercice 5 :

Que fait l'algorithme suivant :

```
1  Programme exo5
2  VAR n, r : Entier ;
3  début
4      n ← 19 ;
5      r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
6      r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
7      r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
8      r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
9      Écrire(n) ;
10 fin
```

Exo 5

```
4      n ← 19 ;
5      r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
6      r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
7      r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
8      r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
9      Écrire(n) ;
```

L4 **n** réf. **19**

L5 **r réf. 1** car 19 MOD 2 = 1 ; AFFICHE ① ; **n réf. 9** car 19 DIV 2 = 9

L6 **r réf. 1** car 9 MOD 2 = 1 ; AFFICHE **1** ; **n réf. 4** car 9 DIV 2 = 4

L7 **r réf. 0** car 4 MOD 2 = 0 ; AFFICHE **0** ; **n réf. 4** car 4 DIV 2 = 2

L8 **r réf. 0** car 2 MOD 2 = 0 ; AFFICHE **0** ; **n réf. 1** car 2 DIV 2 = 1

L9 AFFICHE **1**

Exo 5

```
4      n ← 19 ;
5      r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
6      r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
7      r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
8      r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
9      Écrire(n) ;
```

Observations de ce qu'on réalise 4 fois à l'identique :

- on affiche le reste de la division de n par 2
- on modifie n en le divisant par 2

Remarquons bien que pour modifier une variable par rapport à son ancienne valeur, il faut placer le nom de la variable à droite, réaliser un calcul et écraser l'ancienne valeur en replaçant le nom de la variable à gauche.

Exo 5

```
4      n ← 19 ;
5      r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
6      r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
7      r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
8      r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
9      Écrire(n) ;
```

On obtient la décomposition binaire de 19 :

- ① pèse $1 = 2^0$(bit de poids faible, ligne 5)
- **1** pèse $2 = 2^1$
- **0** pèse $4 = 2^2$
- **0** pèse $8 = 2^3$
- **1** pèse $16 = 2^4$ (bit de **poids fort**, ligne 9)

Somme des poids des bits non nuls : $\mathbf{16} + \mathbf{0} + \mathbf{0} + \mathbf{2} + \mathbf{1} = 19$
En plaçant le bit de **poids fort** à gauche : **1**0**0**1**1** est l'écriture de 19 en base 2.

Exo 5

CONCLUSION

Cet algorithme donne la décomposition binaire des entiers positifs nécessitant au maximum 5 bits (n de 0 à 31)

```
          16  8  4  2  1
n = 0      0  0  0  0  0
n = 1      0  0  0  0  1
n = 2      0  0  0  1  0
n = 3      0  0  0  1  1
n = 4      0  0  1  0  0
n = 5      0  0  1  0  1
 ...
n = 15     0  1  1  1  1
n = 16     1  0  0  0  0
n = 17     1  0  0  0  1
 ...
n = 29     1  1  1  0  1
n = 30     1  1  1  1  0
n = 31     1  1  1  1  1
```

Exo 5

Compléments
Si on veut les bits de n = 32 ou plus, il faudrait rajouter une ligne supplémentaire par bit voulu.

```
4     n ← 19 ;
5     r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
6     r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
7     r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
8     r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
9     r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;  /* nouveau */
10    Écrire(n) ;
```

Ce corps d'algorithme permet de gérer n de 0 à 63.

Exo 5

Compléments
Si on veut les bits de n = 64 ou plus, il faudrait rajouter une ligne supplémentaire par bit voulu.

```
4     n ← 19 ;
5     r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
6     r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
7     r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
8     r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;
9     r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;  /* nouveau */
10    r ← n MOD 2 ; Écrire(r) ; n ← n DIV 2 ;  /* nouveau */
11    Écrire(n) ;
```

Ce corps d'algorithme permet de gérer n de 0 à 127.
Et pour aller plus loin sans limite sur n ?

5 Bilan

Résumé des connaissances à avoir

Via le TD :

- Séquentialité d'un algorithme ;
- 3 primitives : affectation, Lire(a), Ecrire(a) ;
- 2 objets : variables et CONSTANTES ;
- DIV et MOD pour obtenir quotient et reste ;
- Les types usuels (entiers, réels, caractères, booléens) ;
- ALGO : Connaître les deux algorithmes de permutation de contenus ;
- ALGO : Connaître l'algorithme de décomposition binaire (division par deux) ;

Et via Moodle : la vidéo et les exos supplémentaires : notion d'interface ;

Complément CONSTANTE

```
CONST PI = 3.1415 ;
Programme perimetreCercle
  VAR rayon, p : Réel ;
début
  Lire(rayon) ;
  p ← 2 × PI × rayon ;
  Écrire(rayon, p) ;
fin
```

Pour la semaine prochaine

Avec Moodle :

1. Récupérer le PDF du TD : **paf.infoforall.fr/algo** ;
2. S'inscrire sur Moodle :
3. **Faire les exos du support du cours 1** ;
4. Lire et réaliser les deux parties proposées sur Moodle (vidéos, petites questions...) ;
5. Réaliser **sérieusement** le QCM final de la leçon 1 : fait partie du CC