

Logique de programmation

Notions théoriques et exemples

Yannick Boogaerts

18 mars 2019

1.	Introduction Algorithmique et programmation	3
2.	Machine Logique	9
3.	Gestion des variables	13
4.	Affichage et saisie.....	19
5.	Type de données booléennes	23
6.	Structure de contrôle : l'alternative	27
7.	Structure de contrôles : les Boucles	33
8.	Type de données caractère texte.....	41
9.	Structure de données : tableau	43
10.	Sous programmes : procédures et fonctions	47
11.	Structure de données : Structure	53

Logique de programmation

1. Introduction Algorithmique et programmation

Algorithmique.....	4
Définitions	4
Exemples	4
Qualités d'un algorithme.....	5
La Programmation.....	5
Langage de programmation	6
Compilateur et interprète	6

Algorithmique

Définitions

L'algorithmique est la science des algorithmes.

Un algorithme est une suite ordonnée d'instructions qui indique la démarche à suivre pour résoudre une série de problèmes équivalents. Un algorithme ne doit contenir que des instructions compréhensibles par celui qui devra l'exécuter

L'algorithmique, vous la pratiquez tous les jours et depuis longtemps

Données	Algorithmes	Résultats
Briques de LEGO	suite de dessins	Camion de pompiers
Meuble en kit	notice de montage	Cuisine équipée
Cafetière	instructions	Expresso
Laine	modèle	Pull irlandais
Farine, œufs, chocolat, etc....	recette	Forêt noire

Exemples

Extrait d'un dialogue entre un touriste égaré et un autochtone :

- « Pourriez-vous m'indiquer le chemin de la gare, s'il vous plait ? »
- « Oui bien sûr : vous allez tout droit jusqu'au prochain carrefour, vous prenez à gauche au carrefour et ensuite la troisième à droite, et vous verrez la gare juste en face de vous. »
- « Merci. »

La réponse de l'autochtone est la description d'une suite ordonnée d'instructions : « allez tout droit, prenez à gauche, prenez la troisième à droite »

Celles-ci manipulent des données : « carrefour, rues »

Et permettent de réaliser la tâche désirée : « aller à la gare. »

Elles sont compréhensibles par des humains.

Qualités d'un algorithme

Validité

La validité d'un algorithme est son aptitude à réaliser exactement la tâche pour laquelle il a été conçu.

- Arrive-t-on effectivement à la gare en exécutant scrupuleusement les instructions dans l'ordre annoncé ?

Robustesse

La robustesse d'un algorithme est son aptitude à se protéger de conditions anormales d'utilisation.

- Le chemin proposé a été pensé pour un piéton, alors qu'il est possible que le « touriste égaré » soit en voiture et que la « troisième à droite » soit en sens interdit.

Réutilisabilité

La réutilisabilité d'un algorithme est son aptitude à être réutilisé pour résoudre des tâches équivalentes à celle pour laquelle il a été conçu.

- L'algorithme de recherche du chemin de la gare est-il réutilisable tel quel pour se rendre à la mairie ?
- A priori non, sauf si la mairie est juste à côté de la gare.

Complexité

La complexité d'un algorithme est le nombre d'instructions élémentaires à exécuter pour réaliser la tâche pour laquelle il a été conçu.

- Si le « touriste égaré » est un piéton, la complexité de l'algorithme de recherche de chemin peut se compter en nombre de pas pour arriver à la gare.

Efficacité

L'efficacité d'un algorithme est son aptitude à utiliser de manière optimale les ressources du matériel qui l'exécute.

- N'existerait-il pas un raccourci piétonnier pour arriver plus vite à la gare ?

La Programmation

La programmation d'un ordinateur consiste à lui fournir un algorithme en tenant compte :

- qu'il ne connaît qu'une série limitée d'instructions binaires (il ne « comprend » pas le langage humain),
- qu'il ne peut effectuer que des traitements sur des données composées de 0 et de 1.

Langage de programmation

Un langage de programmation est composé :

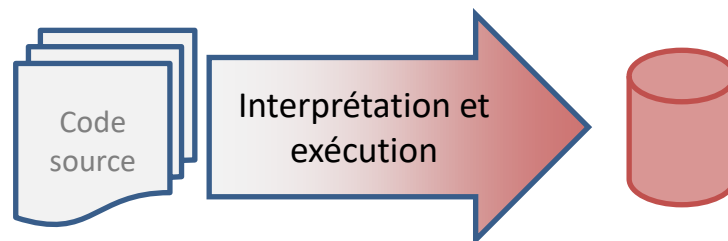
- d'un ensemble de mots-clés (choisis arbitrairement),
- de règles très précises indiquant comment on peut assembler ces mots pour former des instructions valides
- de procédures de traduction des instructions du langage en séquence d'instructions binaires reconnues par le micro-processeur.
- D'un ensemble de sous-programmes utilitaires formant l'API (Application programming interface) du langage

Ils permettent de faire abstraction des mécanismes bas niveaux de la machine. Ils facilitent la rédaction et la compréhension d'un code source par un humain.

Compilateur et interprète

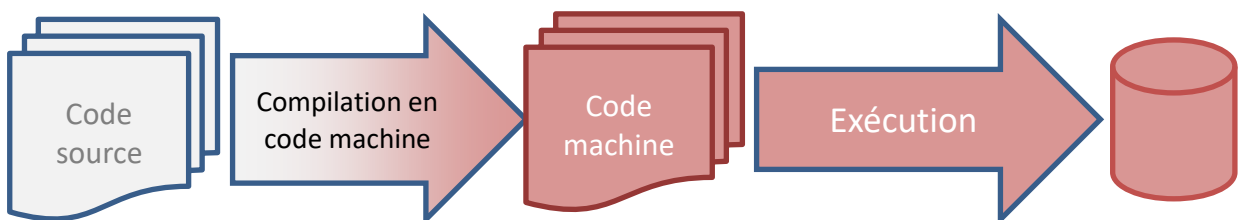
La traduction des textes écrits dans un langage de programmation en instructions machines est réalisée soit par des interprètes, soit par des compilateurs.

Les interprètes :



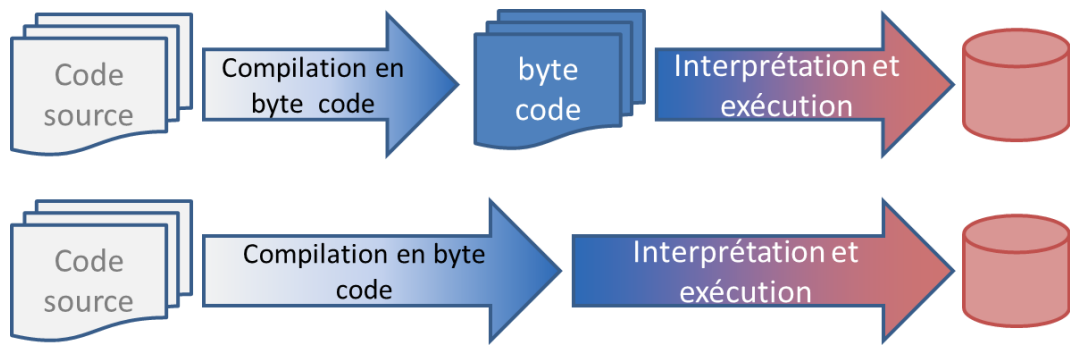
Les interprètes traduisent et exécutent les instructions les unes après les autres.

Les compilateurs :



Les compilateurs traduisent toutes les instructions du programme en langage machine et sauvegarde cet état dans un fichier exécutable.

L'ordinateur exécute le code machine sans utiliser le code source, ce qui permet de gagner du temps à l'exécution.

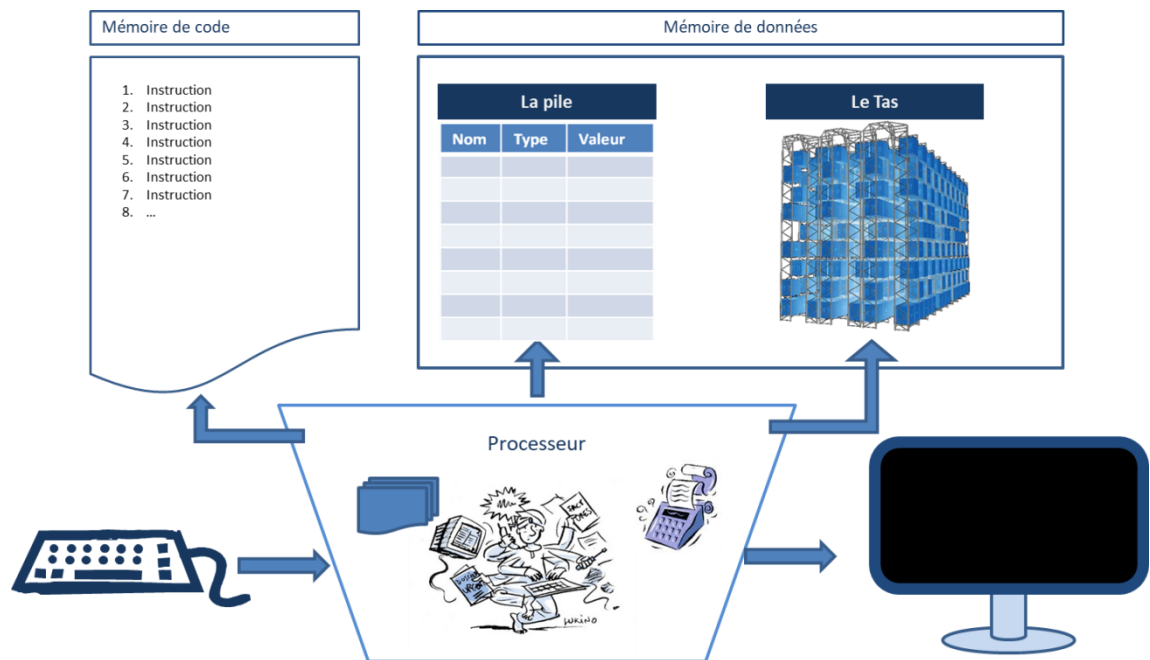
Solutions ibrides

Logique de programmation

2. Machine Logique

Description	10
Début et fin de programme.....	11
Syntaxe	11
Procédure	11
Représentation	11
Premier programme	11
Opérations et types de données	12
Type de données numériques	12
Opérateurs sur les numériques :	12

Description



Pour pouvoir apprendre à écrire des programmes, il est nécessaire de connaître les caractéristiques et les compétences des machines avec lesquelles nous voulons communiquer.

De façons à nous concentrer sur la logique des programmes, nous imaginons une machine logique ne reprenant que les éléments nécessaires à notre propos.

1. La mémoire de code : liste d'instructions numérotées.
2. La mémoire de données : ensemble de zones où il est possible de mémoriser des informations. Cet ensemble est organisé de deux manières différentes la pile et le tas.
 - a. La pile : à chaque zone mémoire est associé un nom qui peut être utilisé dans les instructions
 - b. Le tas : les zones mémoires sont accessibles via leurs adresses.
3. Un canal d'entrée (par exemple le clavier)
4. Un canal de sortie (par exemple l'écran)
5. Le processeur est l'élément dynamique de notre machine
 - a. Elle possède un ensemble de procédures permettant d'exécuter les instructions de notre langage logique.
 - b. Elle est capable d'effectuer des opérations simples sur les données afin de produire de nouvelles données résultats.
 - c. Elle est capable de charger une instruction dans la mémoire de code.
 - d. Elle est capable de lire et de modifier les données de la mémoire de données.
 - e. Elle est capable de lire les données sur le canal d'entée
 - f. Elle est capable d'écrire des données sur le canal de sortie

Début et fin de programme

Syntaxe

Debut nomDuProgramme

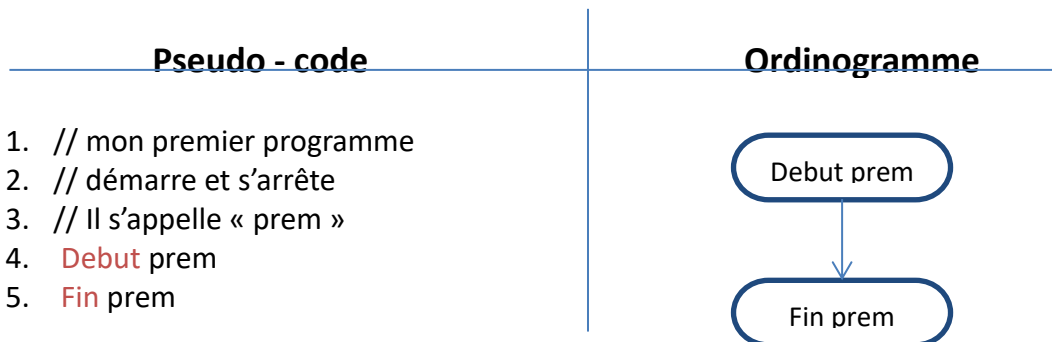
Fin nomDuProgramme

Procédure

Au lancement du programme la Machine Logique (ML) :

- Mémorise le nom du programme
- Charge le code dans la mémoire de code
- Recherche une instruction « Debut » suivie du nom du programme à exécuter
- Exécute l'instruction ayant le numéro suivant.
- Arrête le programme quand il exécute l'instruction « Fin » suivie du nom du programme mémorisé

Représentation



Premier programme

1. Lancement du programme « prem »
2. Mémorisation du nom du programme dans la machine logique
3. Chargement du code dans la mémoire du code
4. La M.L. parcourt les instructions à la recherche de l'instruction de début :
 - a. Ligne 1 : ligne de commentaire => suivante
 - b. Ligne 2 : ligne de commentaire => suivante
 - c. Ligne 3 : ligne de commentaire => suivante
 - d. Ligne 4 : Début du programme « prem » => trouvé
5. Chargement de la ligne suivante
6. Fin de programme

Opérations et types de données

La manière de réaliser une opération sur des données dépend du type des données

- Certaines opérations n'ont aucun sens sur certains types de données
 - La technique pour filtrer de l'eau est très différente de la technique pour filtrer les entrées à une soirée
 - Tandis que filtrer des montagnes n'a pas de sens

Il en est de même pour la machine logique. Une opération sur des valeurs ne pourra être exécutée qu'en fonction du type de données de ces valeurs. Il sera donc nécessaire de définir les types de données connues par la machine logique et pour chaque type :

- Le symbole identifiant du type
- La liste des valeurs possibles du type
- les opérations possibles sur les valeurs du type
- les règles d'écriture des valeurs littérales (si le type le permet)

Type de données numériques

Pour la ML une donnée numérique est équivalente à un nombre réel en algèbre.

- Symbole identifiant du type : **N**
- Règles d'écriture : identiques aux règles d'écriture des nombres décimaux en algèbre
 - Par exemple : 12 456,7 0,005

Opérateurs sur les numériques :

Priorité	Opérateurs	Description
1	()	Expression entre parenthèses
2	a^b , \sqrt{a}	Exposant, racine carrée
3	+, -	Signe plus, signe moins
4	*, /, DIV, MOD	Multiplication, division, division entière, modulo
5	+, -	Addition, soustraction

Logique de programmation

3. Gestion des variables

Définitions	14
Déclaration des variables et des constantes	14
Syntaxes.....	14
Exemple de déclarations	14
Représentation	15
Création des variables	15
Evaluation des Expressions	16
Procédure d'évaluation d'une expression.....	16
Instruction d'assignation	16
Description	16
Syntaxe :	16
Représentation	17
Table de valeur	17

Définitions

- **Constante littérale** : donnée écrite directement dans le code
- **Constante symbolique** : Nom attribué à une valeur. La valeur attribuée ne pourra pas être modifiée pendant l'exécution du programme
- **Variable** : la variable associe également un nom à une valeur, mais la valeur pourra être modifiée lors de l'exécution du programme
 - La valeur des variables est enregistrée dans la pile
- **Assignment** : opération d'attribution d'une valeur à une variable

Déclaration des variables et des constantes

Les variables et les constantes sont déclarées en début de programme dans le bloc de déclaration

Syntaxes

- Syntaxe du bloc de déclaration:

```
VARIABLES LOCALES :  
    [0..n] déclaration de constante  
    [0..n] déclaration de variable  
FIN VARIABLES LOCALES
```

- Syntaxe d'une déclaration de constante

```
CONST Identifiant : Type <- ConstanteLittérale // description
```

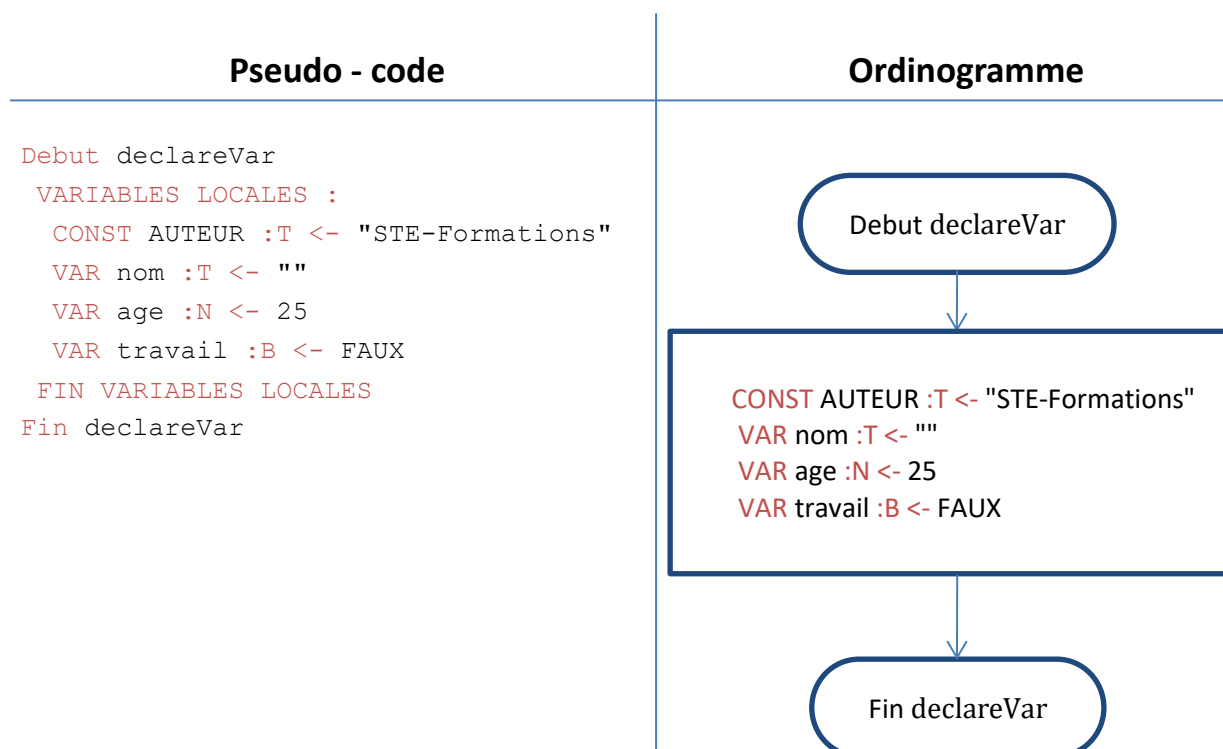
- Syntaxe d'une déclaration de variable

```
VAR Identifiant : Type <- ConstanteLittérale // description
```

Exemple de déclarations

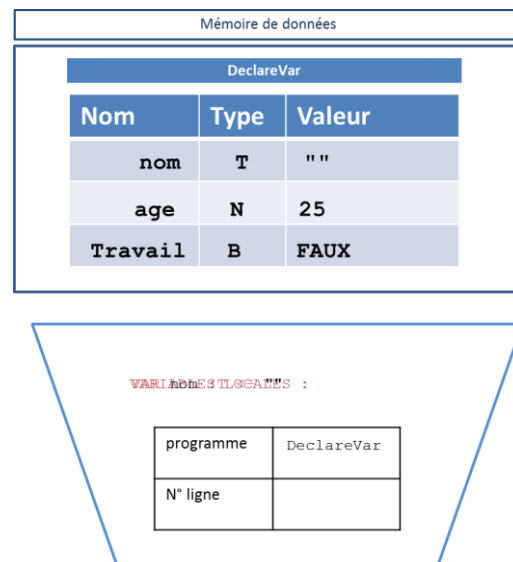
```
VARIABLES LOCALES :  
    // nom du responsable de l'algorithme  
    CONST AUTEUR :T <- "STE-Formations"  
    VAR nom :T <- "" // nom du stagiaire  
    VAR age :N <- 25 // age du stagiaire  
    // travail = VRAI si le stagiaire à un contrat d'emploi  
    VAR travail :B <- FAUX  
FIN VARIABLES LOCALES
```

Représentation



Création des variables

1. Chargement du code en mémoire
2. Début de programme
3. Début de bloc de déclarations des variables locales
4. Création une page mémoire pour le programme.
5. Déclaration de la constante « **AUTEUR** »
6. Déclaration de la variable « **nom** »
7. Création de la variable « **nom** » de type **Texte** et de valeur **chaîne vide**.
8. Déclaration et création de la variable « **age** » de type **Numérique** et de valeur **25**.
9. Déclaration et création de la variable « **travail** » de type **booléenne** et de valeur **FAUX**.
10. Fin de déclaration des variables locales
11. Fin de programme
12. Destruction de la page mémoire



Evaluation des Expressions

Lorsqu'une instruction contient une expression, la ML commence par évaluer l'expression avant d'effectuer l'instruction

Procédure d'évaluation d'une expression

1. Chaque variable est remplacée par sa valeur actuelle
 1. X par 15
 2. Y par 12
 3. Z par -1
 4. X par 15
2. Chaque opération est effectuée puis remplacée par son résultat dans l'ordre de priorité des opérateurs
 1. $5 * 15$ par 75
 2. $2 * 12$ par 24
 3. $24 * -1$ par -24
 4. $75 - -24$ par 99
 5. $99 \bmod 15$ par 9

Mémoire de données		
Nom	Type	valeur
X	N	15
Y	N	12
Z	N	-1

$(5 * X - 2 * Y * Z) \bmod X$

Instruction d'assignation

Description

- Une instruction d'assignation provoque la modification de la valeur d'une variable
- La valeur assignée à une variable doit être de même type que la variable
- Lors de l'assignation d'une expression dans une variable
 1. L'expression est évaluée
 2. Le résultat de l'évaluation est assignée à la variable

Attention : la valeur se trouvant dans la variable avant l'assignation est définitivement perdue à la fin de l'instruction

Syntaxe :

nomVariable \leftarrow Expression

Représentation

Pseudo - code

```

Debut ex1
  variables locales
    var a :N <- 1
    var b :N <- 2
  fin variables locales
  b <- a + b
  a <- b - a
  b <- a + b
  a <- b - a
  b <- a + b
  a <- b - a
Fin ex1

```

Ordinogramme

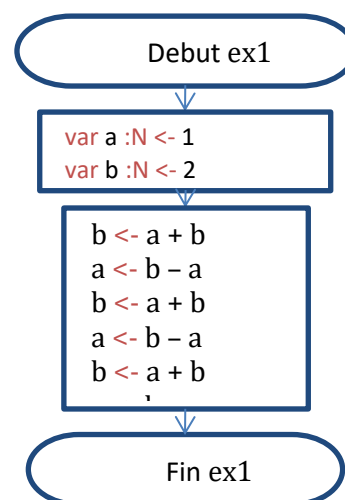


Table de valeur

Les tables de valeurs reprennent l'état de la mémoire à la fin de chaque instruction

```

1. Debut ex1
2.   variables locales
3.     var a :N <- 1
4.     var b :N <- 2
5.   fin variables locales
6.   b <- a + b
7.   a <- b - a
8.   b <- a + b
9.   a <- b - a
10.  b <- a + b
11.  a <- b - a
12. Fin ex1

```

N° de ligne	a	b
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		

Logique de programmation

4. Affichage et saisie

Afficher	20
Syntaxe :	20
Deroulement	20
Représentation	20
Exemple :	20
Saisir	21
Syntaxe :	21
Remarques :	21
Représentation	21
Affichage et la lecture dans une table de valeurs	22

Afficher

L'instruction « afficher » provoque l'affichage d'une liste de valeur à l'écran.

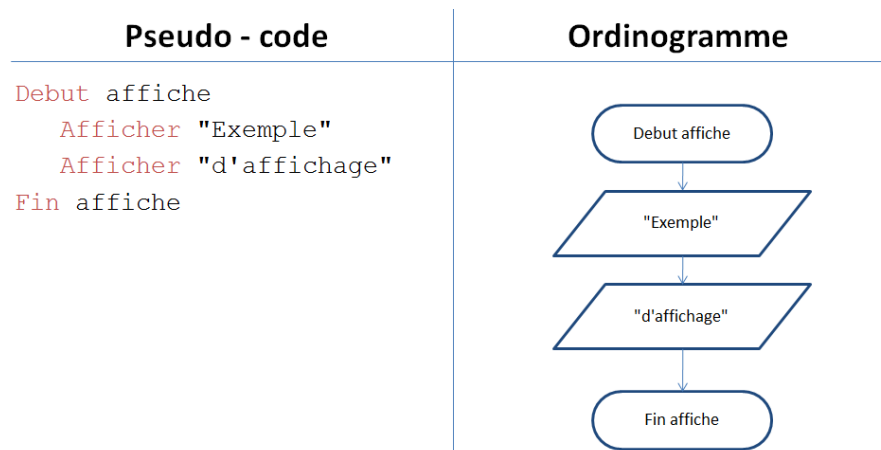
Syntaxe :

afficher expression1, expression2, ...

Déroulement

1. La ML calcule le résultat de chaque expression et affiche les résultats les uns à la suite des autres.
2. Si une expression commence et se termine par le caractère guillemet (") le texte entre les guillemets est affiché tel quel à l'écran.

Représentation



Exemple :

afficher "la division entière de ", a, " par ", b, " est ", a DIV b

Déroulement

1. Affichage de la première valeur « la division entière de »
2. Remplacement de a par 7
3. Affichage de la deuxième valeur « 7 »
4. Affichage de la troisième valeur « par »
5. Remplacement de b par 3
6. Affichage de la quatrième valeur « 3 »
7. Remplacement de a par 7
8. Remplacement de b par 3
9. Remplacement de « 7 DIV 3 » par 2
10. Affichage de la cinquième valeur « 2 »

Si au moment de l'exécution de l'instruction a=7 et b=3, la ligne affichée à l'écran sera:

la division entière de 7 par 3 est 2

Saisir

L'instruction "saisir" provoque : l'interruption de l'exécution du programme dans l'attente d'une valeur communiquée par l'utilisateur et la réception et la mise en mémoire de la valeur reçue

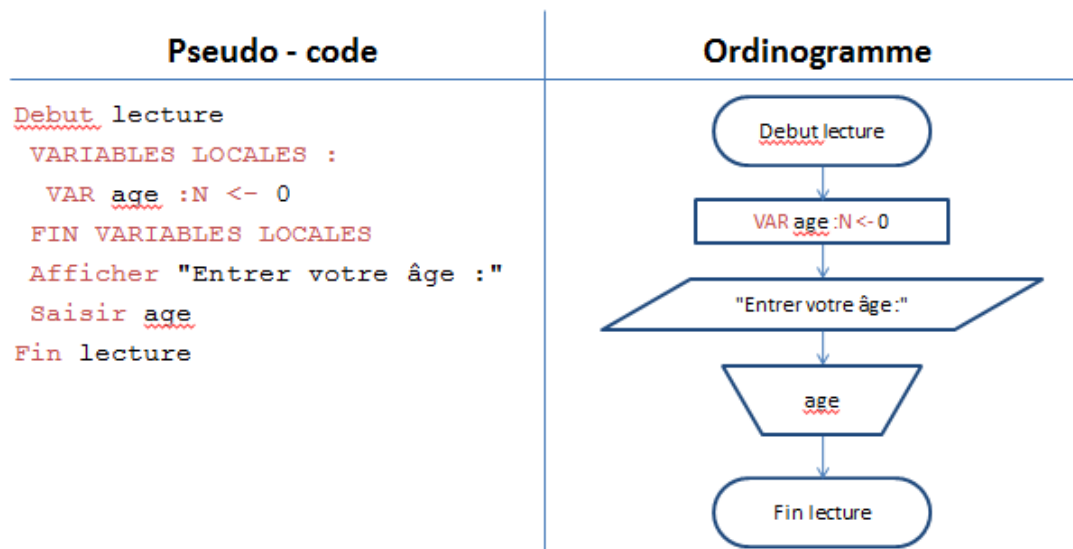
Syntaxe :

saisir nomDeVariable

Remarques :

- Le type de la valeur saisie est celui de la variable
- La machine Logique n'accepte que des valeurs de ce type
- Quand la machine logique reçoit une valeur du bon type, elle modifie la valeur de la variable dans la mémoire de données

Représentation



Affichage et la lecture dans une table de valeurs

```

1. Debut multiplie
2.  VARIABLES LOCALES :
3.   VAR a :N <- 0
4.   VAR b :N <- 0
5.   VAR result : N <- 0
6.  FIN VARIABLES LOCALES
7.  Afficher "MULTIPLICATION"
8.  Afficher "premier terme :"
9.  Saisir a
10. Afficher "deuxième terme :"
11. Saisir b
12. result <- a * b
13. Afficher a," x ",b," = ",result
14. Fin multiplie

```

N°	a	b	<u>result</u>	Ecran / clavier
1	?	?	?	
2	?	?	?	
3	0	?	?	
4	0	0	?	
5	0	0	0	
6	0	0	0	
7	0	0	0	-> MULTIPLICATION
8	0	0	0	-> premier terme:
9	5	0	0	<- 5
10	5	0	0	-> deuxième terme:
11	5	7	0	<- 7
12	5	7	35	
13	5	7	35	-> 5 x 7 = 35
14	?	?	?	

Logique de programmation

5. Type de données booléennes

Valeurs booléennes.....	24
Opérateurs booléens.....	24
La négation (non)	24
La conjonction (ET)	24
La disjonction (OU)	24
Les opérateurs de comparaison	25
Priorités de opérateurs.....	25
Expression booléenne	25
Exemple d'évaluation d'une expression booléenne	25
Syntaxe	25
Exemple de code	26

Valeurs booléennes

- Symbole identifiant du type : **B**
- Règles d'écriture : **VRAI** ou **FAUX**

Opérateurs booléens

La négation (non)

L'opérateur de négation donne un résultat inverse à la valeur de son opérande

Table de valeurs :

A	Non A
VRAI	FAUX
FAUX	VRAI

La conjonction (ET)

Le résultat d'une conjonction n'est **VRAI** que si ses deux opérandes sont **VRAI**

Table de valeurs :

A	B	A et B
VRAI	VRAI	VRAI
VRAI	FAUX	FAUX
FAUX	VRAI	FAUX
FAUX	FAUX	FAUX

La disjonction (OU)

Le résultat d'une conjonction est **VRAI** si au moins un de ses deux opérandes est **VRAI**

Table de valeurs :

A	B	A ou B
VRAI	VRAI	VRAI
VRAI	FAUX	VRAI
FAUX	VRAI	VRAI
FAUX	FAUX	FAUX

Les opérateurs de comparaison

- Les opérateurs de comparaison ont comme résultat une valeur booléenne
- Les deux opérandes d'une comparaison doivent être de même type et la comparaison doit être implémentée par le type de données
- Les comparateurs d'égalité et de différence ($=$, \neq) sont définis pour :
 - les valeurs de type numérique
 - les valeurs de type booléen
- Les comparateurs d'ordre ($<$, \leq , $>$, \geq) sont définis pour :
 - les valeurs de type numérique.
- Remarque : pour chaque nouveau type, il faudra définir quels opérateurs de comparaison sont définis.

Priorités de opérateurs

Priorité	Opérateurs	Description
1, 2	$()^b, a, \sqrt{a}$	Expression entre parenthèses, exposant
3	$+, -, \text{non}$	Signe plus, signe moins, négation
4, 5, 6	...	Opérateurs numériques
7	$<, \leq, >, \geq$	plus petit, plus petit ou égal, plus grand, plus grand ou égal
8	$=, \neq$	Égalité, différence
9	et	Conjonction
10	ou	Disjonction

Expression booléenne

Une expression booléenne est une expression dont le résultat est une valeur booléenne

Exemple d'évaluation d'une expression booléenne

```
( 45 / 5 ≥ 6 + 4 ) ou non ( 3 * 4 = 7 ) et ( VRAI ou ( 5 ≠ 5 ) )
( 9 ≥ 6 + 4 ) ou non ( 3 * 4 = 7 ) et ( VRAI ou ( 5 ≠ 5 ) )
( 9 ≥ 10 ) ou non ( 3 * 4 = 7 ) et ( VRAI ou ( 5 ≠ 5 ) )
    FAUX ou non ( 3 * 4 = 7 ) et ( VRAI ou ( 5 ≠ 5 ) )
    FAUX ou non ( 12 = 7 ) et ( VRAI ou ( 5 ≠ 5 ) )
    FAUX ou non FAUX et ( VRAI ou ( 5 ≠ 5 ) )
    FAUX ou VRAI et ( VRAI ou ( 5 ≠ 5 ) )
    FAUX ou VRAI et ( VRAI ou FAUX )
    FAUX ou VRAI et VRAI
    FAUX ou VRAI
```

Syntaxe

Déclaration variable booléenne

VAR *Identifiant* : B <- *ConstanteBooléenne* // *description*

Assignation variable booléenne

Identifiant <- ExpressionBooléenne

Expression booléenne

ConstanteBooléenne

VariableBooléenne

OpérationdeComparaison

(ExpressionBooléenne)

NON ExpressionBooléenne

ExpressionBooléenne **ET** ExpressionBooléenne

ExpressionBooléenne **OU** ExpressionBooléenne

Constante booléenne

VRAI | **FAUX**

Opération de comparaison

ExpressionBooléenne opérateurEgalité ExpressionBooléenne

ExpressionNumérique opérateurEgalité ExpressionNumérique

ExpressionNumérique comparateurOrdre ExpressionNumérique

Opérateur d'égalité

= | **≠**

Comparateur d'ordre

< | **≤** | **>** | **≥**

Exemple de code

```
1. Debut bonCafe
2.   variables locales
3.     var qt :N <- 0
4.     var sucre :N <- 0
5.     var bon :B <- FAUX
6.   fin variables locales
7.   Afficher "quantité de café : "
8.   Saisir qt
9.   Afficher "Nombre de sucre : "
10.  Saisir sucre
11.  bon <- sucre = 1 et qt > 0,15 et qt <= 0,20
12.  Afficher "c'est un bon café : ", bon
13. Fin bonCafe
```

Logique de programmation

6. Structure de contrôle : l'alternative

Alternative simple	28
Syntaxe de la structure "alternative simple":	28
Représentation	28
Table de valeurs	29
Alternative simple variante	29
Syntaxe de la structure "alternative simple variante":	29
Alternatives simples imbriquées	30
Ordinogramme	30
Pseudo-code	30
Alternative composée	30
Syntaxe de la structure "alternative composée":	31
Représentation	31

Alternative simple

La structure alternative permet d'effectuer une suite d'instructions si une condition est remplie et d'en effectuer une autre si celle-ci ne l'est pas.

L'exécution de l'alternative commence par l'évaluation de la condition (vraie ou fausse) suivie de l'exécution de la suite d'instructions associées à la réponse obtenue.

Syntaxe de la structure "alternative simple":

SI expressionBooléen

ALORS

[0..n] Instructions

SINON

[0..n] Instructions

FINSI

Lorsque le résultat de l'évaluation de l'expression booléenne est :

- **VRAI** : seules les instructions entre le **ALORS** et le **SINON** sont exécutées
- **FAUX** : seules les instructions entre le **SINON** et le **FINSI** sont exécutées

Représentation

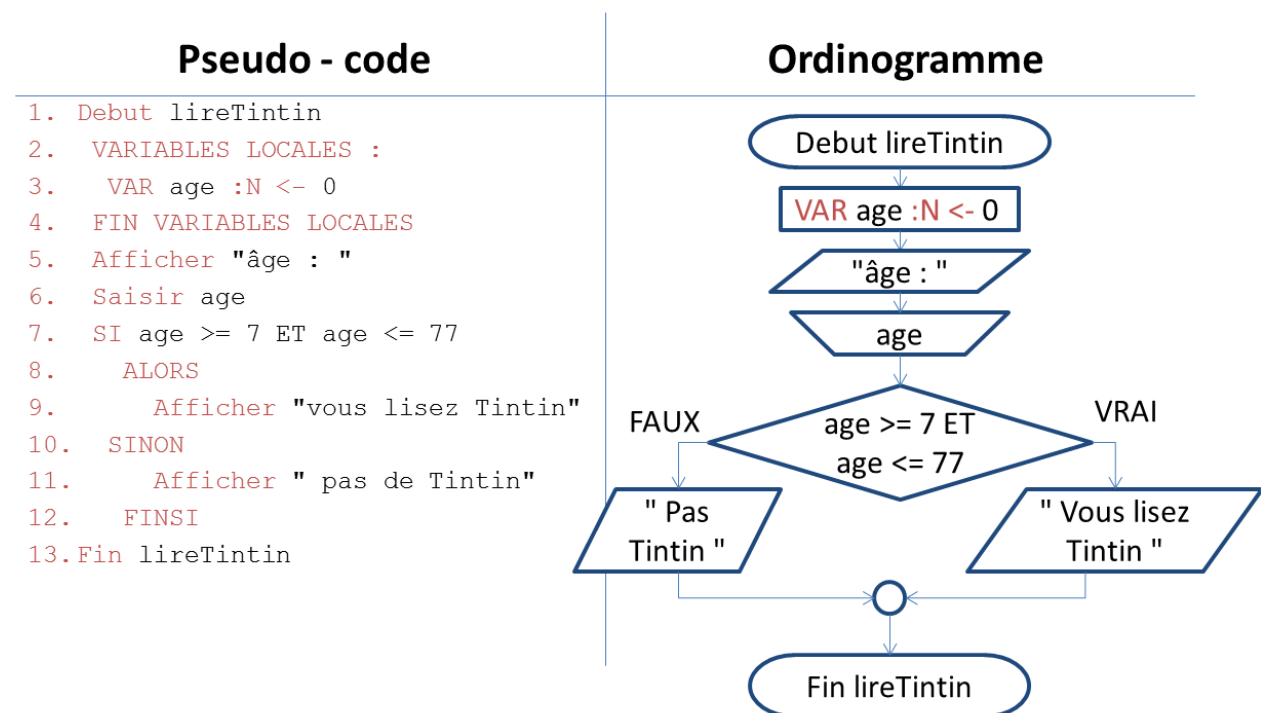


Table de valeurs

1. Debut lireTintin	N°	age	Ecran / clavier	N°	age	Ecran / clavier
2. VARIABLES LOCALES :	1	?		1	?	
3. VAR age :N <- 0	2	?		2	?	
4. FIN VARIABLES LOCALES	3	0		3	0	
5. Afficher "âge : "	4	0		4	0	
6. Saisir age	5	0	-> âge :	5	0	-> âge :
7. SI age >= 7 ET age <= 77	6	80	<- 80	6	35	<- 35
8. ALORS	7	80		7	35	
9. Afficher "vous lisez Tintin"	10	80		8	35	
10. SINON	11	80	->pas de Tintin	9	35	->vous lisez Tintin
11. Afficher " pas de Tintin"	12	80		12	35	
12. FINSI	13	?		13	?	
13. Fin lireTintin						

Alternative simple variante

Lorsque aucune instruction n'est à exécuter quand le test est faux, on n'indiquera pas le sinon.

Syntaxe de la structure "alternative simple variante":

SI experssionBooléen

ALORS

[1..n] Instructions

FINSI

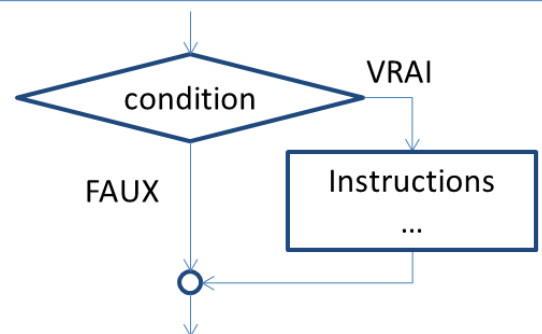
Pseudo - code

```

SI condition
  ALORS
    Instruction
  ...
FINSI

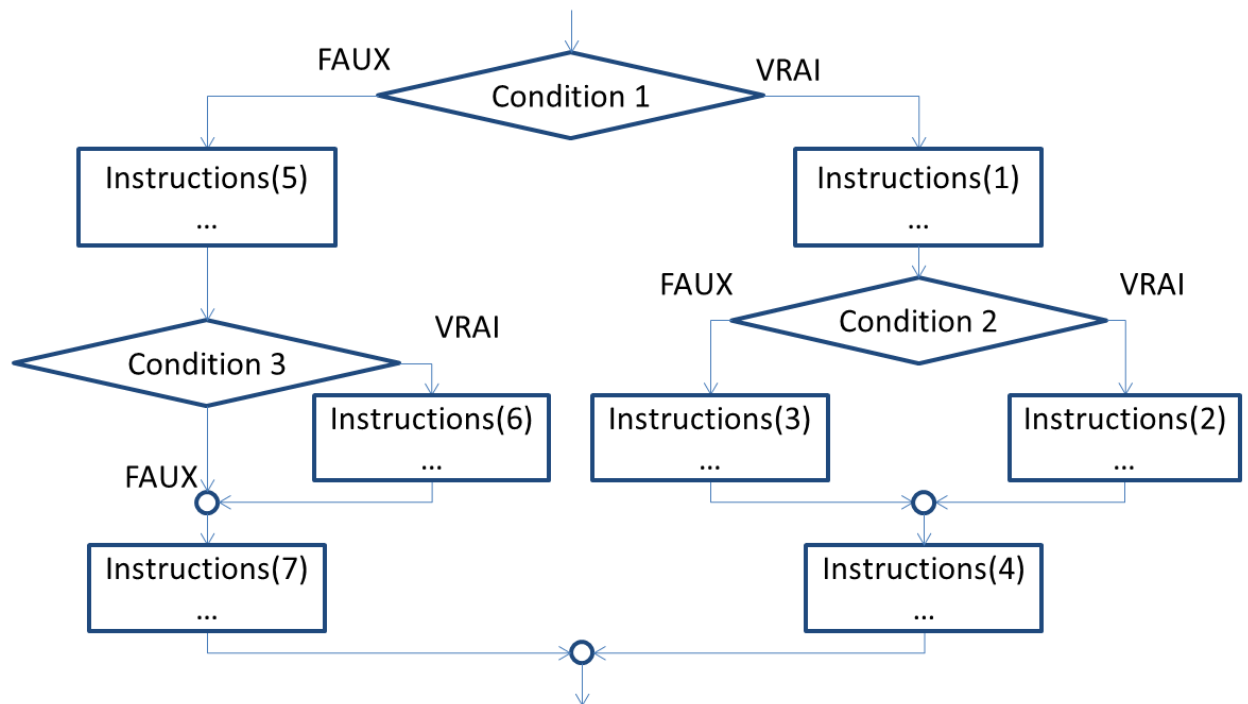
```

Ordinogramme



Alternatives simples imbriquées

Ordinogramme



Pseudo-code

```

SI condition 1
ALORS
  instructions(1)
SI condition 2
  ALORS
    instructions(2)
  SINON
    instructions(3)
  FINSI
instructions(4)
SINON
  instructions(5)
  SI condition 3
  ALORS
    instructions(6)
  FINSI
instructions(7)
FINSI
  
```

Alternative composée

La structure alternative composée permet d'effectuer une suite d'instructions en fonction de la valeur d'une variable.

L'exécution de l'alternative commence par une recherche du bloc d'instructions liées à la valeur de la variable et se poursuit en exécutant les instructions de ce bloc.

Syntaxe de la structure "alternative composée":

CAS OU expressionNumérique

CAS valeurNumérique
[0..n] Instructions

CAS autreValeurNumérique
[0..n] Instructions

...

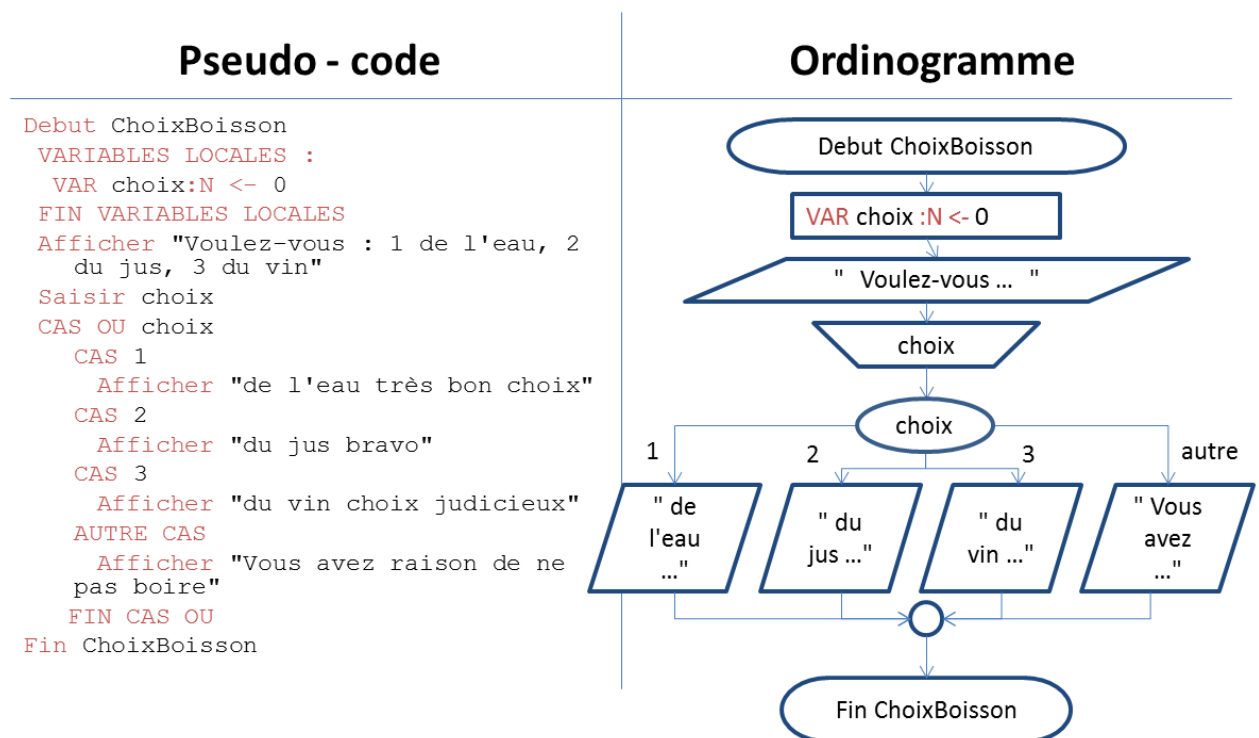
AUTRES CAS

[0..n] Instructions

FIN CAS OU

- Seules les instructions se trouvant dans le cas qui a la même valeur que l'expression numérique seront exécutées.
- Si aucun cas n'a la valeur de l'expression, ce sont les instructions de "AUTRE CAS" qui sont exécutées.

Représentation



Logique de programmation

7. Structure de contrôles : les Boucles

La boucle « tant que »	34
Syntaxe de la structure "tant que":	34
L'exécution de la boucle « tant que » :	34
Bonnes pratiques.....	34
Exemple	35
Table de valeurs	35
La boucle « Pour »	36
Boucle « Pour » : une boucle« Tant que » particulière.....	36
Représentation	36
Exécution de la boucle «pour » :	36
Syntaxe	37
Remarques :.....	37
Variante si le pas est négatif	37
Exemple	38
Table de valeurs	38
La boucle « jusqu'à ce que »	39
Syntaxe de la structure « Jusqu'à ce que »:	39
Représentation	39
Choisir le type de boucle	39
Transformation de boucle	40

La boucle « tant que »

La structure de boucle permet d'effectuer plusieurs fois une suite d'instructions.

Avec la boucle « Tant que », la suite d'instructions sera répétée tant qu'une condition est remplie.

Syntaxe de la structure "tant que":

```
TANT QUE expression Booléen
    [0..n] Instructions
FIN TANT QUE
```

L'exécution de la boucle « tant que » :

1. Evaluer de la condition de boucle :
 - si vrai passer au point 2,
 - si faux passer au point 3;
2. Exécuter les instructions de la boucle puis revenir au point 1;
3. Exécuter les instructions se trouvant après la boucle.

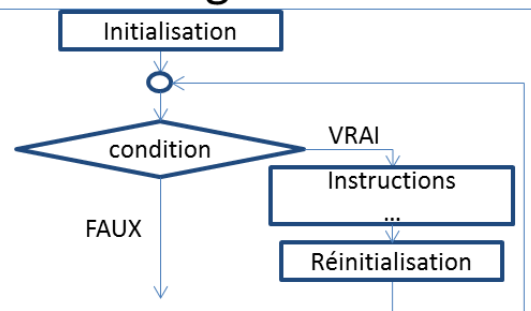
Bonnes pratiques

1. Pour que l'exécution puisse se terminer, il faut que la condition devienne fausse
2. Il faut que les instructions de la boucle modifient les variables de la condition
3. Pour rendre le code plus lisible et éviter les erreurs à ce niveau, la bonne pratique veut que :
 - Les instructions juste avant la condition initialisent les variables de la condition (même si elles le sont déjà par ailleurs)
 - Les instructions modifiant ces variables dans la boucle se situent juste avant le "FIN TANT QUE".

Pseudo - code

```
Initialisation
TANT QUE condition
    Instruction
    ...
Réinitialisations
FIN TANT QUE
```

Ordinogramme



Exemple

Pseudo - code

```

Debut sommeSuite
VARIABLES LOCALES :
    VAR somme :N <- 0
    VAR entre :N <- 0
FIN VARIABLES LOCALES
Afficher "Entrez les valeurs à
    additionner."
Afficher "Entrez 0 pour terminer."
Saisir entre
TANT QUE entre ≠ 0
    somme <- somme + entre
    Saisir entre
FIN TANT QUE
Afficher "Somme :", somme
Fin sommeSuite

```

Ordinogramme

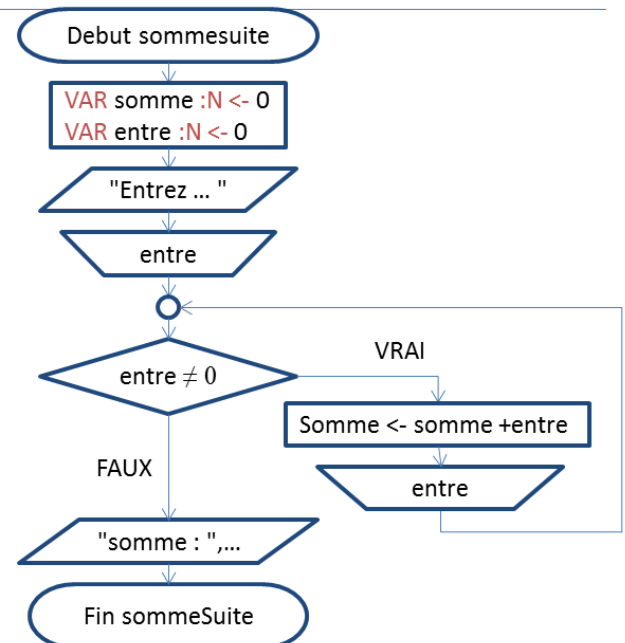


Table de valeurs

```

1.  Debut sommeSuite
2.  VARIABLES LOCALES :
3.      VAR somme :N <- 0
4.      VAR entre :N <- 0
5.  FIN VARIABLES LOCALES
6.  Afficher "Entrez les
    valeurs à additionner."
7.  Afficher "Entrez 0 pour
    terminer."
8.  Saisir entre
9.  TANT QUE entre ≠ 0
10.     somme <- somme + entre
11.     Saisir entre
12.  FIN TANT QUE
13.  Afficher "Somme :", somme
14.  Fin sommeSuite

```

N°	somme	entre	Ecran / clavier
5	0	0	
6	0	0	-> Entrez les valeurs à additionner
7	0	0	-> Entrez 0 pour terminer
8	0	5	<- 5
9,10	5	5	
11	5	35	<- 35
12, 9, 10	40	35	
11	40	-10	<- -10
12, 9, 10	30	-10	
11	30	0	<- 0
12, 9, 13	30	0	-> Somme : 30
14	?	?	

La boucle « Pour »

Boucle « Pour » : une boucle « Tant que » particulière

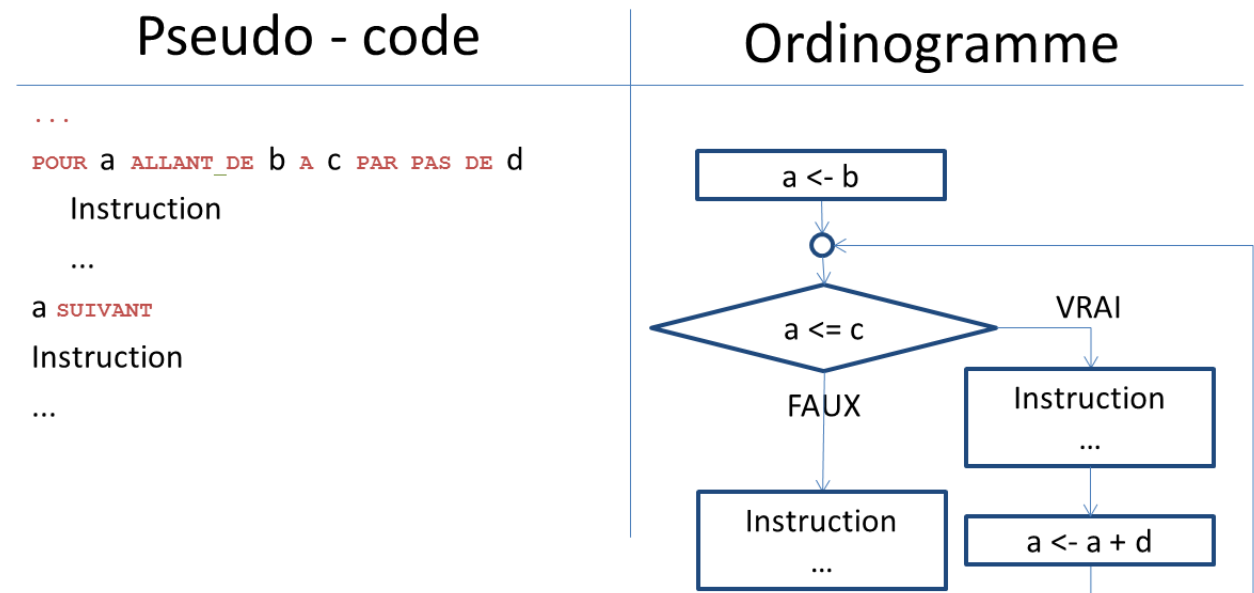
- Le nombre d'itérations de la boucle est fixe.
- Il n'existe pas de cas où la boucle doit se terminer avant le nombre d'itérations.

Exemple

```

cpt <- 1
TANT QUE cpt <= 10
  Instruction
...
cpt <- cpt + 1
FIN TANT QUE
    
```

Représentation



Exécution de la boucle «pour» :

1. Assigner de la valeur « b » à la variable « a »
2. Evaluer si « a » est plus petit ou égal à « c »
 - si vrai passer au point 3,
 - si faux passer au point 6;
3. Exécuter les instructions de la boucle
4. Incrémenter « a » de la valeur de « d »
5. Revenir au point 2;
6. Exécuter les instructions se trouvant après la boucle.

Syntaxe

POUR a ALLANT DE b A c [PAR PAS DE d]
[0..n] Instructions
a SUIVANT

- a : variable numérique entière
- b : expression numérique entière
- c : expression numérique entière
- d : constante numérique entière positive

Remarques :

- La variable « a » et celles utilisées pour définir les valeurs « b » et « c » ne peuvent pas être modifiées pendant l'exécution de la boucle.
- La définition du pas est facultative. Si elle n'est pas définie « d »=1.

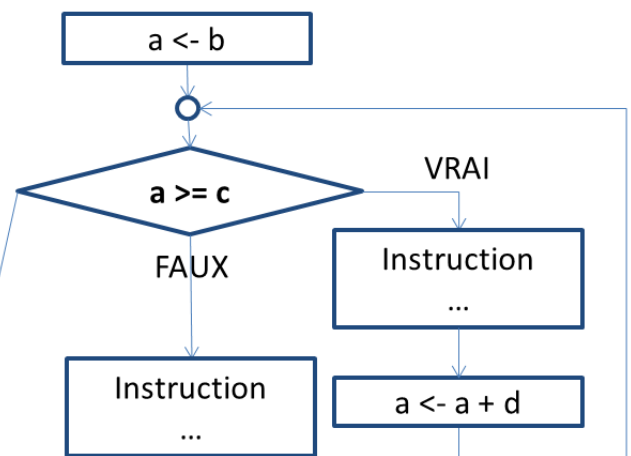
Variante si le pas est négatif

Dans la représentation ci-dessous la valeur de la variable « d » est négative

Pseudo - code

...
 POUR a ALLANT DE b A c PAR PAS DE d
 Instruction
 ...
 a SUIVANT

Le test de boucle change:
 A <= c devient a >= c

Ordinogramme

Exemple

Pseudo - code

```

Debut somme5Nombres
  VARIABLES LOCALES :
    VAR somme :N <- 0
    VAR entre :N <- 0
    VAR cpt :N <- 0
  FIN VARIABLES LOCALES
  Afficher "Entrez les 5 valeurs à
    additionner."
  POUR cpt ALLANT DE 1 A 5
    Saisir entre
    somme <- somme + entre
    cpt SUIVANT
  Afficher "Somme :", somme
Fin somme5Nombres

```

Ordinogramme

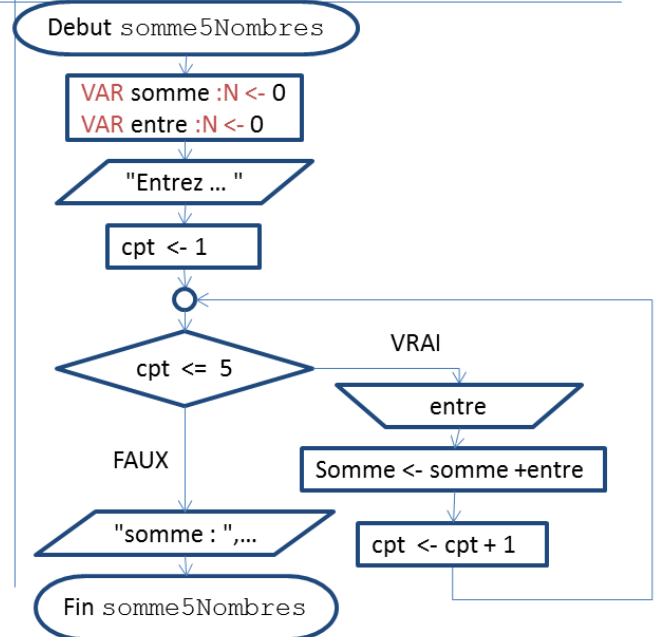


Table de valeurs

```

1.  Debut somme3Nombres
2.    VARIABLES LOCALES :
3.      VAR somme :N <- 0
4.      VAR entre :N <- 0
5.      VAR cpt :N <- 0
6.    FIN VARIABLES LOCALES
7.    Afficher "Entrez les 3
      valeurs à additionner."
8.    POUR cpt ALLANT DE 1 A 3
9.      Saisir entre
10.     somme <- somme + entre
11.     cpt SUIVANT
12.     Afficher "Somme :", somme
13.  Fin somme3Nombres

```

N°	somme	entre	cpt	Ecran / clavier
6	0	0	0	
7	0	0	0	-> Entrez les 3 valeurs à additionner
8	0	0	1	
9	0	5	1	<- 5
10	5	5	1	
11	5	5	2	
8,9	5	35	2	<- 35
10	40	35	2	
11	40	35	3	
8,9	40	-10	3	<- -10
10	30	-10	3	
11	30	-10	4	
8,12	30	-10	4	-> Somme : 30
13	?	?	?	

La boucle « jusqu'à ce que »

Avec la boucle « Jusqu'à ce que » :

- La suite d'instructions sera exécutée jusqu'à ce que la condition soit **VRAI**
- La suite d'instructions contenue sera exécutée au moins 1 fois dans tous les cas.

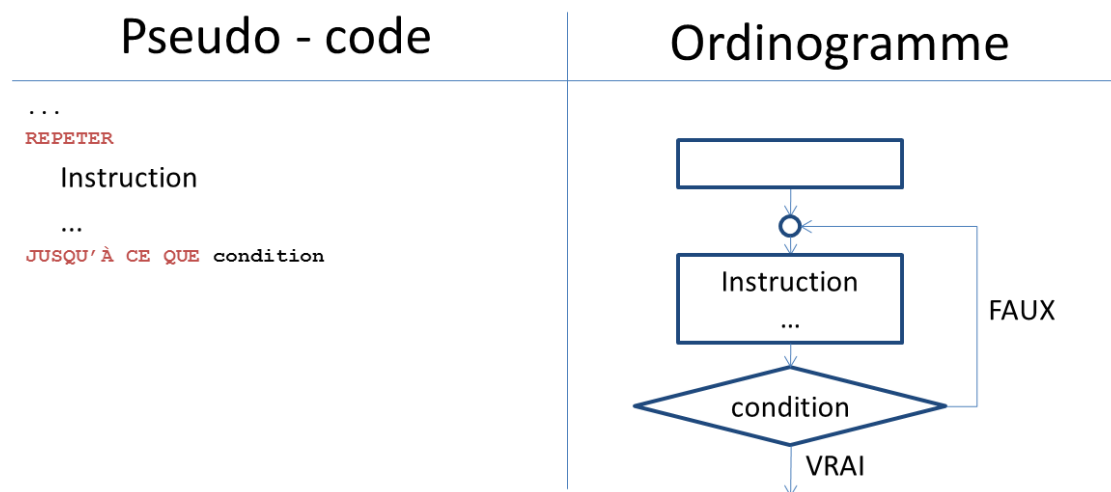
Syntaxe de la structure « Jusqu'à ce que »:

REPETER

[0..n] Instructions

JUSQU'À CE QUE expression booléenne

Représentation



Choisir le type de boucle

- Si le nombre d'itérations de la boucle est calculable avant le début des itérations :
 - Boucle « Pour »
- Sinon si le nombre d'itérations est toujours au minimum 1
 - Boucle « Jusqu'à ce que »
- Sinon
 - Boucle « Tant que »

Transformation de boucle**Boucle « Pour »**

```
POUR cpt ALLANT DE 3 A 10 PAR PAS  
  DE 2  
  Instruction  
  ...  
cpt SUIVANT
```

Boucle « Tant que »

```
cpt <- 3  
TANT QUE cpt <= 10  
  Instruction  
  ...  
  cpt <- cpt + 2  
FIN TANT QUE
```

Boucle « jusqu'à ce que »

```
REPETER  
  Instruction  
  ...  
JUSQU'À CE QUE condition
```

Boucle « Tant que »

```
Instruction  
...  
TANT QUE NON condition  
  Instruction  
  ...  
FIN TANT QUE
```


Logique de programmation

8. Type de données caractère texte

Type Caractère	41
Opérateurs sur les caractères :	41
Remarque	41
Type Texte	41
Opérateurs sur les textes:	42
Concaténation	42
Fonction de manipulation des textes	42

Type Caractère

Dans le cadre de ce cours de logique, nous utiliserons une définition simple de la notion de caractère. Nous nous limiterons aux lettres de l'alphabet latin + le caractère espace.

- Symbole identifiant du type : **C**
- Règles d'écriture : **'a', 'b'...**

Opérateurs sur les caractères :

Priorité	Opérateurs	Description
1	()	Expression entre parenthèses
3	\oplus	concaténation
9	=, \neq , <, \leq , >, \geq	Égalité, différence, plus petit, plus petit ou égal, plus grand, plus grand ou égal

Remarque

Un caractère est plus petit qu'un autre s'il est avant dans l'ordre alphabétique.

Type Texte

Un Texte est une suite de caractères.

- Symbole identifiant du type : **T**
- Règles d'écriture : **"mon texte"**

Opérateurs sur les textes:

Priorité	Opérateurs	Description
1	()	Expression entre parenthèses
3	⊕	concaténation
9	=, ≠, <, ≤, >, ≥	Égalité, différence, plus petit, plus petit ou égal, plus grand, plus grand ou égal

Concaténation

- Mettre bout à bout deux textes

Exemple

```

1  DEBUT concat
2    VARIABLES LOCALES :
3      VAR nom :T <- "Dupont"
4      VAR prenom :T <- "Dédé"
5      VAR carl :C <- '-'
6      VAR result :T <- ""
7  FIN VARIABLES LOCALES
8  result <- carl ⊕ carl // concaténation de caractères
9  result <- result ⊕ nom ⊕ " " ⊕ prenom // concaténation de textes
10 result <- result ⊕ carl ⊕ carl // concaténation de texte et de caractères
11 Afficher result
12 FIN concat

```

N°	result	Ecran / clavier
1,2,3,4,5,6	" "	
7,8	"_ "	
9	"--Dupont Dédé"	
10	"--Dupont Dédé--"	
11	"--Dupont Dédé--"	-> --Dupont Dédé--

Fonction de manipulation des textes

Fonction	Description
longueur(texte : T):N	Nombre de caractères du texte
caract(texte : T, position : N) :C	Caractère à la position dans le texte
sousChaine(texte : T, début : N, fin : N) :T	Sous chaine commençant à début et finissant à fin -1

Exemple

```

1. Debut testFonctions
2.  VARIABLES LOCALES :
3.    VAR entree :T <- ""
4.  FIN VARIABLES LOCALES
5.  AFFICHER "Entrer le texte à tester"
6.  SAISIR entree
7.  AFFICHER longueur(entree)
8.  AFFICHER caract(entree, 3)
9.  AFFICHER sousChaine(entree, 3, 8)
10. Fin testFonctions

```

1	2	3	4	5	6	7	8	9
m	o	n		t	e	x	t	e

N°	Clavier/Ecran
1,2,3,4	
5	->Entrer le texte à tester
6	<- mon texte
7	-> 9
8	-> n
9	-> n tex

Logique de programmation

9. Structure de données : tableau

Problématique.....	44
Définition :.....	44
Déclaration des tableaux.....	44
Syntaxe d'une déclaration de tableau.....	44
Exemple	45
Assignation et lecture des cases d'un tableau	45
Problématique résolue.....	45

Problématique

Dans une liste de 20 entiers, afficher le nombre de valeurs qui sont supérieures à la moyenne des éléments de cette liste.

- Les entiers doivent être parcourus deux fois :
 1. Pour calculer la moyenne.
 2. Pour comparer chaque entier à cette moyenne.
- Il faut enregistrer chaque entier
 1. Il faut 20 variables.
 2. Il n'est pas possible de faire une boucle qui répète la même opération sur des variables différentes.

Définition :

Un tableau est un ensemble de variables

- de même type,
- désignées par un même nom,
- et distinguées les unes des autres par leur numéro (appelé aussi indice).

Indices	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Valeurs	9	12	15	10	7	11	13	6	8	2	18	20	14	9	5	6	7	15	16	11

Déclaration des tableaux

- Les tableaux sont déclarés avec les autres variables.

VARIABLES LOCALES :

[0..n] déclaration de constante

[0..n] déclaration de variable

[0..n] déclaration de tableau

FIN VARIABLES LOCALES

Syntaxe d'une déclaration de tableau

VAR Identifiant : Type[Constante] // description

Remarques:

- La constante entre crochets détermine le nombre de cases du tableau.
- Les cases du tableau ne sont pas initialisées lors de la déclaration.

Exemple

```

DEBUT déclarationTableau
  VARIABLES LOCALES :
    CONST TAILLE:N <- 20 //nombre de cotes
    VAR cotes:N[TAILLE] //tableau de cotes
    VAR ind:N <- 1 //indice d'une cote
  FIN VARIABLES LOCALES

  //Saisie des cotes
  AFFICHER "Entrer les 20 cotes"
  POUR ind ALLANT DE 1 A TAILLE
    SAISIR cotes[ind]
  ind SUIVANT
FIN déclarationTableau

```

Assignation et lecture des cases d'un tableau

Chaque case d'un tableau est une variable et peut donc être utilisée comme tel.

```

cote[1] <- 5
a <- cote[1]
cote[1] <- cote[1] + cote[2]

```

Les indices de cases peuvent être calculés.

```

cote[ 5 + 2 ] <- 12
cote[ a ] <- cote[ a + 1 ]
cote[ a + 1 ] <- cote[ a ] + 1

```

Lors de la lecture ou de l'assignation d'une case si l'indice est hors des limites du tableau le programme est en erreur.

Problématique résolue

```

DEBUT déclarationTableau
  VARIABLES LOCALES :
    //nombre de cotes
    CONST TAILLE:N <- 20
    //tableau de cotes
    VAR cotes:N[TAILLE]
    //indice d'une cote
    VAR ind:N <- 1
    //somme des valeurs
    VAR somme:N <- 0
    //moyenne des valeurs
    VAR moy:N <- 0
    //nombre de valeurs >= moyenne
    VAR nbr:N <- 0
  FIN VARIABLES LOCALES

  //Saisie des cotes et calcul de somme
  AFFICHER "Entrer les ", TAILLE, "
  cotes"
  POUR ind ALLANT DE 1 A TAILLE
    SAISIR cotes[ind]
    somme <- somme + cote[ind]
  ind SUIVANT
  //Calcul de moyenne
  moy <- somme / TAILLE
  //Calcul nombre de valeurs >= moyenne
  POUR ind ALLANT DE 1 A TAILLE
    SI cotes[ind] >= moy
      ALORS
        nbr <- nbr + 1
      FINSI
  ind SUIVANT
  //Affichage du résultat
  AFFICHER nbr," valeurs >= ", moy
FIN déclarationTableau

```


Logique de programmation

10. Sous programmes : procédures et fonctions

Définition	48
Structure d'un sous-programme	48
Exemple	48
Appel d'un sous-programme	49
Ordre d'exécution du code	49
Variables paramètres	50
Portée et durée vie des variables	50
Fonction et procédure	50
Exemple	50

Définition

- Un sous-programme est une séquence d'instructions qui peut être appelée par un programme ou un sous-programme.
- Il est intéressant d'isoler une séquence d'instructions dans un sous-programme.
 - Lorsqu'une séquence d'instructions est répétée à plusieurs endroits d'un programme.
 - Lorsqu'une séquence d'instructions est réutilisable dans d'autres programmes.

Structure d'un sous-programme

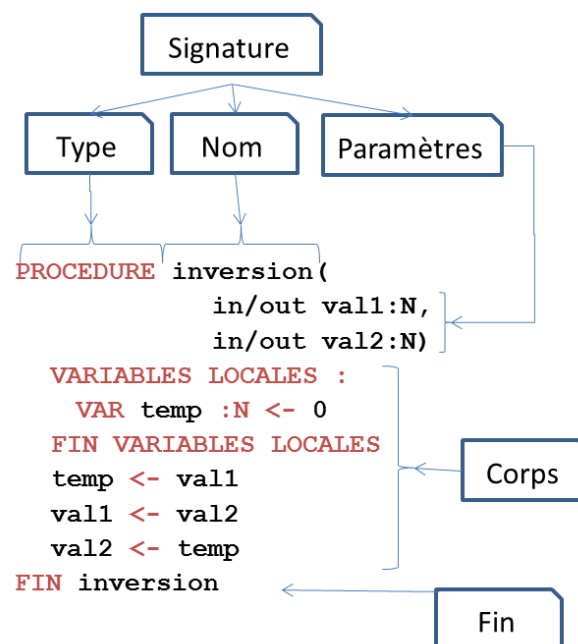
La définition d'un sous-programme commence par sa signature suivit de son code et se termine par un marque de fin de sous-programme.

La signature est composée du type (procédure / fonction) du nom suivit de la définition des paramètres)

Les paramètres sont des variables initialisées par le programme appelant leurs déclaration définissent leurs noms le type de donnée et le type de passage de paramètre.

Le corps d'un sous-programme peut contenir tous les types d'instructions.

Exemple



Appel d'un sous-programme

```

DEBUT tri
  VARIABLES LOCALES :
    VAR A :N <- 0
    VAR B :N <- 0
    VAR C :N <- 0
  FIN VARIABLES LOCALES
  AFFICHER "Entrer 3 valeurs"
  DEMANDER A
  DEMANDER B
  DEMANDER C
  SI A > B
    ALORS inversion(A,B)
  FINSI
  SI B > C
    ALORS
      inversion(B,C)
      SI A > B
        ALORS inversion(A,B)
      FINSI
    FINSI
  FINSI
  ...

```

```

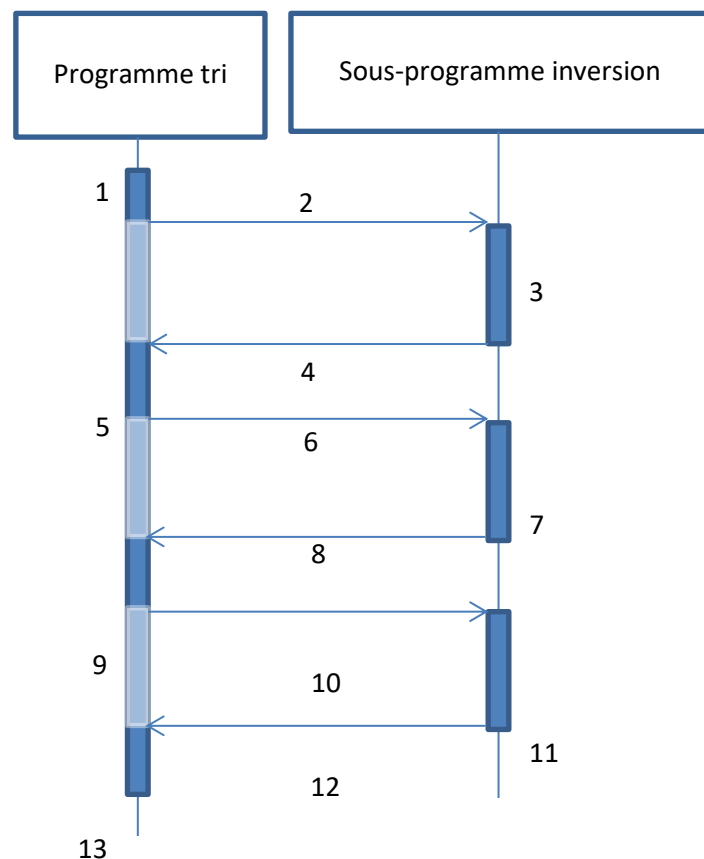
PROCEDURE inversion(
  in/out val1:N,
  in/out val2:N)
  VARIABLES LOCALES :
    VAR temp :N <- 0
  FIN VARIABLES LOCALES
  temp <- val1
  val1 <- val2
  val2 <- temp
FIN inversion

```

Programme de tri appelant le sous-programme inversion

Ordre d'exécution du code

1. le programme tri s'exécute jusqu'au premier appel de inversion
2. val1 et val2 sont initialisées avec les valeurs de A et B
3. le sous-programme inversion s'exécute
4. A et B reçoivent les valeurs de val1 et val 2
5. le programme tri s'exécute jusqu'au deuxième appel de inversion
6. val1 et val2 sont initialisées avec les valeurs de B et C
7. le sous-programme inversion s'exécute
8. B et C reçoivent les valeurs de val1 et val 2
9. le programme tri s'exécute jusqu'au troisième appel de inversion
10. val1 et val2 sont initialisées avec les valeurs de A et B
11. le sous-programme inversion s'exécute
12. A et B reçoivent les valeurs de val1 et val 2
13. Le programme tri se termine



Variables paramètres

Action lors du passage de paramètres

- In : les variables sont initialisées par l'appel du sous-programme.
 - Avant l'exécution du sous-programme chaque paramètre réel est copié dans son paramètre formel.
- Out : la valeur des paramètres est retournée au programme appelant.
 - Après l'exécution du sous-programme chaque paramètre formel est copié dans son paramètre réel.

Il est possible de combiner les actions

- In (passage par valeur)
- Out (passage par résultat)
- In/Out (passage par variable ou par référence)

Remarques :

- si le paramètre est déclaré en « in » le paramètre réel peut être un littéral ou une expression.
- si le paramètre est déclaré en « out » ou « in/out » le paramètre réel doit être une variable.

Portée et durée vie des variables

- La portée d'une variable définit quel code à accès à la variable.
- La durée de vie d'une variable définit quand la variable est créée en mémoire et quand elle est détruite en mémoire.
- Une variable peut être « vivante » sans être accessible.
- Les variables locales et les paramètres d'un sous-programme
 - Ont une durée de vie du début de l'appel à la fin de l'appel du sous-programme.
 - Ne sont accessibles que par le code du sous-programme.

Fonction et procédure

- Les fonctions sont des sous programmes qui retournent une valeur.
- Il faut préciser le type de valeur retournée à la fin de la signature.
 - fonction aire(in long :N, in larg :N) :N
- La dernière instruction du code est « RETOURNE » suivie de la valeur à retourner au programme appelant.
 - RETOURNE résultat
- L'appel d'une fonction est remplacé par sa valeur de retour dans l'expression du programme appelant.

Exemple

DEBUT Aire_piece	FONCTION aire(
------------------	----------------

Sous programmes : procédures et fonctions

<pre> VARIABLES LOCALES : VAR ht :N <- 0 VAR lg :N <- 1 VAR total :N <- 0 FIN VARIABLES LOCALES AFFICHER "Entrer la hauteur de la piece" DEMANDER ht AFFICHER "Entrer la longueur du mur (-1 pour finir)" DEMANDER lg TANT QUE lg >= 0 total <- total + aire(ht, lg) AFFICHER "Entrer la longueur du mur (-1 pour finir)" DEMANDER lg FIN TANT QUE AFFICHER total FIN Aire_piece </pre>	<pre> in long:N, in larg:N):N VARIABLES LOCALES : VAR resultat :N <- 0 FIN VARIABLES LOCALES resultat <- long * larg RETOURNE resultat FIN aire </pre>
--	--

Logique de programmation

11. Structure de données : Structure

Définition	54
Syntaxe	54
Exemples de déclarations.....	54
Déclaration de variables d'un type composé	55
Assignation et lecture d'un champ d'une variable.....	55
Exemples d'assignation.	55
Exemple de lecture	55
La pile et le tas.....	55
gestion de la mémoire.....	55
Deux implémentations des Structures	56
Structures implémentées dans la pile	56
Structures implémentées dans le tas	56
Création des instances dans le tas	56
Syntaxe de la création d'une instance:	56

Définition

Une Structure est une définition pour un type de données construit à partir de types primitifs ou de types composés.

Syntaxe

Syntaxe de définition de structure :

```
STRUCTURE nomIdentifiant  
    [1..n] déclarationDeChamps  
FINSTRUCTURE
```

Syntaxe de déclaration de champs:

```
VAR nomIdentifiant : type  
  
Ou  
  
VAR nomIdentifiant [ taille ] : type
```

- Les structures se déclarent en dehors du corps du programme.
- Une fois déclarée, elles peuvent être utilisées dans tous les programmes et sous-programmes.

Exemples de déclarations

```
STRUCTURE Personne  
    VAR nom : T  
    VAR prenom : T  
    VAR naissance : Date  
FINSTRUCTURE  
  
STRUCTURE Date  
    VAR jour : N  
    VAR mois : N  
    VAR an : N  
FINSTRUCTURE
```

Déclaration de variables d'un type composé

VAR client : Personne

VAR dtReunion : Date

La déclaration peut se situer à tout endroit où l'on peut déclarer une variable :

- Comme variable du programme principal.
- Comme paramètres d'un sous-programme.
- Comme variable locales d'un sous-programme.

Assignment et lecture d'un champ d'une variable

Exemples d'assignment.

```
client.nom <- "Dupont"
```

Assignment de la valeur "Dupont" au champ nom (de type Texte) de la variable client (de type Personne)

```
client.naissance.an <- 1993
```



Assignment de la valeur 1993 au champ an (de type numérique) du champ naissance (de type Date) de la variable client (de type Personne)

Exemple de lecture

```
AFFICHER client.nom , " ", client.prenom
```

La pile et le tas

gestion de la mémoire

 La Pile	 Le Tas
Durée de vie	
L'espace mémoire est réservé par la déclaration des variables.	L'espace mémoire est réservé par l'instruction de création (CRÉER).
L'espace mémoire est libéré quand le code ou est déclaré la variable se termine.	L'espace mémoire est libéré quand la référence de l'espace n'est plus accessible depuis la pile.
Accessibilité	
L'espace mémoire est accessible à partir du nom des variables	L'espace mémoire est accessible à partir de la référence de l'espace mémoire.
L'accessibilité au variable dépend de leur type : globale, locale	L'accessibilité à une référence dépend de l'accessibilité à la variable qui stocke la référence

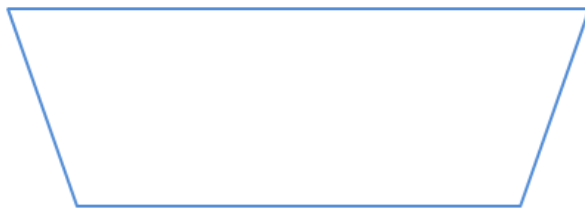
Deux implémentations des Structures

Structures implémentées dans la pile

A la déclaration d'une variable il y a réservation d'autant d'emplacement que nécessaire pour chacun des champs.

La durée de vie des champs est la même que pour la variable déclarée

Nom	Type	Valeur
<u>client.nom</u>	T	"Dupont"
<u>client.prenom</u>	T	"Toto"
<u>client.naissance.jour</u>	N	15
<u>client.naissance.mois</u>	N	9
<u>client.naissance.an</u>	N	1961



```

1. DEBUT encoderClient
2.  VARIABLES LOCALES:
3.    VAR client : Personne
4.  FIN VARIABLES LOCALES
5.    client.nom <- "Dupont"
6.    client.prenom <- "Toto"
7.    client.naissance.jour <- 15
8.    client.naissance.mois <- 9
9.    client.naissance.an <- 1961
10. ...
11. Fin encoderClient
  
```

Structures implémentées dans le tas

Création des instances dans le tas

- La déclaration d'une variable ne réserve qu'une place pour une référence dans la pile.
- L'instruction CRÉER permet de réserver une place pour chaque champ d'une instance de structure dans le tas.
- L'instruction CRÉER retourne la référence de l'endroit où a été créé l'instance.
- La référence doit être stockée dans une variable du type de la structure pour une utilisation future.

Syntaxe de la création d'une instance:

Variable <- CRÉER Type

Nom	Type	Valeur
client	Personne	A

nom	T	"Dupont"
prenom	T	"Toto"
naissance	Date	B

jour	N	15
mois	N	9
an	N	1961

```

1. DEBUT encoderClient
2.  VARIABLES LOCALES :
3.    VAR client : Personne
4.  FIN VARIABLES LOCALES
5.  client <- CRÉER Personne
6.  client.nom <- "Dupont"
7.  client.prenom <- "Toto"
8.  client.naissance <- CRÉER Date
9.  client.naissance.jour <- 15
10. client.naissance.mois <- 9
11. client.naissance.an <- 1961
12. ...
13. Fin encoderClient

```

Comment les instructions ci-dessous transforment-elles les valeurs stockées en mémoire ?

clients[2]:Personne

1	2
A	B

nom	T	"Dupont"
prenom	T	"Paul"
naissance	Date	X

nom	T	"Dupond"
prenom	T	"Toto"
naissance	Date	M

jour	N	15
mois	N	9
an	N	1980

jour	N	5
mois	N	7
an	N	1993

```

...
clients[1].nom <- "Durant"
clients[2].naissance <- clients[1].naissance
clients[1].naissance.jour <- 12
clients[2].naissance.mois <- 4
...

```