INTRODUCTION ALGORITHME
1ère PARTIE

Mr KHATORY
(GIM 1° A)

1

INTRODUCTION ALGORITHME

I. Notion d'Algorithme:

Du mathématicien persan Al-Khawa-Rizm (Bagdad, 780 – 850)
Pour les notions de Al-Jabr (Algèbre) théorie du calcul

Selon le LAROUSSE, la définition d'algorithme est « un ensemble de règles opératoires dont l'enchaînement permet de résoudre un problème au moyen d'un nombre fini d'opérations. »

Quelques points importants :

- Un algorithme décrit un traitement sur un ensemble fini de données de nature simple (nombres ou caractères), ou plus complexes (données structurées)
- Un algorithme est constitué d'un ensemble fini d'actions composées d'opérations ou actions élémentaires. Ces actions élémentaires doivent être effectives (réalisable par la machine), non ambiguës.
- Un algorithme doit toujours se terminer après un nombre fini d'opérations.
- L'expression d'un algorithme nécessite un langage clair (compréhension) structuré (enchaînements d'opérations) non ambiguë, universel (indépendants du langage de programmation choisi)

Problème : un tel langage n'existe pas, on définit son propre langage.

2

INTRODUCTION ALGORITHME

II. Méthodologie de conception d'un algorithme :

Analyse descendante : (ou programmation structurées) : on décompose un problème complexe en sous problèmes et ces sous problèmes en d'autres sous problèmes jusqu'à obtenir des problèmes faciles a résoudre.

On résout les sous problèmes simples sous forme d'algorithme puis on recompose les algorithmes pour obtenir l'algorithme global du problème de départ.

Garder à l'esprit :

- **La modularité** : un module résout un petit problème donné. Un module doit être réutilisable.
- Lisibilité de l'algorithme (mise en page, commentaires, spécification : dire quoi mais pas comment)
- Attention à la complexité de l'algorithme :
  - *Complexité en temps* : mesure du temps d'exécution en fonction de la taille des données
  - *Complexité en espace* : espace mémoire nécessaire pour effectuer les traitements.
- Ne pas réinventer la roue (c'est-à-dire ne pas refaire les programmes standard dont les solutions sont connues) ce qui implique avoir une certaine culture et un outil technique standard.

3

Structure d'un algorithme

ALGORITHME <*NOM*>
<*Déclaration des variables*>
DEBUT
<*Actions*>
FIN

Un algorithme est constitué:

- d'un ***entête*** composé du MOT Réservé **ALGORITHME** et d'un nom de l'algorithme à réaliser
- d'une ***zone de déclaration*** des identificateurs (variables) utilisés dans l'algorithme
- et d'un ***corps délimité*** par deux mots réservés **DEBUT** et **FIN**. C'est ici qu'on écrit les actions de l'algorithme

ALGORITHME addition
VAR a,b,c :ENTIER
DEBUT
LIRE(a,b)
c←a+b
ECRIRE(c)
FIN

4

ELEMENT DE BASE D'ALGORITHME

I. Notion d'objet :

Un algorithme ou une action manipule des données pour obtenir un résultat.
Pour cela on manipule des objets simples ou structurés.

Un objet va être caractérisé par :

- un identificateur (son nom) : pour le désigner cet identificateur doit être parlant : **q**=quotient,**R**=Reste,**Moy**=Moyenne,**ADR**=Adresse….

- Un type (nature de l'objet : entier, caractère…) simple ou structuré. Un type détermine en particulier les valeurs possibles de l'objet et les opérations primitives applicable à l'objet.
R: ***ENTIER***
CAR: ***CARACTERE ; ADR: CHAINE***

- Une valeur (contenu de l'objet) unique. Cette valeur peut varier au cours de l'algorithme ou d'une exécution à l'autre : ces objets sont des variables.

Dans les cas contraires (valeur fixe) ce sont des constantes. Tous les objets manipulés par un algorithme doivent être clairement définis :
Mots clefs

```
CONST     : PI=3.14
VAR  a, b : ENTIER
x, y      : CARACTERE
```

a, b, x, y sont des identificateurs

5

Notion d'objet

1. Constantes et variables

2. Type logique

3. Type CARACTERE

4. Type CHAINE DE CARACTERE

5. Type énumérés

6. Type intervalles

7.Type structurés

8. Type pointeur

Une constante est un objet dont l'état reste inchangé durant toute l'exécution d'un programme

CONST PI=3.14
NOM="PASCAL"

Une variable est un objet dont le contenu peut être modifié par une action

ENTIER: Pour représenter les nombres entiers

REEL : Pour représenter les nombres réels.

Exemples:
VAR Classement : ENTIER
Moyenne : REEL

6

Notion d'objet

1. Constantes et variables
2. ***Type logique***
3. Type CARACTERE
4. Type CHAINE DE CARACTERE
5. Type énumérés
6. Type intervalles
7. Type structurés
8. Type pointeur

Une variable de ***type logique*** (**booléen**) peut prendre deux valeurs VRAIE ou FAUSSE.

VAR EXISTE :BOOLEEN
EXISTE← VRAIE

Les opérations principales les plus utilisées sont :

- ***Les opérateurs logiques***: NON, ET , OU
- ***Opérateurs de comparaison*** : = , ≤ , ≥ , ≠

VAR A , B , TROUVE :BOOLEEN
TROUVE ← (A **ET** B)

VAR X,Y: REEL
SUP :BOOLEEN
SUP ← (X > Y)

7

Notion d'objet

1. Constantes et variables
2. Type logique
3. ***Type CARACTERE***
4. Type CHAINE DE CARACTERE
5. Type énumérés
6. Type intervalles
7. Type structurés
8. Type pointeur

Il s'agit du domaine constitué des c***aractères alphabétiques*** et ***numériques***

VAR C : **CARACTERE**
C ← 'A'

8

Notion d'objet

1. Constantes et variables

2. Type logique

3. Type CARACTERE

4. Type CHAINE DE CARACTERE

5. Type énumérés

6. Type intervalles

7.Type structurés

8. Type pointeur

Une chaine de caractère est un objet qui peut contenir ***plusieurs caractères*** de manière ordonnée

VAR NOM : **CHAINE[30]** → Chaine de 30 caractères ('maximum')
ADRESSE : **CHAINE** → Chaine de 128 caractères maximum

9

Notion d'objet

1. Constantes et variables

2. Type logique

3. Type CARACTERE

4. Type CHAINE DE CARACTERE

5. Type énumérés

6. Type intervalles

7.Type structurés

8. Type pointeur

Un ***type énuméré*** est un type permettant de représenter des objets pouvant prendre leur valeur dans une liste finie et ordonnée de noms.

TYPE SEMAINE=(lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche)
TYPE COULEUR=(rouge, vert, bleu)

10

Notion d'objet

1. Constantes et variables
2. Type logique
3. Type CARACTERE
4. Type CHAINE DE CARACTERE
5. Type énumérés
6. ***Type intervalles***
7. Type structurés
8. Type pointeur

Un ***type intervalle*** est un type dont les objets prennent leur valeur dans une portion de l'intervalle des valeurs d'un autre type (entier, énuméré ou caractère).

Exemple : NBRE=0**..**99
OUVRABLE=lundi**..**vendredi

11

Notion d'objet

1. Constantes et variables
2. Type logique
3. Type CARACTERE
4. Type CHAINE DE CARACTERE
5. Type énumérés
6. Type intervalles
7. ***Type structurés***
8. Type pointeur

Une structure est un objet contenant un ensemble d'objets de types différents, appelés champs. Un type doit donc décrire l'ensemble des champs contenus dans ses objets.

Exemple :
STRUCTURE ETUDIANT
{ NOM : **CHAINE**
NOTE : **REEL**
Classement :**ENTIER**
}

12

Notion d'objet

1. Constantes et variables

2. Type logique

3. Type CARACTERE

4. Type CHAINE DE CARACTERE

5. Type énumérés

6. Type intervalles

7.Type structurés

PLUSTARD !!

8. Type pointeur

13

ELEMENT DE BASE D'ALGORITHME

II. Actions élémentaires :

Actions élémentaires : opérations simple, directement utilisable.

A. Affectation

Permet de donner une valeur à une variable :

A ⇦ 28 « reçoit » Si A avait une valeur auparavant, cette valeur disparaît : elle est écrasée par 28

Format général :
<id_variable> ⇦ <expression> A⇦28+13
Les types <id_variable> et <expression> doivent être compatibles.

Attention : A⇦B+2
B doit avoir une valeur. Or au début d'une action, les variables ont une valeur indéterminée; B doit avoir été initialisé.

14

ELEMENT DE BASE D'ALGORITHME

II. Actions élémentaires :

Actions élémentaires : opérations simple, directement utilisable.

A. Affectation

Permet de donner une valeur à une variable :

A ⇦ 28 « reçoit » Si A avait une valeur auparavant, cette valeur disparaît : elle est écrasée par 28

Format général :
<id_variable> ⇦ <expression> A⇦28+13
Les types <id_variable> et <expression> doivent être compatibles.

Attention : A⇦B+2
B doit avoir une valeur. Or au début d'une action, les variables ont une valeur indéterminée; B doit avoir été initialisé.

B. Les opérations en entrée/ sortie

Elles permettent de récupérer une valeur venant de l'extérieur (lecture) ou de transmettre une valeur à l'extérieur (écriture) .

```
VAR A, B, C : ENTIER
LIRE (A, B)
C⇦A+B
ECRIRE (C)
```

plus rapide ⇨

```
VAR A, B  : ENTIER
LIRE (A, B)
ECRIRE (A+B)
```

15

ELEMENT DE BASE D'ALGORITHME

III.Les structures de contrôle conditionnelles :

Une action décrit un enchaînement d'actions élémentaires. L'enchaînement est décrit par les structures de contrôle.

Une structure de contrôle déjà vue : l'enchaînement séquentiel

```
LIRE (A, B)
C ⇦ 2*A + B
```

La plupart des autres structures de contrôle utilisent la notion de condition (expression booléenne) :

Une condition a une valeur qui est, soit vraie, soit fausse.

Pour déterminer la réalité de cette valeur on utilise :

les opérateurs de comparaisons : =, ≤, ≥, ≠
les opérateurs booléens (logique) : ET, OU, NON

Exemple:

SUP ⇦ (A > B) SUP prend la valeur vraie si A est supérieur à B

16

ELEMENT DE BASE D'ALGORITHME

III.Les structures de contrôle conditionnelles :

A. L'alternative SI-ALORS-SINON

Elle permet d'effectuer tel ou tel traitement en fonction de la valeur d'une condition.

Syntaxe :

SI <condition>
ALORS < action _alors>
FINSI

SI <condition>
ALORS < action _alors>
SINON < action _sinon>
FINSI

Exemple :
LIRE (note)
SI note ≥ 10
ALORS ECRIRE(''Admis'')
SINON ECRIRE(''Ajourné'')
FINSI

Remarque, la ligne *SINON* <*action_sinon*> est facultative.

Principe de fonctionnement :

1 : la condition est évaluée
2 : Si la condition a la valeur vrai on exécute <action_alors>
Si la condition a la valeur fausse on exécute <action_sinon>

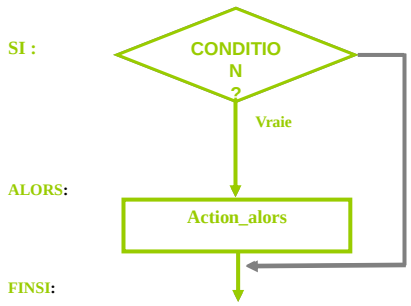
Remarque :

Les <action_alors> ou <action_sinon> peuvent être soit :
- des actions élémentaires
- des composées (bloc)

17

ELEMENT DE BASE D'ALGORITHME

Alternative Simple

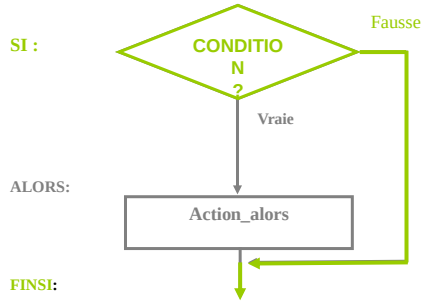


ORGANIGRAMME

18

ELEMENT DE BASE D'ALGORITHME

Alternative Simple

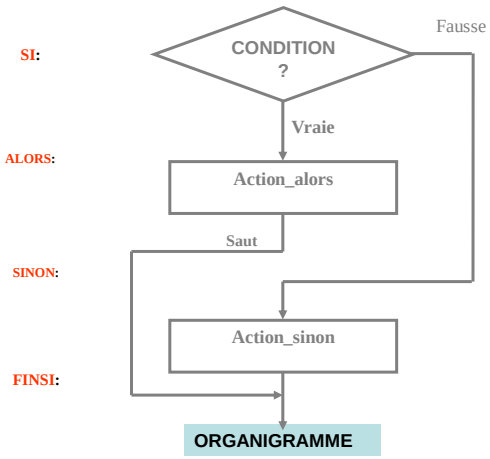


ORGANIGRAMME

19

ELEMENT DE BASE D'ALGORITHME

Alternative Complète

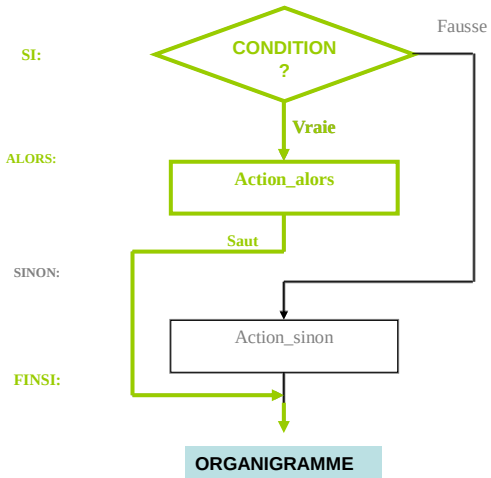


ORGANIGRAMME

20

ELEMENT DE BASE D'ALGORITHME

Alternative Complète

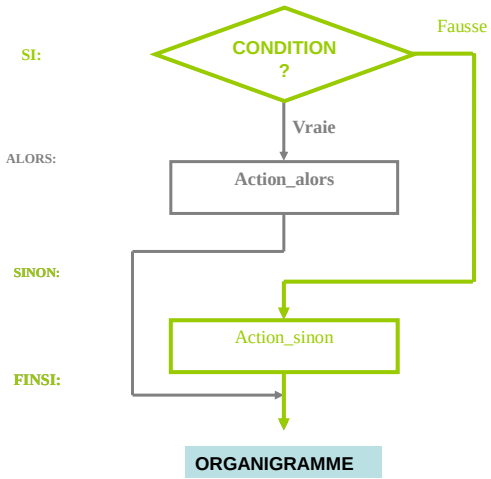


ORGANIGRAMME

21

ELEMENT DE BASE D'ALGORITHME

Alternative Complète

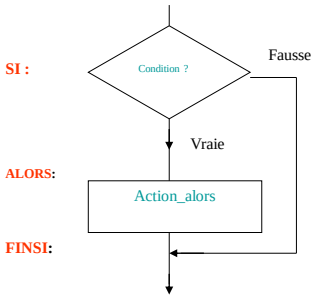


ORGANIGRAMME

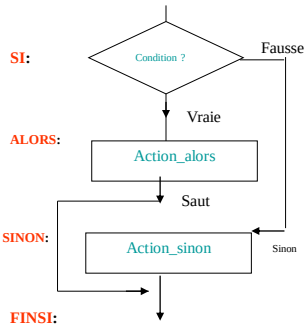
22

ELEMENT DE BASE D'ALGORITHME

Alternative Simple



Alternative Complète



ORGANIGRAMME

23

ELEMENT DE BASE D'ALGORITHME

B. Structure à choix multiples SELON-QUE

Syntaxe :
SELONQUE
<condition 1> : <action 1>
<condition 2> : <action 2>
…
<condition n> : <action n>
SINON : <action_sinon>
FINSELONQUE

Exemple :
SELONQUE
Note ≥ 16 : ECRIRE(''TB'')
Note ≥ 14 : ECRIRE(''B'')
Note ≥ 12 : ECRIRE(''AB'')
Note ≥ 10 : ECRIRE(''Passable'')
SINON : ECRIRE(''ajourné'')
FINSELONQUE

Fonctionnement :

la condition 1 est évaluée :
- Si la condition 1 est vraie, alors on exécute l'action correspondante et on quitte la structure selon-que
- Si la condition 1 est fausse , on évalue la condition 2…et ainsi de suite.
- Si aucune n'est vraie on effectue l'action sinon.

Remarque : en programmation, cette structure peut exister mais avec une forme ou un fonctionnement éventuellement différent. Si elle n'existe pas, il faut se souvenir que, en fait, SELON QUE est un raccourci d'écriture pour des SI imbriqués

24

ELEMENT DE BASE D'ALGORITHME

IV. Structures répétitives :

Idée : répéter un ensemble d'opérations, arrêter la répétition en fonction d'une condition

A. La structure TANT QUE :

Syntaxe :
TANTQUE <condition>
FAIRE <actions>
FINTANTQUE

L'action peut être simple ou composée

Fonctionnement :

1 : la condition est évaluée
2 : si la condition est fausse : c'est fini, on quitte le tant que
3 : si la condition est vraie, on exécute le contenu du tant que puis on remonte à l'étape 1 tester de nouveau la condition

Exemple :
i⇦1
TANTQUE i≤10
FAIRE
ECRIRE (i*i)
i⇦i+1
FINTANTQUE

Afficher les dix premiers carrés

25

ELEMENT DE BASE D'ALGORITHME

IV. Structures répétitives :

B. Structure REPETER JUSQU'A :

Syntaxe :
REPETER
<actions>
JUSQU'A <condition>

Fonctionnement :

1 : on exécute le corps(actions)
2 : on évalue la condition
3 : si Vraie : on quitte le répéter
4 : si Fausse on recommence

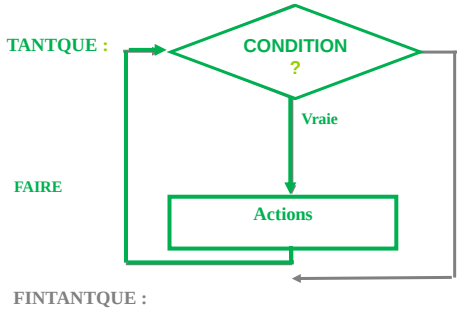
Remarques :

Il y a toujours au moins une exécution du corps. La structure répéter permet de répéter un traitement 1 ou plusieurs fois.

26

ELEMENT DE BASE D'ALGORITHME

BOUCLE TANTQUE

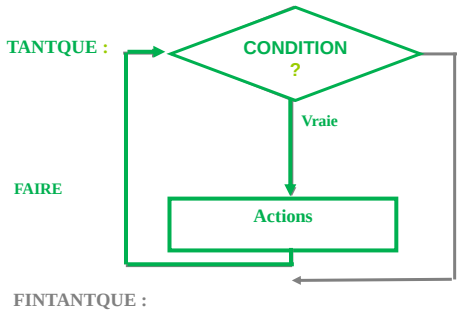


ORGANIGRAMME

27

ELEMENT DE BASE D'ALGORITHME

BOUCLE TANTQUE

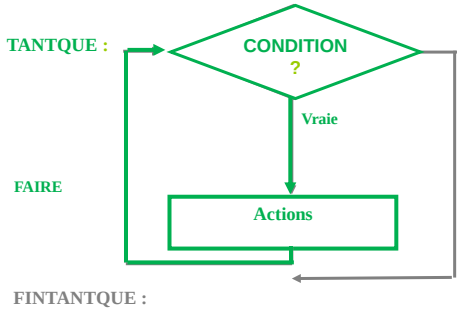


ORGANIGRAMME

28

ELEMENT DE BASE D'ALGORITHME

BOUCLE TANTQUE

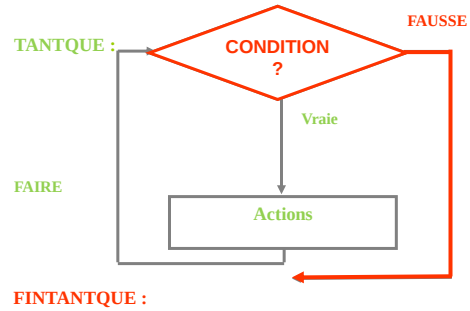


ORGANIGRAMME

29

ELEMENT DE BASE D'ALGORITHME

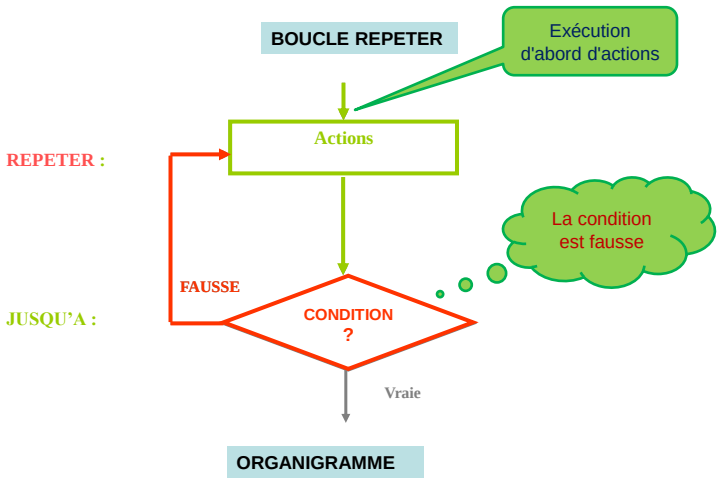
BOUCLE TANTQUE



ORGANIGRAMME

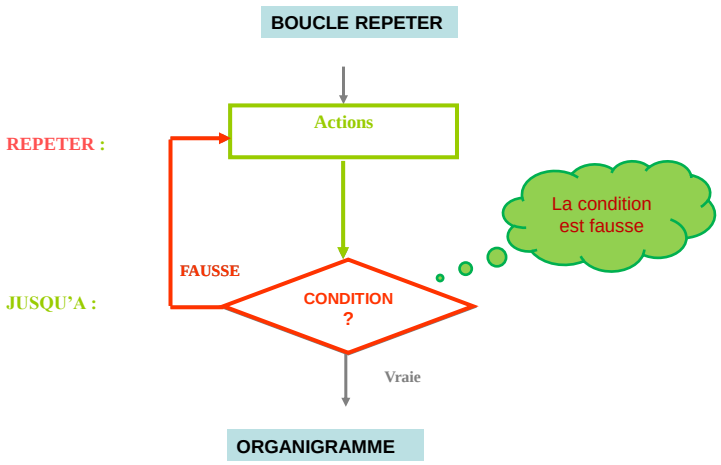
30

ELEMENT DE BASE D'ALGORITHME



31

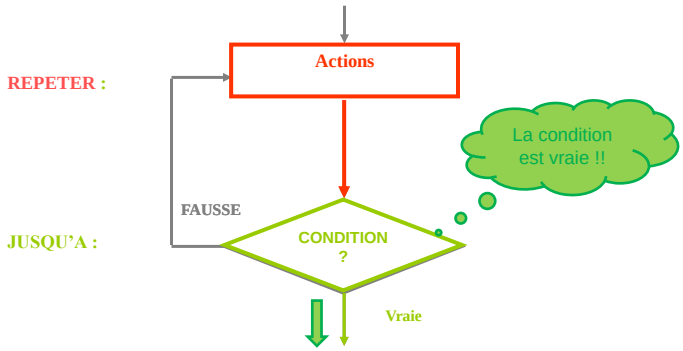
ELEMENT DE BASE D'ALGORITHME



32

ELEMENT DE BASE D'ALGORITHME

BOUCLE REPETER

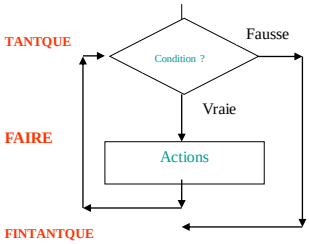


ORGANIGRAMME

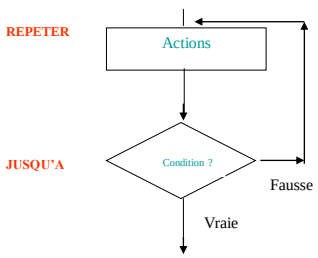
33

ELEMENT DE BASE D'ALGORITHME

Boucle TANTQUE



Boucle REPETER



ORGANIGRAMME

34

ELEMENT DE BASE D'ALGORITHME

IV.Structures répétitives :

C. Structure POUR

Elle permet de parcourir un intervalle en répétant un traitement pour chacune des valeurs de cet intervalle.

Syntaxe :

POUR <id_variable> DE <val_inférieure> A <val_supérieure>
[par pas de <val_pas>] ⇦ facultatif
FAIRE <actions>
FINPOUR

Fonctionnement :

1 : Automatiquement Au départ , on a id_variable ⇦ val_inférieure
Donc, on n'a pas besoin d'initialiser, la structure se charge de la faire
2 : id_variable > val_supérieure ?
Si oui, alors STOP, on quitte la structure
Sinon :
– on exécute le programme
– automatiquement, l'incrémentation se fait (+1 ou +pas si l'on a définit un pas particulier, par défaut, le pas est 1)
– on remonte au début du 2 tester la condition id_variable > val_supérieure ?

35

ELEMENT DE BASE D'ALGORITHME

IV.Structures répétitives :

STRUCTURE POUR

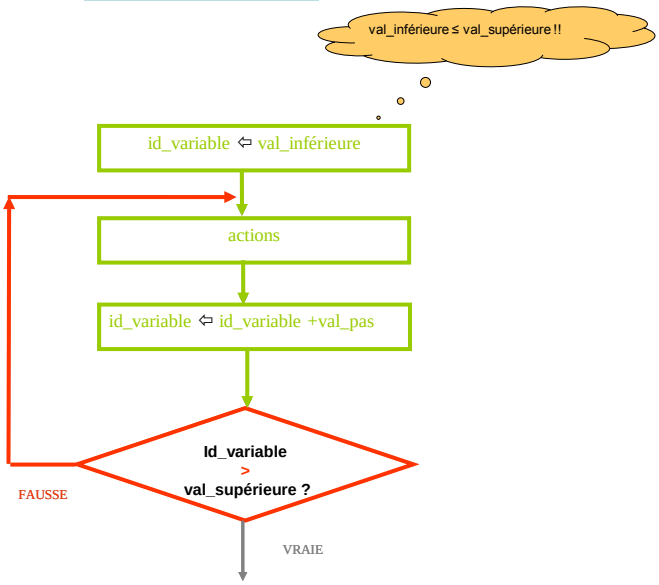
POUR <id_variable>
DE <val_inférieure>
A <val_supérieure>
[par pas de <val_pas>
FAIRE <actions>
FINPOUR

Exemple :Factorielle de n : n!=1*2*3..*(n-1)*n

Fact ⇦ 1
POUR i DE 1 A n
FAIRE fact ⇦ fact*i
FINPOUR

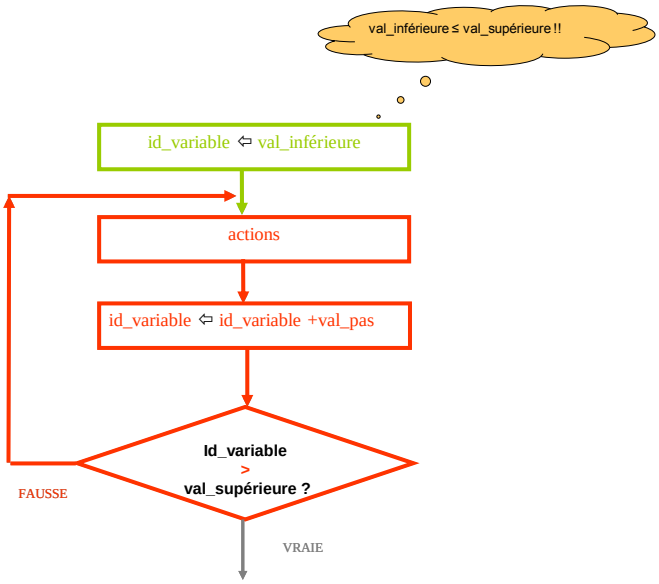
36

BOUCLE POUR

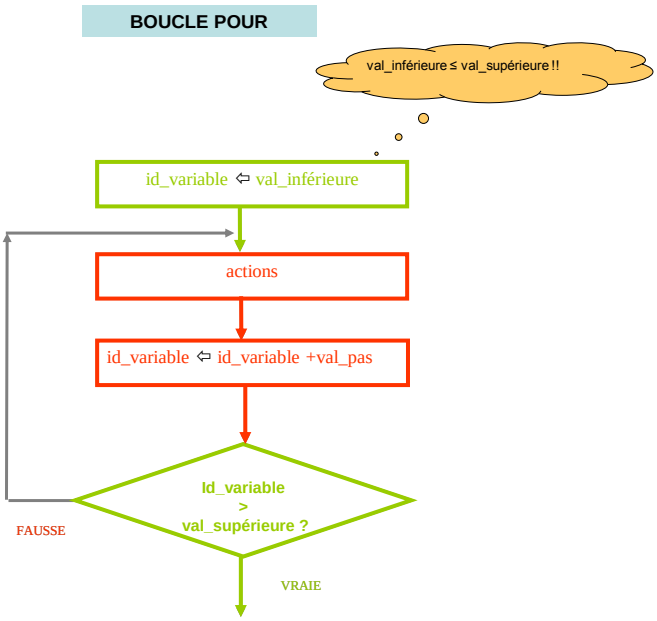


37

BOUCLE POUR



38



39

ELEMENT DE BASE D'ALGORITHME

V.Procedures et Fonctions :

Un algorithme est composé d'un ensemble fini d'actions. En fait, on distingue :

- les **Procedures** (ou ACTIONS) qui réalisent un traitement (tri du fichier étudiant…)
- les **fonctions** qui effectuent un calcul et retournent un résultat

A. PROCEDURE

Syntaxe

PROCEDURE <nom_procedure> <(Liste des paramètres)>
< déclaration des objets locaux de la procedure>
DEBUT
{corps de la procedure}
FIN

Il existe deux types de paramètres transmis:

- par valeur
- par référence

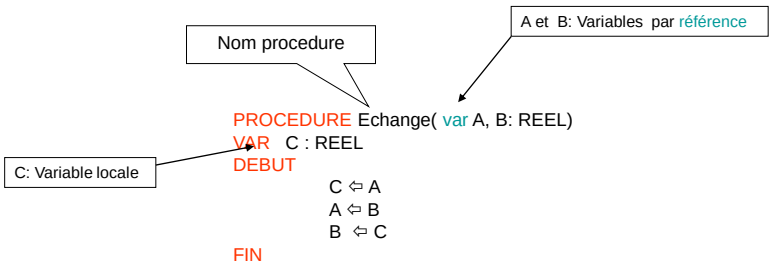
40

ELEMENT DE BASE D'ALGORITHME

V.Procedures et Fonctions :

Exemple d'une Procedure

Ecrire un algorithme d'une procédure qui échange deux variables A et B



41

ELEMENT DE BASE D'ALGORITHME

V.Procedures et Fonctions :

B. FONCTIONS

Syntaxe

FONCTION <nom_fonction> (<liste des paramètres>) : <type de résultat>
< déclaration des objets locaux à la fonction>
DEBUT
{ corps de la fonction}
RETOURNER(resultat)
FIN

À Ne Pas oublier !!

Exemple d'une Fonction

Ecrire une fonction maximum qui donne (**Retourne**) le maximum de deux entiers :

42

ELEMENT DE BASE D'ALGORITHME

V.Procedures et Fonctions :

Exemple:

```
FONCTION maximum (x, y: ENTIER) : ENTIER
VAR   max :ENTIER
DEBUT
      SI     x < y
      ALORS            max ⇦ y
      SINON            max ⇦ x
      FINSI
      RETOURNER (max)
FIN
```

Fonction qui détermine le maximum de deux entiers

```
PROCEDURE minETmax( a,b: ENTIER)
VAR Min,Max:ENTIER
DEBUT
      Min  ⇦ -maximum (-a, -b)
      Max ⇦  maximum (a, b)
      ECRIRE (min, max)
FIN
```

Procedure qui affiche le minimum et le maximum de deux entiers, en faisant appel à la fonction maximum

43

ELEMENT DE BASE D'ALGORITHME

V.Procedures et Fonctions :

Fonction récursive:

La récursivité consiste à remplacer une boucle par un appel à la fonction elle-même.

Exemple :

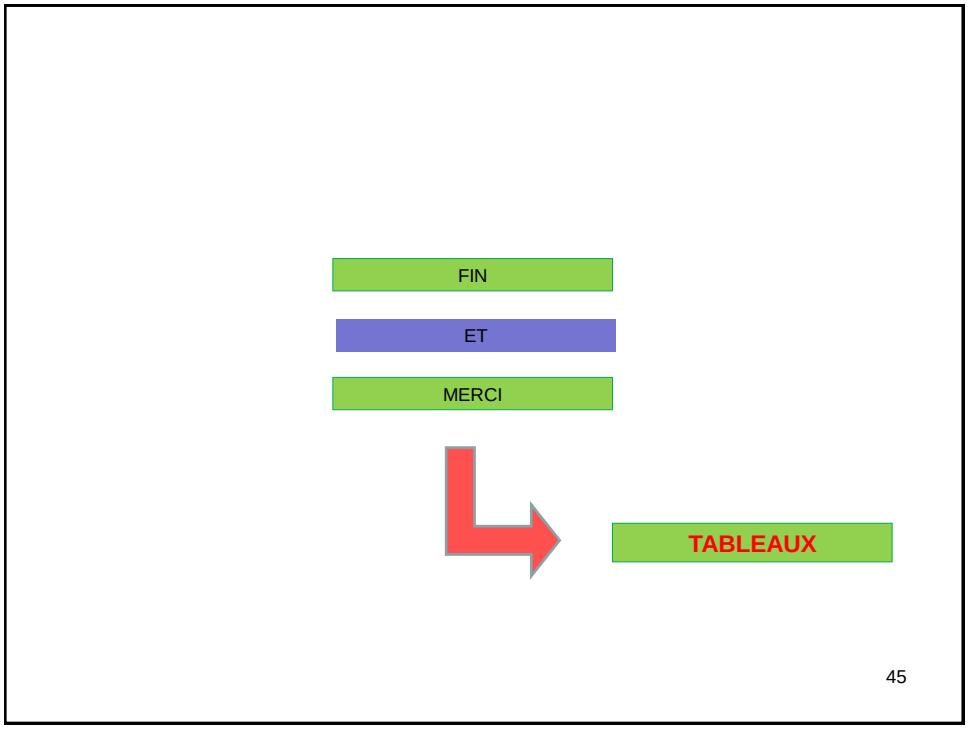
Considérons la suite factorielle, elle est définie par :

0!=1
n! = (n-1)! * n ⇨ factorielle(0)=1
factorielle(n)= factorielle(n-1) *n

La fonction peut s'écrire simplement

```
FONCTION factorielle (n: ENTIER):ENTIER
DEBUT
      SI  (n=0)
      ALORS          retourner(1)
      SINON          retourner( factorielle(n-1) *n)
      FINSI
FIN
```

44


FIN
ET
MERCI
TABLEAUX
45