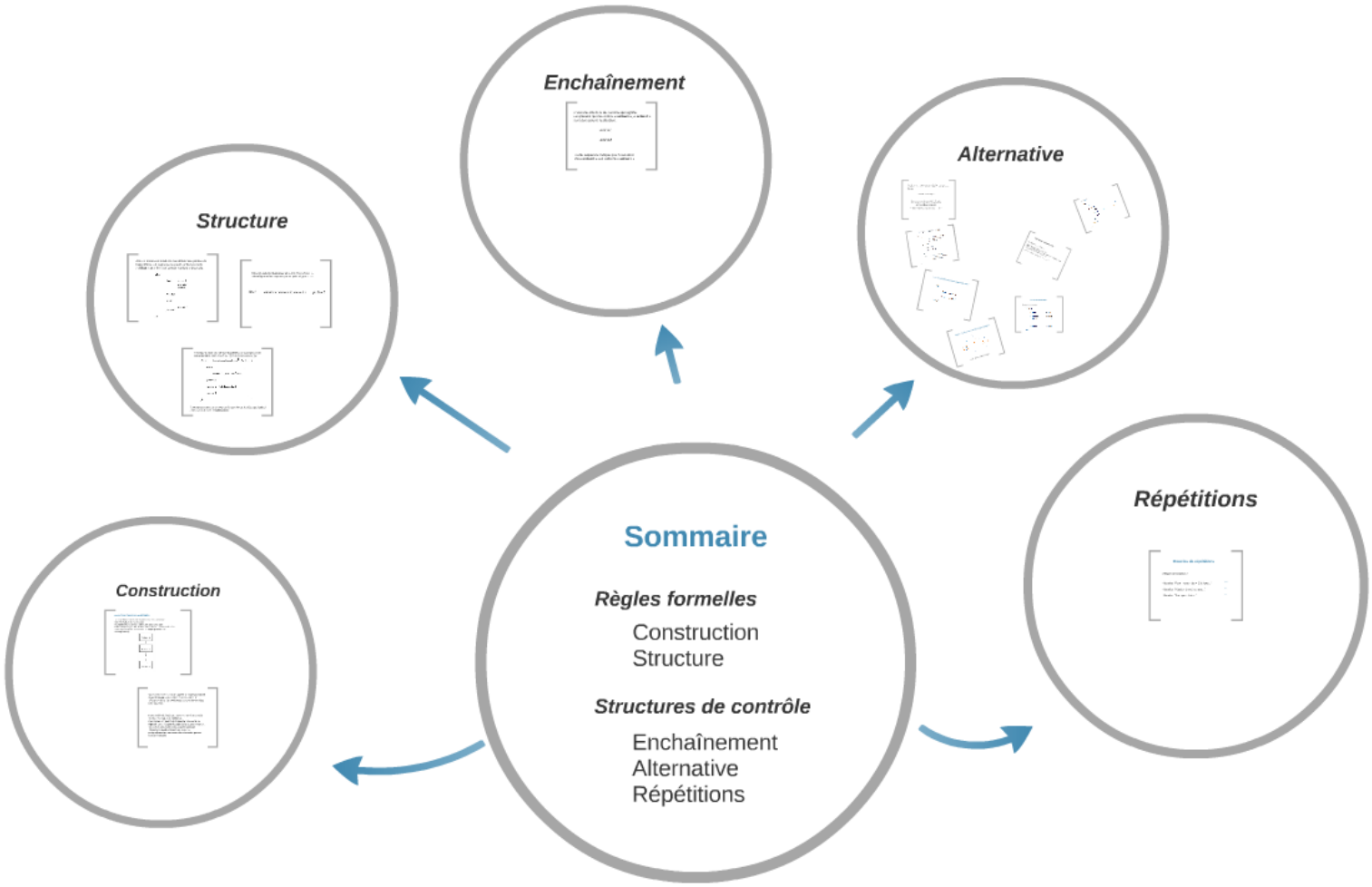
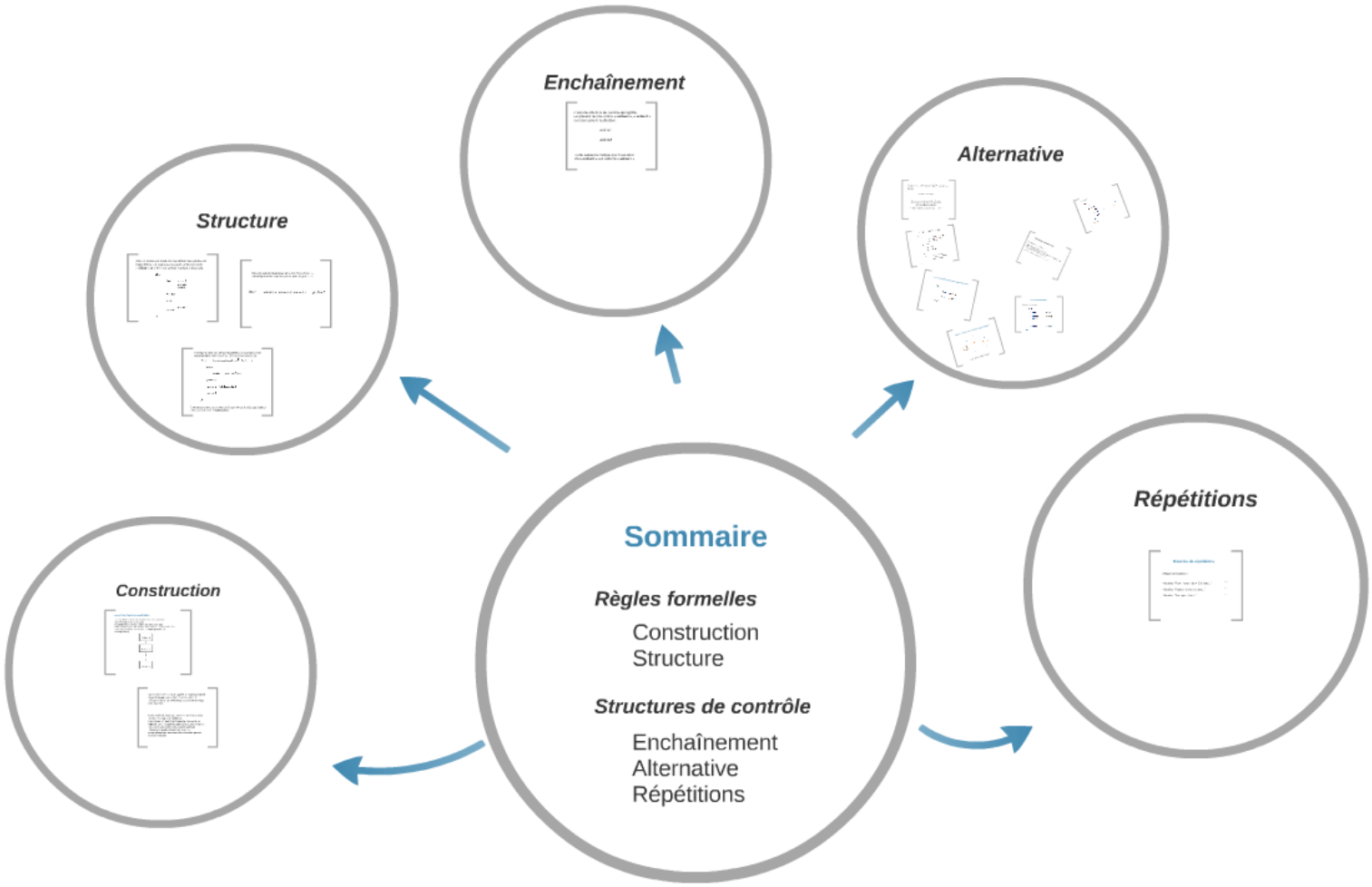


Programmation Algorithmique



Programmation Algorithmique

Sommaire

Règles formelles

Construction
Structure

Structures de contrôle

Enchaînement
Alternative
Répétitions

Construction

CONSTRUCTION D'ALGORITHMES

Un algorithme se présente comme une suite d'actions
que l'ordinateur devra exécuter.
On représente chacune d'elles dans des rectangles
reliés entre eux par des flèches pour indiquer l'ordre d'exécution.
Cette représentation se nomme un **organigramme** (ou
ordinogramme).

Nous utiliserons ce qu'on appelle la **représentation algorithmique** qui consiste à écrire dans un langage naturel les différentes actions demandées à la machine.

Cette méthode implique l'observation d'un certain nombre de **règles d'écritures**.
Ces règles ont pour but d'apporter une certaine **rigueur** dans l'apprentissage de la programmation, mais elles demeurent très souples puisque l'important réside uniquement dans la **compréhension correcte des énoncés par un lecteur humain**.

CONSTRUCTION D'ALGORITHMES

Un algorithme se présente comme une suite d'actions
que l'ordinateur devra exécuter.
On représente chacune d'elles dans des rectangles
reliés entre eux par des flèches pour indiquer l'ordre d'exécution.
Cette représentation se nomme un **organigramme** (ou ordinogramme).

Nous utiliserons ce qu'on appelle la **représentation algorithmique** qui consiste à écrire dans un langage naturel les différentes actions demandées à la machine.

Cette méthode implique l'observation d'un certain nombre de **règles d'écritures**.
Ces règles ont pour but d'apporter une certaine **rigueur** dans l'apprentissage de la programmation, mais elles demeurent très souples puisque l'important réside uniquement dans la **compréhension correcte des énoncés par un lecteur humain**.

Structure

Afin de mettre en évidence les différentes parties de l'algorithme, on regroupe souvent, entre les mots « **début** » et « **fin** » un certain nombre d'énoncés.

```
début
    bloc1     énoncé1
              énoncé 2
              énoncé 3
    fin bloc1
    bloc2
              énoncé 4
    fin bloc2
fin
```

On peut également **grouper** plusieurs énoncés sur la même ligne en les séparant par un point-virgule « ; » :

```
bloc1     énoncé 1 ; énoncé 2 ; énoncé 3     fin bloc1
```

A n'importe quel endroit de l'algorithme, on peut placer un **commentaire** entre crochets « [mon commentaire] » :

```
début     [résolution de l'équation ax² + bx + c = 0]
    calcul
        énoncé 1   [calcul de b² - 4ac]
    fin calcul
    énoncé 2   [calcul des racines]
    énoncé 3
fin
```

Remarquez dans ces exemples d'algorithmes, le décalage textuel connu sous le nom d'**indentation**.

Afin de mettre en évidence les différentes parties de l'algorithme, on regroupe souvent, entre les mots « **début** » et « **fin** » un certain nombre d'énoncés.

début

bloc1 *énoncé1*
énoncé 2
énoncé 3

fin bloc1

bloc2

énoncé 4

fin bloc2

fin

On peut également **grouper** plusieurs énoncés sur la même ligne en les séparant par un point-virgule « ; » :

bloc1 énoncé 1 ; énoncé 2 ; énoncé 3 fin bloc1

A n'importe quel endroit de l'algorithme, on peut placer un **commentaire** entre crochets « [mon commentaire] » :

début [résolution de l'équation $ax^2 + bx + c = 0$]

calcul

énoncé 1 [calcul de $b^2 - 4ac$]

fin calcul

énoncé 2 [calcul des racines]

énoncé 3

fin

Remarquez dans ces exemples d'algorithmes, le décalage textuel connu sous le nom d'**indentation**.

Enchaînement

C'est une structure de contrôle qui signifie simplement que les ordres « **action1** », « **action2** » sont des actions réalisables.

action1

action2

Cette séquence indique que l'exécution de « **action2** » suit celle de « **action1** »

C’est une structure de contrôle qui signifie simplement que les ordres « **action1** », « **action2** » sont des actions réalisables.

action1

action2

Cette séquence indique que l’exécution de « **action2** » suit celle de « **action1** »

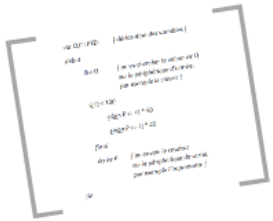
Alternative

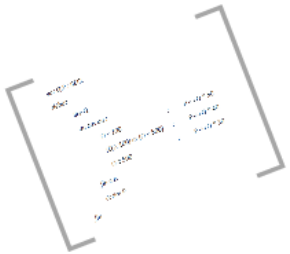

Une alternative en programmation est le fait de décider d'effectuer telle ou telle action en fonction d'une expression logique.

Prenons un exemple :

Une entreprise rend un produit à 50€ l'unité, si la quantité commandée est inférieure à 100 et à 40€ si elle est supérieure.

Pour une quantité Q quel prix P doit-on payer ?



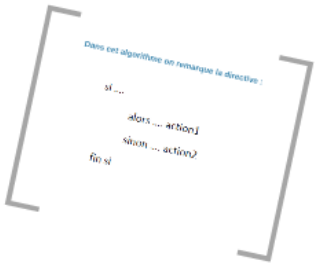



Reprenons notre exemple

Pour rappel du procédé vendu :

SOIT si la quantité est inférieure à 100 ALORS le prix est multiplié par 50 SINON si la quantité est supérieure ou égale à 100 et inférieure à 500 ALORS elle est multipliée par 40 SINON

Pour une quantité Q, quel prix P doit-on payer ?


Dans cet algorithme on remarque la directive :

si ...

alors ... action1

sinon ... action2

fin si


Cas de N alternatives

Directive « au cas où »

condition 1 : action 1

condition 2 : action 2

condition 3 : action 3

- - - - - - -

condition N : action N

fin cas


Représentation sous forme d'organigramme :

Test portant une variable booléenne

Une alternative en programmation est le fait de décider d’effectuer telle ou telle action en fonction d’une expression logique.

Prenons un exemple :

Une entreprise vend un produit à 50€ l'unité,
si la quantité commandée et inférieure à 100
et à 40€ si elle est supérieure.

Pour une quantité Q quel prix P doit-on payer ?

```
var Q,P : REEL       [ déclaration des variables ]

début

    lire Q           [ on va chercher la valeur de Q
                       sur le périphérique d'entrée,
                       par exemple le clavier ]

    si Q < 100

        alors P <= Q * 50

        sinon P <= Q * 40

    fin si

    écrire P         [ on envoie le résultat
                       sur le périphérique de sortie,
                       par exemple l'imprimante ]

fin
```

Dans cet algorithme on remarque la directive :

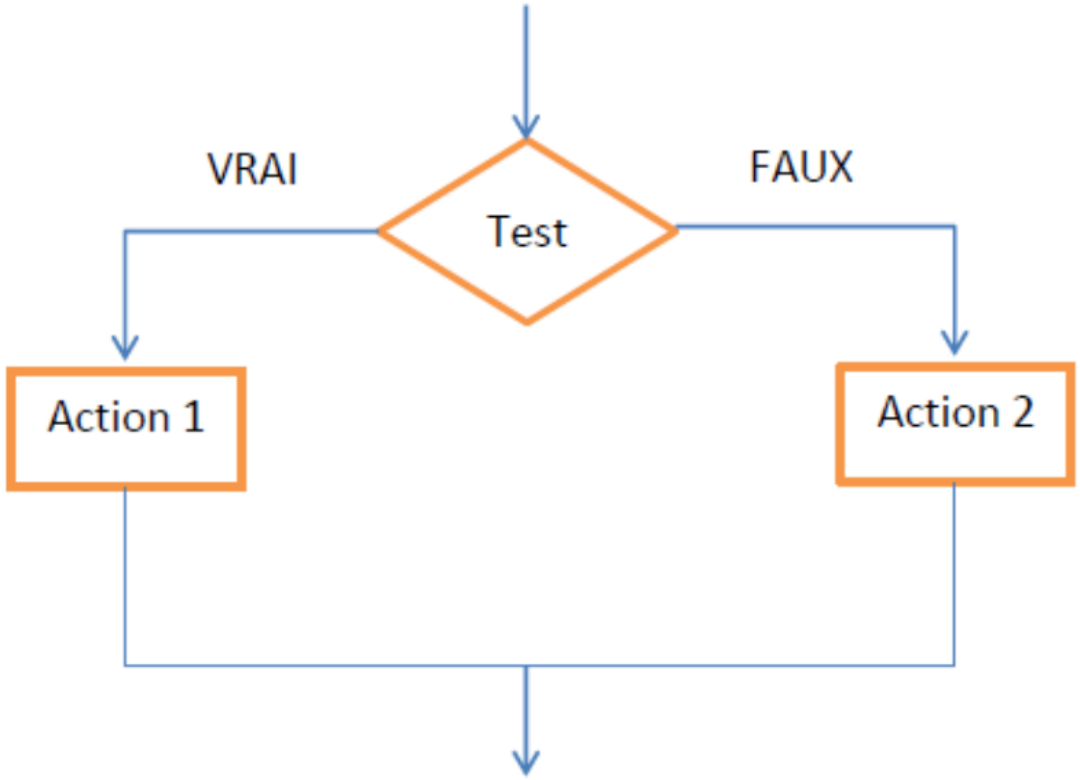
si ....

alors action1

sinon action2

fin si

Représentation sous forme d'organigramme :



Test portant une condition booléenne

Cas de N alternatives

Directive « au cas où »

au cas où :

condition 1 :	action 1
condition 2 :	action 2
condition 3 :	action 3
.....	
condition N :	action N

fin cas

Reprenons notre exemple

Prix unitaire du produit vendu :

50€ si la quantité est inférieure à 100
40€ si la quantité est supérieure ou égale à 100 et inférieure à 500
30€ si elle est supérieure ou égale à 500

Pour une quantité Q quel prix P doit-on payer ?

```
var Q,P : REEL
début
        lire Q
        au cas où :
                Q < 100                 :       P <= Q * 50
                (Q ≥ 100) et (Q < 500)  :       P <= Q * 40
                Q ≥ 500                 :       P <= Q * 30
        fin cas
        écrire P
fin
```

Répétitions

Boucles de répétitions

3 types de boucles :

- Boucles "Pour I variant de A à B, faire..."
- Boucles "Répéter jusqu'à ce que..."
- Boucles "Tant que... faire..."

Boucles de répétitions

3 types de boucles :

- Boucles "Pour I variant de A à B, faire..."

- Boucles "Répéter jusqu'à ce que..."

- Boucles "Tant que... faire..."

Boucles "Pour I variant de A à B, faire..."

Exemple : Calcul de la somme des N premiers entiers.

```
var N, I, SOMME : ENTIER
lire N          [n est supposé différent de 0]
début
    SOMME ⇐ 0   [On initialise la variable SOMME à 0]
        Pour I variant de 1 à N faire SOMME ⇐ SOMME + I
        fin pour
    Ecrire SOMME
fin
```

Boucles "Répéter jusqu'à ce que..."

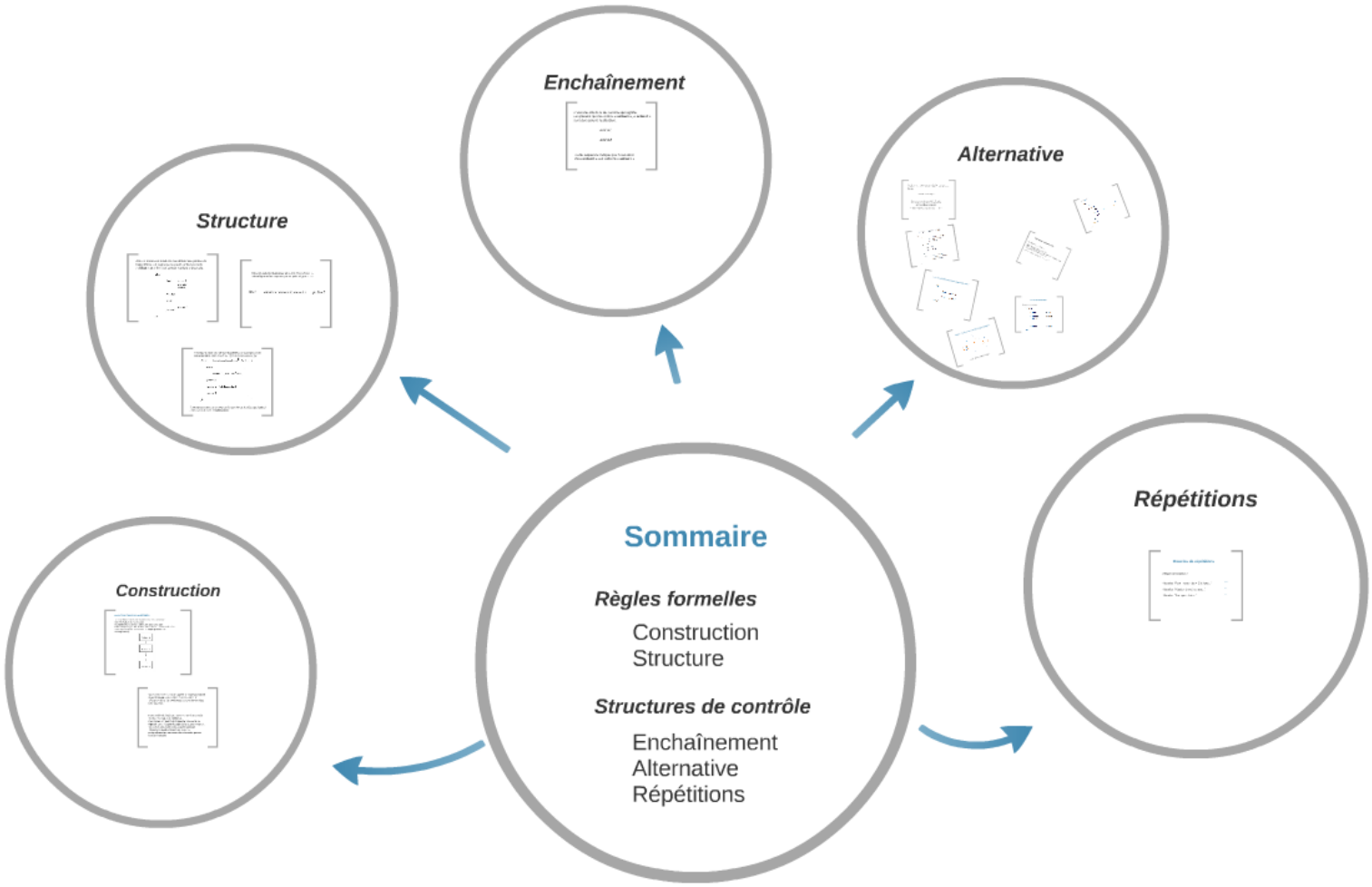
```
var N, I, SOMME : ENTIER
début
    lire N
    I ⇐ 0 ; SOMME ⇐ 0
    répéter I ⇐ I + 1
        SOMME ⇐ SOMME + I
    jusqu'à I = N
    écrire SOMME
fin
```

On effectue le test de la condition à la fin de chaque boucle

Boucles "Tant que... faire..."

```
var N, I, SOMME : ENTIER

I ⇐ 0 ; SOMME ⇐ 0

début

    lire N

    tant que I < N faire

        I ⇐ I + 1
        SOMME ⇐ SOMME + I

    fin tant que

    écrire SOMME

fin
```

On effectue le test de la condition au début de chaque boucle



Programmation Algorithmique