Algorithmique et Structures de Données1
CH2: Les structures conditionnelles et itératives

Enseignant: Fethi Mguis
Sections: LGLSI1/LIRIS1

A.U: 2023/2024

Objectifs

1. Construire des algorithmes comportant des traitements conditionnels
2. Construire des algorithmes comportant des traitements itératifs

1 Les structures conditionnelles

En programmation, on est souvent confronté à des situations où on a besoin de choisir entre 2 ou plusieurs traitements selon la réalisation ou non d'une certaine condition ; d'où la notion de traitement conditionnel.
On distingue 2 structures de traitement conditionnel à savoir :

- La structure de sélection simple : dans laquelle on a à choisir entre 2 traitements au plus.
- La structure de sélection multiple : dans laquelle on a la possibilité de choisir un traitement parmi plusieurs.

1.1 Structure de sélection simple

1.1.1 Forme simple

Syntaxe

```
Si (Condition) Alors
| Séquence d'instructions 1
Fin Si
```

Cette primitive a pour effet d'exécuter la "séquence d'instructions" si et seulement si la condition est vérifiée. L'exécution de cette instruction se déroule selon l'organigramme suivant :

Exemple 1 : Ecrire un algorithme qui permet, à partir de la saisie d'un nombre, d'afficher un message pour indiquer l'impossibilité de l'utiliser comme diviseur s'il est égal à 0.

```
Algorithme DIVISEUR_ZÉRO
Var
   nb : Entier
Début
| Ecrire("Donner un entier :")
| Lire(nb)
| Si (nb=0) Alors
| | Écrire(nb,"ne peut pas être un diviseur")
| Fin Si
Fin
```

Exemple 2 Ecrire un algorithme calcule le salaire d'un salaire d'un employé à partir du nombre d'heures travaillés, du taux horaire et du nombre d'années de services. Les employés ayant une ancienneté de plus

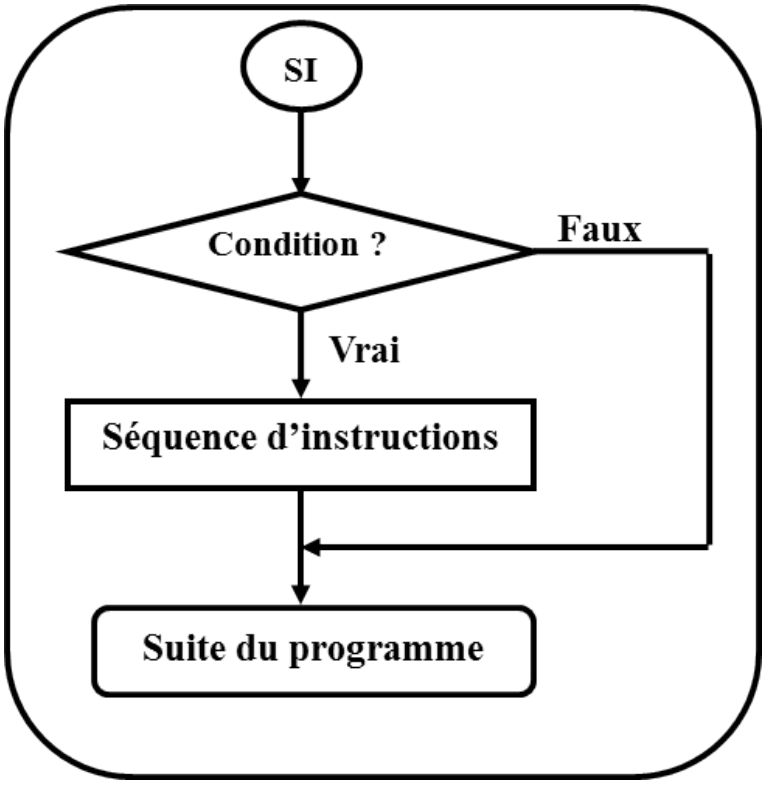

FIGURE 1 – Schéma d'exécution d'une instruction conditionnelle simple.

de 10 ans bénéficient d'une allocation supplémentaire de 45D.

```
Algorithme SALAIRE
Var
   nh, th, anc, salaire : réel
Début
 Écrire("Donner le nombre d'heures travaillées :")
 Lire(nh)
 Écrire("Donner le taux horaire :")
 Lire(th)
 Écrire("Donner l'ancienneté :")
 Lire(anc)
 salaire←nh * th
 Si (anc>10) Alors
 | salaire←salaire + 45
 Fin Si
 Ecrire("salaire de l'employé =", salaire)
Fin
```

1.1.2 Forme alternative

Syntaxe

```
Si (Condition) Alors
| Séquence d'instructions 1
Sinon
| Séquence d'instructions 2
Fin Si
```

- "Condition" est une expression à valeur logique.
- "Séquence d'instructions 1" et "Séquence d'instructions 2" sont des séquences d'actions élémentaires. Ces deux traitements sont exclusifs, c'est à dire soit l'un ou l'autre qui sera exécuté mais jamais les deux à la fois.
- L'exécution de cette instruction se déroule suivant l'organigramme suivant :

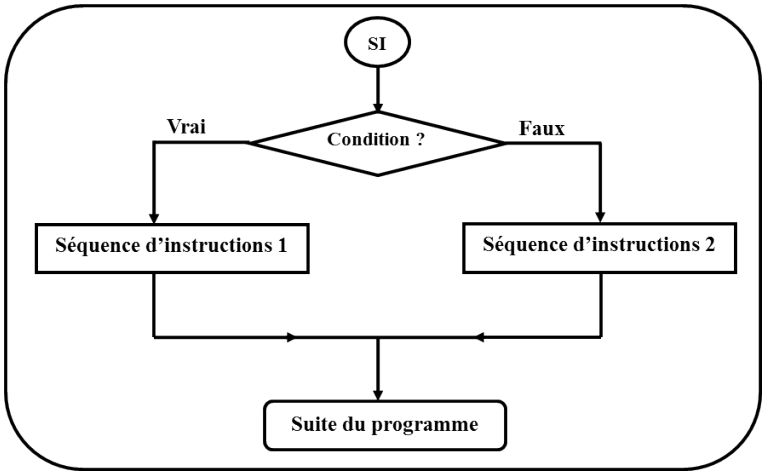

FIGURE 2 – Schéma d'exécution d'une instruction conditionnelle alternative.

Exemple1 Ecrire un algorithme qui lit un entier et affiche ensuite s'il est pair ou pas.

```
Algorithme PARITÉ
Var
   NB : entier
Début
  Écrire("Donner un nombre entier :")
  Lire(NB)
  Si (NB mod 2 = 0) Alors
    Écrire(NB, "est pair")
  Sinon
    Écrire(NB, "est impair")
  Fin Si
Fin
```

Exemple 2 Ecrire un algorithme qui calcule et affiche la valeur absolue d'un entier quelconque lu au clavier.

```
Algorithme VAL_ABS
Var
   x, va : entier
Début
  Écrire("Donner un nombre entier :")
  Lire(x)
  Si (x>0) Alors
    va←x
  Sinon
    va←x
  Fin Si
  Écrire("|", x, "| = ", va)
Fin
```

1.1.3 Forme imbriquée

Syntaxe

```
Si (Condition 1) Alors
| Séquence d'instructions 1
Sinon
| Si (Condition 2) Alors
| | Séquence d'instructions 2
| Sinon
| | ...
| | Si (Condition n) Alors
| | | Séquence d'instructions n
| | Sinon
| | | Séquence d'instructions n+1
| | Fin Si
| | ...
| |
| Fin Si
Fin Si
```

Lorsque l'évaluation de la "Condition 1" produit la valeur :

- "VRAI" seul le "traitement 1" est exécuté
- "FAUX" on passe à l'évaluation de la "condition 2", si elle produit la valeur :
  - "VRAI" seul le "traitement 2" est exécuté
  - "FAUX" on passe à l'évaluation de la "condition 3" et ainsi de suite.

Si aucune des N conditions ne produit la valeur "VRAI", par conséquent le "traitement n+1" est exécuté.

Exemple Ecrire un algorithme qui permet de saisir deux entiers A et B, teste si A est supérieur, inférieur ou égale à B puis afficher le Résultat.

```
Algorithme TEST
Var
   A, B : entier
Début
| Écrire("Donner un entier :")
| Lire(A)
| Écrire("Donner un autre entier :")
| Lire(B)
| Si (A>B) Alors
| | Écrire (A, " est supérieur à ", B)
| Sinon
| | Si (A<B) Alors
| | | Écrire(A, " est inférieur à ", B)
| | Sinon
| | | Écrire(A, " est égal à ", B)
| | Fin Si
| Fin Si
Fin
```

1.2 Structure de sélection multiple

Syntaxe

```
Selon(Sélecteur) faire
| Liste_valeurs_1 : Séquence_d'instructions_1
| Liste_valeurs_2 : Séquence_d'instructions_2
| ...
| Liste_valeurs_n : Séquence_d'instructions_n
| Sinon : Séquence_d'instructions_n+1
Fin Selon
```

- "Sélecteur" est un identificateur (Nom).
- "Traitement_i" est une séquence d'actions élémentaires.

- "Liste_de_valeurs_i" peut être donnée sous forme de constante ou d'intervalle de constantes de même type avec la variable du paramètre "Sélecteur".
- La partie SINON est facultative.
- Si aucune des égalités entre le sélecteur et une valeur parmi les listes des valeurs n'est trouvée, et si la partie "Sinon" existe, le traitement correspondant à cette partie "Sinon" est exécuté tandis que si la partie "Sinon" n'existe pas, alors l'exécution se poursuit à l'instruction immédiatement après le "Fin selon".

Exemple 1 Ecrire un algorithme qui permet de saisir un numéro du jour de la semaine (entre 1 et 7) et d'afficher le nom de ce jour en toute lettre.

```
Algorithme SEMAINE
Var
   NJ : entier
Début
  Écrire("Donner un entier :")
  Lire(NJ)
  Selon(NJ) faire
    1 : Écrire("Lundi")
    2 : Écrire("Mardi")
    3 : Écrire(" Mercredi")
    4 : Écrire("Jeudi")
    5 : Écrire("Vendredi")
    6 : Écrire("Samedi" )
    7 : Écrire("Dimanche")
    Sinon : Écrire("Numéro du jour incorrect")
  Fin Selon
Fin
```

Exemple 2 Ecrire un algorithme qui permet de saisir un nombre entre 0 et 9 et d'afficher la nature de ce nombre (zéro, pair, impair).

```
Algorithme SEMAINE
Var
   NB : entier
Début
  Écrire("Donner un entier :")
  Lire (NB)
  Selon(NB) faire
    0 : Écrire("Chiffre zéro")
    1, 3, 5, 7, 9 : Écrire("Chiffre impair")
    2, 4, 6, 8 : Écrire("Chiffre pair")
    Sinon : Écrire("Chiffre erroné")
  Fin Selon
Fin
```

2 Les structures itératives

Il est souvent nécessaire d'exécuter plusieurs fois une action ou un groupe d'actions, non pas indéfiniment mais un certain nombre de fois(pas forcement connu à l'avance) : c'est la notion de boucle.

2.1 La structure "Pour"

Syntaxe

```
Pour vc de vi à vf faire
  Traitement
Fin Pour
```

Interprétation

- vc : compteur de type entier.
- vi et vf : sont respectivement valeur initiale et valeur finale de vc.
- "Traitement" : action ou séquence d'actions à répéter (Vf - Vi + 1) fois.
- La boucle "Pour" est utilisée lorsque le nombre de répétition du traitement est connu a l'avance.
- Le paramètre compteur (vc) reçoit une valeur initiale de type entier au moment de l'arrivée, pour la première fois, à la boucle "Pour". Il ne doit pas être modifié par une action de traitement à l'intérieur de la boucle.
- Le compteur est incrémenté automatiquement (augmenté de 1) à chaque exécution du corps de la boucle "Pour". Cette valeur d'incrémentation est souvent appelée le pas de la boucle.
- L'exécution s'arrête quand le compteur atteint la valeur finale de la boucle.

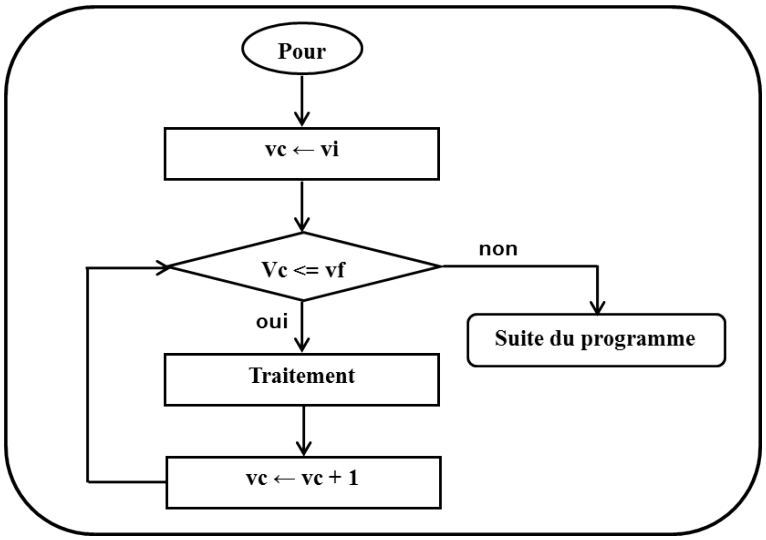

FIGURE 3 – Schéma d'exécution d'une boucle "Pour".

Exemple

```
Pour i de 1 à 5 faire
| Écrire( i * 100 )
Fin Pour
```

Cette boucle affiche respectivement les nombres 100, 200, 300, 400, 500

Remarques

1. Une boucle "pour" peut être exécutée 1 ou n fois.
2. Si le pas de la boucle "pour" est différent de 1, dans ce cas il faut ajouter l'option "pas = constante"

Exemples

```
Pour i de 5 à 1 (pas=-1) faire
| Écrire( i * 100 )
Fin Pour
```

Dans ce cas le compteur i sera décrémenter de 1 après chaque parcours. Cette boucle affiche respectivement les nombres 500, 400, 300, 200, 100

```
Pour i de 1 à 10 (pas=2) faire
| Écrire( i * 100 )
Fin Pour
```

Dans ce cas le compteur i sera incrémenter de 2 après chaque parcours. Cette boucle affiche respectivement les nombres 100, 300, 500, 700, 900

Exercice 1 Ecrire un algorithme qui permet de calculer et d'afficher la somme des nb (saisi à partir du clavier) premiers entiers naturels.

```
Algorithme SOMME
Var
   nb, i, som : Entier
Début
| Ecrire("Donner un entier :")
| Lire(nb)
| som←0
| Pour i de 1 à nb faire
| | som←som + i
| Fin Pour
| Écrire("La somme des", nb, "premiers nombres est", som)
Fin
```

Exercice Ecrire un algorithme qui lit un entier n qu'on le suppose positif puis afficher tous ses diviseurs.

```
Algorithme DIVISEURS
Var
   n, i : Entier
Début
| Ecrire("Donner un entier :")
| Lire(n)
| Pour i de 1 à n faire
| | Si (n MOD i = 0) Alors
| | | Écrire( i )
| | Fin Si
| Fin Pour
Fin
```

2.2 La structure "Répéter ... Jusqu'à"

Syntaxe

```
Répéter
| Traitement
Jusqu'à (Condition)
```

Interprétation

- "Condition" : condition d'arrêt et de sortie de la boucle.
- "Traitement" : action ou séquence d'actions à exécuter tant que la condition n'est pas vérifiée.
- Le traitement est exécuté tant que la condition n'est pas vérifiée, dés que celle-ci est vérifiée, l'exécution du traitement correspondant à la boucle s'arrête.
- Le nombre de répétition obtenu avec cette boucle n'est pas connu à l'avance.
- Le traitement de la boucle "Répéter" est exécuté au moins une seule fois quelque soit le résultat de la condition.
- La condition de contrôle de la boucle "Répéter" doit être initialisée avant la boucle (autrement la boucle ne peut pas être exécutée), et à l'intérieur de la boucle cette même condition doit être modifiée (autrement on aura une boucle infinie).

Exemple 1

```
i←1
Répéter
| Écrire( i * 100 )
| i←i + 1
Jusqu'à (i ¿ 5)
```

Cette boucle affiche respectivement les nombres 100, 200, 300, 400, 500

Exercice Ecrire un algorithme qui permet de calculer et d'afficher la somme des nb (saisi à partir du clavier) premiers entiers naturels. (Remplacer la boucle "Pour" par une boucle "Répéter")

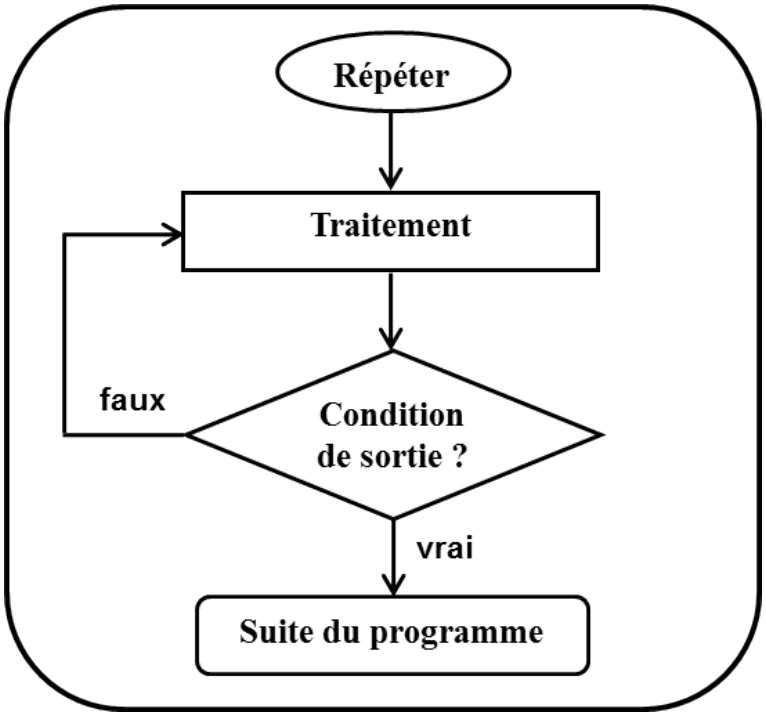

FIGURE 4 – Schéma d'exécution d'une boucle "Répéter".

```
Algorithme SOMME
Var
   nb, i, som : Entier
Début
 Ecrire("Donner un entier :")
 Lire(nb)
 som←0
 i←1
 Répéter
 | som←som + i
 | i←i + 1
 Jusqu'à (i ¿, nb)
 Écrire("La somme des", nb, "premiers nombres est", som)
Fin
```

2.3 La structure "Tant Que"

Syntaxe

```
Tant que (Condition) faire
| Traitement
Fin Tq
```

Interprétation

- "Condition" : est une expression logique déterminant la condition de maintien de la boucle.
- "Traitement" : action ou séquence d'actions à exécuter tant que la condition est vérifiée.
- Le traitement est exécuté tant que la condition est vérifiée, dés que celle-ci cesse d'être vérifiée, l'exécution du traitement correspondant à la boucle s'arrête.
- La condition ne peut pas être vérifiée dés la première exécution d'une répétitive, dans ce cas le traitement ne sera jamais exécuté (0 fois).
- Le nombre de répétition obtenu avec la boucle "Tant Que" n'est pas connu à l'avance.
- La condition de contrôle de la boucle "Tant Que" doit être initialisée avant la boucle(autrement la boucle ne peut pas être exécutée), et à l'intérieur de la boucle cette même condition doit être modifiée (autrement on aura une boucle infinie).

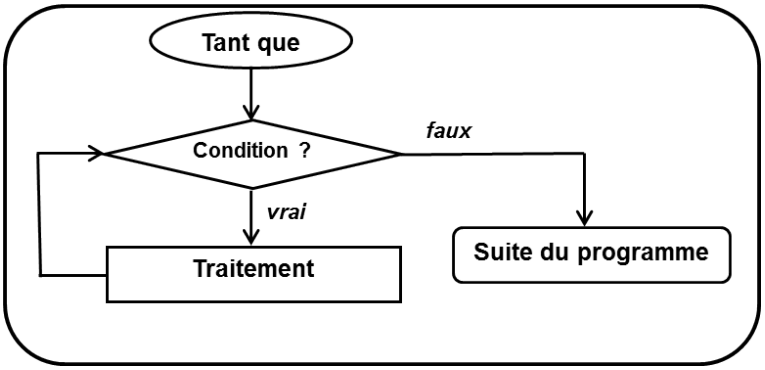

FIGURE 5 – Schéma d'exécution d'une boucle "TantQue".

Exemple

```
i←1
Tant que (i<5) faire
| Écrire ( i * 100 )
| i←i + 1
Fin Tq
```

Cette boucle affiche respectivement les nombres 100, 200, 300, 400, 500

Exercice Ecrire un algorithme qui lit un entier positif n puis affiche tous ses diviseurs.

```
Algorithme DIVISEURS
Var
   n, i : Entier
Début
 Répéter
 | Ecrire("Donner un entier :")
 | Lire(n)
 Jusqu'à (n>0)
 i←1
 Tant que (i<=n) faire
 | Si (n MOD i = 0) Alors
 | | Écrire ( i )
 | Fin Si
 | i←i + 1
 Fin Tq
Fin
```

2.4 Choix de la structure itérative

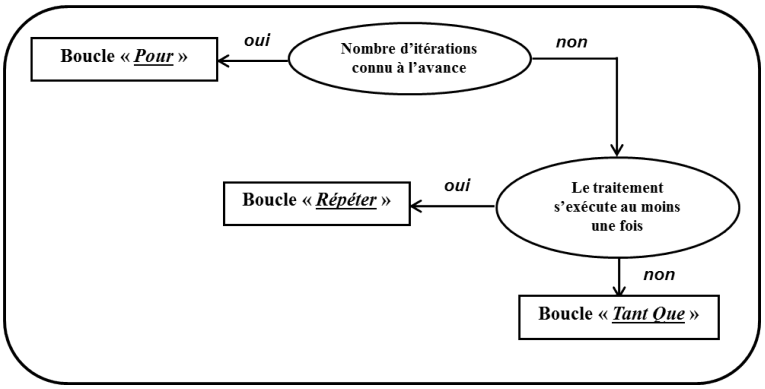

FIGURE 6 – Critères de choix de la structure itérative adéquate.