

INTRODUCTION

1) Terminologie

1.1) L'étude algorithmique

L'étude algorithmique est un ensemble de réflexions sur la production de règles, de conditions et d'actions devant être appliqués dans la définition et la conception d'une suite logique d'instructions. Une fois posé, ceci formera alors : « un algorithme ».

1.2) Un algorithme

Un algorithme est une suite logique de règles, pouvant être textuelle (écrit sur un document avec un ensemble de mots spécifiques), ou graphique (formes géométriques spécifiques à l'action ou la règle à décrire).

1.2.a) Algorithme textuel

Un algorithme textuel est écrit en utilisant un ensemble de mots-clés, spécifiques à la langue parlée par la personne écrivant l'algorithme. Pour une large diffusion, l'utilisation de l'anglais est préférée.

Dans nos exemples, les mots-clés seront en français.

Note: Cet ensemble de mots-clés et instructions est appelé : « pseudo-code ».

1.2.b) Algorithme graphique

Un algorithme graphique, appelé aussi « algorithme » ou « logigramme », est décrit en utilisant un ensemble de symboles, ces symboles sont définis par la norme ISO 5807.

certaines de ces symboles seront présentés dans ce document.

INTRODUCTION

2) L'utilisation d'un algorithme

2.1) Pourquoi faire des algorithmes ?

L'intérêt de faire des algorithmes est de pouvoir décrire sous forme d'une suite logique, un ensemble d'instructions, une procédure ou toutes actions nécessitant d'être détaillées dans une forme structurée et séquentielle.

2.2) Quand utiliser des algorithmes ?

Un algorithme doit être utilisé pour la description d'un comportement logiciel, de tout système logique, d'une fonction mathématique, etc. Le développement logiciel est un domaine où les algorithmes sont omniprésents.

INTRODUCTION

2.3) Comment écrire un algorithme ?

2.3.a) Utilisation de l'ISO 5807

Pour l'écriture d'algorithme, l'utilisation de l'ISO 5807 est préférée. Voici un exemple d'un algorithme :

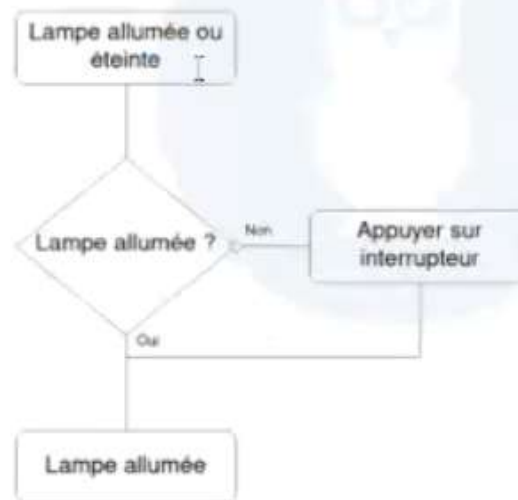


FIGURE 1 – Algorithme idempotent pour allumer une lampe.

INTRODUCTION

2.3.b) Le pseudo-code

Pour l'écriture textuelle d'algorithme, le pseudo-code en anglais est préféré, mais dans ce document nous utiliserons le français. Voici un exemple d'un algorithme écrit :

```
1 // Définition d'un algorithme nommé "mon_algo".
2 // Cet algorithme est un exemple simple sur l'écrit d'une structure simplifiée,
3 // l'utilisation d'une variable et l'appel d'une fonction.
4 ALGORITHME mon_algo
5 VARIABLES
6 // Déclaration d'une variable de type entier.
7 entier: a
8 DEBUT
9     // Affectation de la valeur 42 à la variable "a".
10    a ← 42
11
12    // Appel de la fonction "AFFICHER()" afin d'écrire à l'écran la valeur de a.
13    AFFICHER ("La variable vaut :", a)
14 FIN
```

STRUCTURE D'UN ALGORITHME

La structure d'un algorithme est précise et particulière, certaines règles doivent être respectées et cette structure peut légèrement varier suivant le type d'algorithme décrit.

3) Les types d'algorithmes

Il existe plusieurs types d'algorithmes, mais chacun d'entre eux a un rôle défini. Lors de l'écriture d'un algorithme en pseudo-code, le type choisi est le tout premier mot présenté dans ce pseudo-code.

3.1) Un « ALGORITHME »

Le type ALGORITHME est utilisé afin de décrire un ensemble d'instructions ayant un départ et une arrivée, ainsi que des appels éventuels à des PROCEDURE et des FONCTION. Généralement, ALGORITHME sert à écrire la routine principale (appelée « main ») d'un programme. Un algorithme est décrit avec le mot-clé : ALGORITHME.

3.2) Une « PROCEDURE »

Le type PROCEDURE est identique à ALGORITHME, sauf que PROCEDURE est en capacité de prendre en charge des « paramètres » lors de son invocation (appel de la procédure), et contrairement à ALGORITHME, PROCEDURE n'est pas utilisé pour écrire la routine principale (« main »), mais des « sous-routines » (fonctions indépendantes). Une procédure est décrite avec le mot-clé : PROCEDURE.

Note : Une PROCEDURE ne retourne pas de résultat, elle effectue des opérations mais ne renvoie aucune valeur à l'appelant.

3.3) Une « FONCTION »

Le type FONCTION est absolument identique à PROCEDURE, sauf que FONCTION retourne un résultat en utilisant le mot-clé : RETOURNE, et le type du résultat retourné est déclaré en fin de signature (détailé dans une section ultérieure). Une fonction est décrite avec le mot-clé : FONCTION.

STRUCTURE D'UN ALGORITHME

4) Particularités de la structure

L'écriture d'algorithme doit respecter certaines règles, telles que :

- l'ordre : d'écriture des « sections »
- le formatage : manière d'écrire un mot-clé suivant sa nature :
 - les mots-clés liés au langage (ALGORITHME, DEBUT, FIN, etc)
 - les types de variables et constantes (entier, reel, booleen, etc)
 - le nom des variables, constantes et des routines (mon_algo, ma_var, ma_fonc, etc)
 - les commentaires, utilisés pour décrire vos instructions de façon plus « humaine »
 - Etc.
- les mots-clés : les instructions sont associées à des mots-clés spécifiques
- les déclarations : de variables, de constantes, de tableaux, etc.
- les opérateurs : suivant l'opération à effectuer, l'opérateur associé doit être écrit d'une certaine façon.

Nous allons donc les détailler.

STRUCTURE D'UN ALGORITHME

4.1) Formatage des éléments

Le formatage des éléments suit une règle liée à l'élément en question.

4.1.a) Formatage des mots-clés

Un mot-clé doit toujours être écrit en majuscule et en gras, exemple :

- **ALGORITHME**: définition de l'algorithme (routine) principal
- **PROCEDURE**: définition d'une sous-routine appelée « procédure »
- **FONCTION**: définition d'une sous-routine appelée « fonction »
- **DEBUT**: définition du point de départ d'une routine et sous-routine
- **FIN**: définition du point d'arrivée d'une routine et sous-routine

la liste des mots-clés est décrite dans la section : Les mots-clés.

STRUCTURE D'UN ALGORITHME

4.1.b) Formatage des identifiants

Un identifiant est le nom d'une déclaration et est écrit en italique.

Un identifiant doit respecter des règles syntaxiques :

- composé de caractères alphanumériques, sans accent: *a..z, A..Z et 0..9*
- composé du caractère « souligné » (appelé « underscore ») : *_*
- ne peut commencer par un chiffre : *42_mon_entier*
- avant d'utiliser un identifiant, celui-ci doit être déclaré antérieurement
- le nom d'un identifiant doit être différent d'un mot-clé ou d'une instruction définie : *procedure, algorithme*
- si l'identifiant est une constante, alors l'écrire en toute majuscule

Un identifiant peut être :

- le nom d'une routine ou sous-routine : *mon_algorithme, calculer_facto, compter_nb_caractère*
- le nom d'une variable : *mon_age, mon_prenom, mon_adresse*
- le nom d'une constante : *MA_CONSTANTE, AGE_PAR_DEFAULT*
- le type d'une variable ou constante : *entier, booleen, reel*

STRUCTURE D'UN ALGORITHME

4.1.c) Formatage des éléments à détailler

Les éléments à « détailler » sont écrits entre crochets :<éléments>.

Les éléments à détailler sont des « zones » où la définition n'a pas encore été effectuée, car l'algorithme est en cours de développement ou de réflexion.

4.1.d) Formatage des éléments optionnels

Les éléments « optimisation » sont écrits entre accolades :{éléments}.

Les éléments optionnels sont des « zones » où la définition n'est pas obligatoire, car l'algorithme fonctionne sans ces éléments, du moins pour l'instant présent, il se peut que dans l'évolution de l'algorithme cette définition soit alors écrite.

4.1.e) Formatage des zones omises

Lors de l'écriture d'algorithme, il se peut que nous souhaitons omettre quelques instructions car ce n'est pas ce que nous voulons mettre en avant, et pour ne pas perdre de temps afin de passer à l'essentiel, il faudra ajouter trois petits points (. . .) là où nous souhaitons définir cette « zone d'omissions ».

STRUCTURE D'UN ALGORITHME

4.1.f) Formatage de la déclaration d'une variable ou constante

Les déclarations des variables et constantes, se font sous la forme suivante :

```
1 // Format de déclaration simple.
2 type_donnee: nom_variable
3 // Format de déclaration multiple.
4 type_donnee: nom_variable1, nom_variable2, nom_variableN
5
6 // Déclaration d'une variable "toto" de type "chaîne", sans valeur.
7 chaîne: toto
8 // Déclaration d'une variable "toto" de type "chaîne", avec valeur.
9 chaîne: toto="Hello Word!"
10 // Déclaration de trois variables de type "chaîne", avec ou sans valeur.
11 chaîne: c1="Hello", c2, c3="Word!"
12
13 // Déclaration d'une variable "t1" de type "tableau d'entier" à 1 dimension,
14 // avec l'indice minimum "entierA" à l'indice maximum "entierB".
15 // Notez que les paramètres du tableau sont séparés par une virgule (",") pour une dimension
16 // donnée, et par un point-virgule (";") entre les dimensions.
17 entier: t1[entierA, entierB]
18
19 // Déclaration d'une variable "t2" de type "tableau d'entier" à 2 dimensions,
20 // avec l'indice minimum "entier1A" à l'indice maximum "entier1B" pour la première dimension,
21 // puis avec l'indice minimum "entier2A" à l'indice maximum "entier2B" pour la
22 // deuxième dimension.
23 entier: t2[entier1A,entier1B;entier2A,entier2B]
24
25 // La récupération d'une valeur dans une dimension ou sous-dimension se fait comme ceci:
26 valeur ← t1[3] // Récupération à l'indice 3 de la première dimension.
27 valeur ← t2[3][1] // Récupération à l'indice 3 de la première dimension, et 1 pour la deuxième.
```

STRUCTURE D'UN ALGORITHME

4.1.g) Exemples de formatages

```
1 // Définition d'un algorithme nommé "mon_algo".
2 // Cet algorithme est un exemple simple sur l'écrit d'une structure simplifiée,
3 // l'utilisation d'une variable et l'appel d'une fonction.
4 ALGORITHME mon_algo
5 VARIABLES
6     // Déclaration d'une variable de type entier.
7     entier: a
8
9     // Déclaration des variables de type chaîne.
10    chaîne: <les chaînes doivent être décrites>
11
12    <la partie déclarations doit être décrite>
13 CONSTANTES
14    (les constantes peuvent être décrites)
15 DEBUT
16    <la partie instructions doit être décrite>
17    ...
18 FIN
```

Dans l'exemple ci-dessus, nous pouvons voir que la routine principale «mon_algo» décrit :

- une variable «a» de type «entier» (ligne 7)
- des variables de type «chaîne» n'étant pas encore définies car cet algorithme n'est pas encore finalisé (ligne 10)
- que la partie « déclarations » n'est pas encore terminée (ligne 12)
- que la partie «CONSTANTES» est optionnelle à cet instant (ligne 14)
- que la partie « instructions » n'est pas encore définie (ligne 16)
- que la partie « instructions » est omise volontairement (ligne 17)

STRUCTURE D'UN ALGORITHME

4.2) Les mots-clés

Un mot-clé est soit une déclaration de section soit une instruction. Écrit en majuscule et en gras : **KEYWORD**.

Voici un tableau des mots-clés de déclaration :

ALGORITHME	Déclaration d'une routine principale
PROCEDURE	Déclaration d'une sous-routine de type procédure
FONCTION	Déclaration d'une sous-routine de type fonction
PARAMETRES	Déclaration d'une section où sont déclarés les paramètres
VARIABLES	Déclaration d'une section où sont déclarées les variables
CONSTANTES	Déclaration d'une section où sont déclarées les constantes
DEBUT	Déclaration du début d'une routine
RETOURNE	Terminaison d'une sous-routine FONCTION en retournant un résultat
FIN	Déclaration de fin d'une routine

STRUCTURE D'UN ALGORITHME

4.2) Les mots-clés

Voici un tableau des mots-clés des structures de prise de décision :

SI	Déclaration d'un début de prise de décision suivant une condition
ALORS	Déclaration de la décision prise
SINON SI	Déclaration d'une prise de décision alternative suivant une autre condition
SINON	Déclaration d'une prise de décision alternative si aucune condition antérieure n'est valide
FIN SI	Déclaration d'une fin de prise de décision
SELON	Déclaration d'une prise de décision à choix multiple
AUTREMENT	Déclaration de l'action par défaut si aucune valeur n'est valide dans un SELON
FIN SELON	Déclaration d'une fin de prise de décision à choix multiple

STRUCTURE D'UN ALGORITHME

4.2) Les mots-clés

Voici un tableau des mots-clés des structures de répétition d'instructions (boucles) :

POUR	Déclaration d'une boucle de type répétition séquentielle
ALLANT DE, A	Définition des conditions d'une boucle POUR
PAR PAS DE	Définition du pas d'une boucle POUR
TANT QUE	Déclaration d'une boucle de type répétition suivant une condition testée
REPETER	Déclaration d'une boucle de type répétition suivant une condition testée après le premier tour
JUSQU'A	Définition de la condition à tester, pour une boucle REPETER
FIN x	Déclaration de la fin d'une boucle de type x (POUR, TANT QUE)

STRUCTURE D'UN ALGORITHME

4.3) Les commentaires

L'écriture d'un commentaire est initialisée par un double slash («//») en début de ligne. Si votre commentaire est écrit sur 3 lignes, alors ces 3 lignes commenceront par «//».

4.3.a Exemples de commentaires

```
1 // Ceci est un commentaire, écris sur trois lignes.
2 // Ce commentaire sert d'exemple.
3 // Chacune de ces lignes commence par un double slash "//".
4 ALGORITHME mon_algo
5     // Les commentaires peuvent être placés n'importe où.
6 DEBUT
7     // Vraiment n'importe où.
8     ...
9 FIN
```



STRUCTURE D'UN ALGORITHME

5) Le découpage par section

Un algorithme est « découpé » en deux « sections », la partie supérieure appelée « Entête » et la partie inférieure appelée « Corps ». L'intérêt de découper un algorithme en deux parties, est de bien distinguer la zone « de préparation » à la zone « exécution ». Ces deux sections ont un rôle précis dans l'écriture d'algorithme.

5.1) La section « Entête »

L'entête est la partie supérieure de l'algorithme, celle-ci contient les descriptions préliminaires de l'algorithme.

L'entête est découpé en deux ou trois parties : la partie «signature», la partie «PARAMETRES» et la partie «déclarations»

Ces dernières ont un rôle précis dans la définition de l'entête.

STRUCTURE D'UN ALGORITHME

5.1.a) La signature

La signature est constituée du type de l'algorithme, du nom de la routine, du nom des paramètres locaux si l'algorithme est une sous-routine, ainsi que du type du résultat retourné si la sous-routine est une **FONCTION**.

Exemples :

```
1 // Signature d'une routine principale.
2 ALGORITHME mon_algo
3
4 // Signature d'une sous-routine de type fonction.
5 // Cette fonction retourne un résultat de type "entier".
6 FONCTION ma_fonc () : entier
7
8 // Signature d'une sous-routine de type fonction avec deux paramètres.
9 // Cette fonction retourne un résultat de type "entier".
10 FONCTION ma_fonc (param1, param2) : entier
11
12 // Signature d'une sous-routine de type procédure.
13 PROCEDURE ma_proc ()
14
15 // Signature d'une sous-routine de type procédure avec deux paramètres.
16 PROCEDURE ma_proc (param1, param2)
```

Il est important de retenir que la signature d'un **ALGORITHME** a une légère différence avec celles des **PROCEDURE** et **FONCTION**, en effet, comme **ALGORITHME** décrit une routine principale alors que **PROCEDURE** et **FONCTION** décrivent une sous-routine, et que celles-ci peuvent être invoquées, il faudra alors ajouter des parenthèses à la suite du nom de ces sous-routines, ces parenthèses contiendront la liste des paramètres locaux si la sous-routine en accepte :

```
1 // Signature d'une routine principale, sans parenthèses.
2 ALGORITHME mon_algo
3
4 // Signature d'une sous-routine de type fonction, avec parenthèses.
5 FONCTION ma_fonc () : entier
6
7 // Signature d'une sous-routine de type procédure, avec parenthèses.
8 PROCEDURE ma_proc ()
```

STRUCTURE D'UN ALGORITHME

5.1.b) Les paramètres

La partie **PARAMETRES** est optionnelle, elle n'est présente que si une sous-routine a des paramètres passés lors de son invocation, appelés « paramètres locaux », ou que celle-ci utilise des paramètres d'environnement appelés « paramètres globaux ».

- **locaux** : sont les paramètres passés lors de l'invocation de la sous-routine, ils sont déclarés dans la signature puis définis dans la section **PARAMETRES**. Les paramètres locaux sont des copies au sein de la sous-routine, c-à-d que leur valeur n'est modifiable que dans la sous-routine, ces modifications n'affecteront pas les variables initiales. Ils sont déclarés avec le mot-clé **PARAMETRES locaux**.
- **globaux** : sont les variables d'environnement, celles-ci sont déclarées au préalable mais leur utilisation est définie dans la section **PARAMETRES**. Les paramètres globaux contrairement aux locaux, ne sont pas des copies, c-à-d que s'ils viennent à être modifiés, la nouvelle valeur sera « visible » par toutes les autres routines utilisant ces paramètres. Ils sont déclarés avec le mot-clé **PARAMETRES globaux**. La déclaration des paramètres globaux est prioritaire sur celle des paramètres locaux.

Note : Les paramètres se définissent de la même façon que des VARIABLES.

Exemple :

```
1 // Sous-routine de type fonction avec deux paramètres locaux et trois paramètres
2 // globaux.
3 // Cette fonction retourne un résultat de type "entier".
4 FONCTION ma_fonc (param1, param2) : entier
5 PARAMETRES globaux
6     // Définition des paramètres globaux utilisés.
7     entier: g_param1, g_param2
8     chaine: g_c1
9 PARAMETRES locaux
10    // Définition des paramètres locaux utilisés.
11    // Ici, param2 sera égal à 42 si jamais ce paramètre a été omis durant l'invocation,
12    // ce procédé permet d'utiliser une valeur par défaut si absent.
13    entier: param1, param2=42
```


STRUCTURE D'UN ALGORITHME

5.1.c) La partie déclarations

La partie déclarations est la section qui succède la partie PARAMETRES, celle-ci est constituée de la déclaration des VARIABLES et CONSTANTES.

Exemple :

```
1 // Sous-routine de type fonction avec deux paramètres locaux et trois paramètres
2 // globaux, deux variables locales et une constante.
3 // Cette fonction retourne un résultat de type "entier".
4 FONCTION ma_fonc (param1, param2) : entier
5 PARAMETRES globaux
6     // Définition des paramètres globaux utilisés.
7     entier: g_param1, g_param2
8     chaine: g_c1
9 PARAMETRES locaux
10    // Définition des paramètres locaux utilisés.
11    entier: param1, param2=42
12 VARIABLES
13    // Définition des variables utilisées.
14    entier: var1, var2
15 CONSTANTES
16    // Définition de la constante utilisée.
17    entier: CONST1
```

STRUCTURE D'UN ALGORITHME

5.2) La section « Corps »

Le corps est la partie inférieure de l'algorithme, celle-ci contient toutes les instructions et opérations qui constituent celui-ci. Le corps est déclaré avec le mot-clé DEBUT et est clôturé par le mot-clé FIN, toutes les instructions présentes entre ces deux séquences sont alors dans une partie appelée « partie instructions ».

5.2.a) Le début et la fin

Tout corps d'algorithme a un début et une fin, que se soit pour une routine principale (ALGORITHME), ou pour une sous-routine (FONCTION, PROCEDURE). Le début est déclaré avec le mot-clé DEBUT et la fin avec FIN.

5.2.b) La partie instructions

La partie instructions est la partie la plus intéressante, celle-ci est ce qui constitue concrètement l'algorithme. La partie instructions contient des opérations arithmétiques, logiques, des affectations de valeur à des variables, l'utilisation de structures de prise de décision, de boucles, etc. Cette partie est l'élément le plus important de l'algorithme, car c'est celui-ci qui sera traduit par la suite en programme via un réel langage de programmation.

Exemple :

```
1 // Cette fonction retourne un résultat de type "booléen", indiquant
2 // si l'âge passé en paramètre est un âge majeur, donc supérieur
3 // ou égal à 18 ans.
4 FONCTION verifie_majorite(age) : booléen
5 PARAMETRES locaux
6     // Définition du paramètre local utilisé.
7     entier: age
8 DEBUT
9     // Test si l'âge est strictement inférieur à 18.
10    Si age < 18 ALORS
11        RETOURNE FAUX
12    SINON
13        RETOURNE VRAI
14    FIN SI
15 FIN
```

COMPOSANTS DE LA PARTIE INSTRUCTIONS

6) L'affectation

L'affectation de variable se fait via une flèche directionnelle vers la gauche («←»). Lorsque qu'il est nécessaire d'affecter une valeur à une variable, la syntaxe à respecter est la suivante :

```
1 // La variable appelée "var" est affectée de la valeur "val".
2 var←val
```

Il faudra noter que seule une valeur d'un type spécifiques est affectable à une variable du même type:

```
1 // La variable "var" de type entier est affectée de 42.
2 entier var←42
```

Exemple :

```
1 // Cet algorithme présente des affectations de valeur à des variables.
2 ALGORITHME mon_algo
3 VARIABLES
4     // Déclaration de variables de type entier.
5     entier: a, b
6
7     // Déclaration d'une variable de type chaîne.
8     chaîne: c1
9 DEBUT
10    // Affecte 42 à a.
11    a←42
12
13    // Affecte 24 à b.
14    b←24
15
16    // Affecte une chaîne de caractères à c1.
17    c1←"hello"
18
19    // Affiche la valeur de c1.
20    AFFICHER (c1) // Affichera : "hello".
21
22    // Affecte la valeur retournée par une fonction
23    // appelée majuscule () à la variable c1.
24    c1←majuscule (c1)
25
26    // Affiche la valeur de c1.
27    AFFICHER (c1) // Affichera : "HELLO".
28 FIN
```



COMPOSANTS DE LA PARTIE INSTRUCTIONS

7) Les opérateurs

Les opérateurs sont les « symboles » utilisés pour effectuer des actions entre différentes valeurs. Il existe différents types d'opérateur, les classiques (arithmétiques), ceux utilisés pour des opérations booléennes (logiques) et ceux utilisés pour tester une relation booléenne entre deux valeurs (relationnels). Il existe aussi un opérateur servant à concaténer plusieurs segments de chaîne de caractères.

7.1) Opérateurs relationnels

Les opérateurs relationnels servent à effectuer des comparaisons binaire et retourne un résultat booléen. Les opérandes (valeurs comparées) doivent être du même type : entier, caractère, etc.

7.1.a) Liste des opérateurs relationnels

=	Test si les deux opérandes sont égaux
<	Test si les deux opérandes sont différents
>	
<	Test si l'opérande de gauche est strictement inférieur à celui de droite
>	Test si l'opérande de gauche est strictement supérieur à celui de droite
<	Test si l'opérande de gauche est inférieur ou égal à celui de droite
=	

COMPOSANTS DE LA PARTIE INSTRUCTIONS

7.1.b) Exemples d'opérations relationnelles

```
1 // Cet algorithme effectue diverses opérations relationnelles, afin de montrer
2 // des exemples d'utilisation.
3 ALGORITHME exemples_relationnels
4 DEBUT
5     // Test d'une relation "strictement inférieur à".
6     SI 1 < 2 ALORS
7         AFFICHER ("1 est strictement inférieur à 2")
8     FIN SI
9
10    // Test d'une relation "strictement supérieur à".
11    SI 2 > 1 ALORS
12        AFFICHER ("2 est strictement supérieur à 1")
13    FIN SI
14
15    // Test d'une relation "inférieur ou égal à".
16    SI 1 <= 2 ALORS
17        AFFICHER ("1 est inférieur ou égal à 2")
18    FIN SI
19
20    // Test d'une relation "est égal à".
21    SI VRAI = VRAI ALORS
22        AFFICHER ("La valeur booléenne VRAI est égale à VRAI")
23    FIN SI
24
25    // Test d'une relation "est différent de".
26    SI "Hello" <> "World!" ALORS
27        AFFICHER ("Le mot 'Hello' est différent du mot 'Word!'")
28    FIN SI
29 FIN
```

Ces opérateurs sont très utilisés dans des structures de prise de décision.

COMPOSANTS DE LA PARTIE INSTRUCTIONS

7.2) Opérateurs arithmétiques

Les opérateurs arithmétiques servent à effectuer des opérations mathématiques simples, et retournent un résultat en fonction de l'opérateur utilisé. Les opérandes (valeurs à calculer) doivent être d'un type numérique : entier ou réel.

7.2.a) Liste des opérateurs arithmétiques

- (préfixe)	Inverse le signe de l'opérande situé directement à droite
+	Effectue une addition
-	Effectue une soustraction
*	Effectue une multiplication
/	Effectue une division

COMPOSANTS DE LA PARTIE INSTRUCTIONS

7.2.b Exemples d'opérations arithmétiques

```
1 // Cet algorithme effectue diverses opérations arithmétiques, afin de montrer des exemples d'utilisation.
2
3 ALGORITHME exemples_arithmetiques
4 VARIABLES
5   entier: a, b
6 DEBUT
7   // Affecte une valeur à a et b.
8   a ← 1
9   b ← 2
10
11   // Inverse le signe de a.
12   a ← -a
13
14   // Additionne b + -a
15   b ← b + a
16
17   // Multiplie b par 2 fois -a.
18   b ← b * (2 * -a)
19
20   // Divise b par 3
21   b ← b / 3
22
23   // Soustrait 1 à b.
24   b ← b - 1
25 FIN
```



COMPOSANTS DE LA PARTIE INSTRUCTIONS

7.3) Opérateurs logiques

Les opérateurs logiques servent à effectuer des opérations booléennes entre deux valeurs, et retourne un résultat booléen. Les opérandes doivent être de type booléen. Écrit en majuscule et en gras : **KEYWORD**.

7.3.a) Liste des opérateurs logiques

NON	Négation logique, utilisé pour inverser une valeur booléenne
ET	ET logique, utilisé pour former une « intersection »
OU	Ou logique, utilisé pour former une « union »
OUEX	Ou exclusif, utilisé pour de l'inversement binaire en général

COMPOSANTS DE LA PARTIE INSTRUCTIONS

7.3.b) Tableaux d'opérations binaires

Voici la liste des opérations logiques avec les opérateurs : ET, OU et OUEX. La valeur décimale «0» représente la valeur booléenne «FAUX» et la valeur décimale «1» représente la valeur booléenne «VRAI».

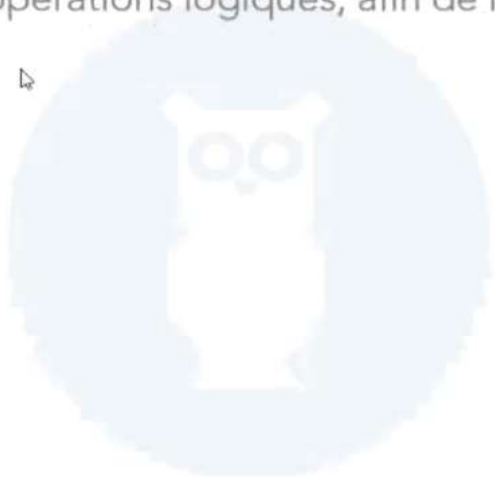
Opérateur « ET »			Opérateur « OU »			Opérateur « OUEX »		
a	b	Resultat	a	b	resultat	a	b	resultat
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0



COMPOSANTS DE LA PARTIE INSTRUCTIONS

7.3.c) Exemples d'opérations logiques

```
1 // Cet algorithme effectue diverses opérations logiques, afin de montrer des exemples d'utilisation.
2
3 ALGORITHME exemples_logiques
4 VARIABLES
5     booleen: a, b
6 DEBUT
7     // Affecte une valeur à a et b.
8     a ← VRAI
9     b ← FAUX
10
11     // Inverse la valeur de a.
12     a ← NON a // a est FAUX
13
14     // Intersection entre a et b.
15     a ← a ET b // a est FAUX
16
17     // Union entre a et NON b.
18     a ← a OU NON b // a est VRAI
19
20     // Ou exclusif entre a et b
21     a ← a OUEX b // a est VRAI
22 FIN
```



COMPOSANTS DE LA PARTIE INSTRUCTIONS

7.4) Opérateur de concaténation

Lorsqu'on souhaite concaténer plusieurs séquences de chaînes de caractères (plusieurs mots), il est alors nécessaire d'utiliser l'opérateur de concaténation, celui-ci est un plus (« + ») à placer entre deux séquences.

7.4.a) Exemples de concaténation

```
1 // Cet algorithme effectue des concaténations de chaînes de caractères.
2 ALGORITHME exemples_concatenation
3 VARIABLES
4     chaine: c1, c2
5 DEBUT
6     // Affecte une valeur à c1 et c2.
7     c1 ← "Hello"
8     c2 ← "World!"
9
10    // Affiche le résultat de la concaténation.
11    AFFICHER(c1 + " " + c2) // Affiche: "Hello World!"
12
13    // Affiche le résultat de la concaténation.
14    AFFICHER(c1 + " Jack!") // Affiche: "Hello Jack!"
15
16    // Pour concaténer une valeur numérique avec une chaîne de caractères, il faudra
17    // passer par une fonction de conversion.
18    // Ici, la fonction chaine() converti l'entier passé en argument et retourne
19    // sa représentation en chaîne de caractères.
20    AFFICHER("41 + 2 = " + chaine(42))
21
22    // La fonction AFFICHER() prend plusieurs paramètres, de n'importe quel
23    // type, cela est pratique lorsque nous souhaitons afficher différentes
24    // variables.
25    // Ici, le même exemple que le précédent mais sans conversion, la fonction
26    // AFFICHER() se charge de convertir les paramètres en chaîne de caractères
27    // automatiquement, il suffit de les séparer par une virgule.
28    AFFICHER("41 + 2 = ", 42)
29
30 FIN
```



COMPOSANTS DE LA PARTIE INSTRUCTIONS

8) L'appel de procédures et de fonctions

L'appel d'une procédure ou d'une fonction se fait en indiquant le nom de la sous-routine, suivi des parenthèses incluant ou non des arguments.

8.1) Quelques notes

- L'appel d'une procédure ou d'une fonction se fait à l'intérieur de la partie instructions d'une routine (entre DEBUT et FIN)
- L'appel d'une fonction peut être précédée d'une affectation à une variable, afin que la variable ait comme valeur : le résultat de la fonction (retourné avec le mot clé RETOURNE)
- Tout comme une affectation, l'appel d'une fonction peut aussi être à l'intérieur d'un appel à une autre fonction ou procédure (entre les parenthèses), afin que le résultat retourné soit pris en compte comme un argument de cet appel.
- Une fonction ou une procédure ne peut pas être affectée d'une valeur, car elles sont « immuables »

COMPOSANTS DE LA PARTIE INSTRUCTIONS

8.2) Exemples d'appels à des sous-routines

```
1 // Cet algorithme effectue des appels à des sous-routines.
2 ALGORITHME exemples_appels
3 VARIABLES
4     chaine: c1, c2
5 DEBUT
6     // Affecte une valeur à c1 et c2.
7     c1 ← "Hello"
8     c2 ← "Word!"
9
10    // Appel la procédure AFFICHER() avec 1 argument.
11    AFFICHER(c1) // Affiche: "Hello"
12
13    // Appel la procédure AFFICHER() avec 2 arguments.
14    AFFICHER(c1, c2) // Affiche: "Hello World!"
15
16    // Appel à une fonction appelée minuscule() avec 1 argument, puis affectation du résultat
17    // dans c1.
18    c1 ← minuscule(c1)
19
20    // Appel à une fonction appelée majuscule() avec 1 argument, puis affectation du résultat
21    // dans c2.
22    c2 ← majuscule(c2)
23
24    // Appel la procédure AFFICHER() avec 2 arguments.
25    AFFICHER(c1, c2) // Affiche: "hello WORLD!"
26
27    // Appel la procédure AFFICHER() avec 2 arguments dont un étant une fonction.
28    AFFICHER(majuscule(c1), c2) // Affiche: "HELLO WORLD!"
29
30    // Ceci est impossible, les sous-routines sont immuables.
31    minuscule ← "je casse la fonction" // Interdit et ne fonctionne pas.
32 FIN
```

COMPOSANTS DE LA PARTIE INSTRUCTIONS

8.2) Exemples d'appels à des sous-routines

```
1 // Cet algorithme effectue des appels à des sous-routines.
2 ALGORITHME exemples_appels
3 VARIABLES
4   chaine: c1, c2
5 DEBUT
6   // Affecte une valeur à c1 et c2.
7   c1 ← "Hello"
8   c2 ← "Word!"
9
10  // Appel la procédure AFFICHER() avec 1 argument.
11  AFFICHER (c1) // Affiche: "Hello"
12
13  // Appel la procédure AFFICHER() avec 2 arguments.
14  AFFICHER (c1, c2) // Affiche: "Hello World!"
15
16  // Appel à une fonction appelée minuscule() avec 1 argument, puis affectation du résultat
17  // dans c1.
18  c1 ← minuscule (c1)
19
20  // Appel à une fonction appelée majuscule() avec 1 argument, puis affectation du résultat
21  // dans c2.
22  c2 ← majuscule (c2)
23
24  // Appel la procédure AFFICHER() avec 2 arguments.
25  AFFICHER (c1, c2) // Affiche: "hello WORLD!"
26
27  // Appel la procédure AFFICHER() avec 2 arguments dont un étant une fonction.
28  AFFICHER (majuscule(c1), c2) // Affiche: "HELLO WORLD!"
29
30  // Ceci est impossible, les sous-routines sont immuables.
31  minuscule ← "je casse la fonction" // Interdit et ne fonctionne pas.
32 FIN
```