1ière Année Licence Informatique – ASD1 – S1
Chapitre 4 : Les boucles (en langage algorithmique et en langage C)

Contenu du Chapitre :

1. Introduction

Les ***boucles*** en programmation sont nécessaires pour **effectuer des actions plusieurs fois**. Il existe essentiellement **trois formes de boucles** : La boucle Pour…, la boucle Tantque… et la boucle Répéter…

Nous rappelons ici que, comme les structures d'alternatives, les structures de boucles provoquent des ruptures dans la séquence naturelle d'exécution des programmes. La différence est que, là où les structures d'alternative provoquent la séquence naturelle par des sauts en avant (CO ← CO + déplacement), les structures de boucles, elles, provoquent des sauts en arrière (CO ← CO – déplacement).

Les définitions des 3 structures algorithmiques de boucles : La boucle Pour…, la boucle Tantque… et la boucle Répéter…, sont alors comme suit :

4.2 La boucle Pour…Finpour (C : for…)

Cette structure de contrôle algorithmique est utilisée lorsqu'on veut répéter des actions un certain nombre de fois, en se basant sur un *compteur* (exemple : i : entier), scannant un intervalle de valeurs entières. Il existe alors plusieurs forme pour cette structure algorithmique. Nous proposons dans ce cours quelques versions des plus répandues, en commençant par les versions les plus simples et les plus utiles :

Version 1 : Dans cette version, l'intervalle v1 … v2 est tel que $v1 \leq v2$. De plus, l'intervalle est parcouru de la valeur minimale v1 à la valeur maximale v2, en incrémentant le compteur vari de la boucle pour par 1 (pas = 1).

1ière Année Licence Informatique – ASD1 – S1
Chapitre 4 : Les boucles (en langage algorithmique et en langage C)

Syntaxe algorithmique de la version 1 de la boucle pour :

Pour vari = v1 jusqu'à v2 faire
Action1
Action2
….
Actionk
Finpour

Où, vari : est une variable de type *entier* (en général) : le compteur de la boucle.

v1 : valeur initiale du compteur.

v2 : valeur finale du compteur.

La valeur d'incrémentation du compteur de boucles vari est 1

Condition : Nous devons avoir : v1 ≤ v2.

Effet : Les actions entre ***faire*** et ***finpour*** sont exécutées avec les valeurs de vari :

v1, v1 + 1, v1 + 2, …, v2

Remarque :

L'incrémentation de vari est automatique.

Il ne faut pas alors tenter de modifier vari à l'intérieur de la boucle **For (par lecture ou par affectation)**.

En cas de modification, la logique de la boucle change complètement.

Exemples :

a. Version 1 : (a) Saisir un nombre entier n et (b) calculez la somme des nombres 1, 2, 3, …, n. Nous demandons d'afficher les résultats partiels : numéro de boucle et somme partielle correspondante .

En langage algorithmique : Version

```
Algorithme SomArithmetiqueFor
Donnée  n : entier
Résultats  som : entier              //  Numéro de boucle
Variables i : entier
Début
   Lire(n)
i ←0
Pour i=1 jusqu'à n faire
    som←som+i
Ecrire(" Boucle :",i, " Somme =",som
finpour
Ecrire("Somme 1+2+++…+n = ",som)
Fin SomArithm
```

Syntaxe en langage C :

```
for (vari=v1 ; vari<= v2 ; vari++) {   // vari++  → vari = vari +1;
    Instr1 ;
    Instr2 ;
    ….
    InstrK ;
}
```

Effet : identique à la version algorithmique.

Exemple en langage C : Voici le programme C équivalent à l'algorithme précédent :

```
#include <stdio.h>
int main(){
int i,n,som;
printf("Valeur de n : ");
scanf("%d",&n) ;
som=0;
for (i=1;i<=n; i++){
    som += i;          // som = som + i ;
    printf(" \n Boucle : %d  Somme = %d",i,som);
}
printf("\n Somme de 1 a  %d : ",n);
printf(" est : %d", som);
}
```

Exécution :

```
Valeur de n : 6
 Boucle : 1  Somme = 1
 Boucle : 2  Somme = 3
 Boucle : 3  Somme = 6
 Boucle : 4  Somme = 10
 Boucle : 5  Somme = 15
 Boucle : 6  Somme = 21
 Somme de 1 a  6 :  est : 21
```

Version 2 :

Dans une autre version, nous pouvons avoir un intervalle v1 … v2, avec v1≥ v2. L'intervalle sera toujours parcouru de gauche à droite :

- valeur initiale du compteur vari : v1 ,
- valeur finale v2,
- en décrémentant la valeur du compteur de 1 dans chaque boucle.

Syntaxe de cette version en langage Algorithmique :

Pour (vari = v1 jusqu'à v2 pas = -1) **faire**

Action 1

…

Action k

Finpour

Exemple précédent dans cette version :

```
Algorithme SomArithmetiqueFor
Donnée  n : entier
Résultats  som : entier                 //  Numéro de boucle
Variables i : entier
Début
   Lire(n)
i ← n
Pour i=n jusqu'à 1 faire              // Pas = -1 (Implicite)
    som ← som+i
Ecrire(" Boucle :",n-i+1, " Somme =",som)
finpour
Ecrire("Somme 1+2+++…+n = ",som)
Fin SomArithm
```

```
for (vari=v1 ; vari<= v2 ; vari--) {          // vari-- :  vari = vari – 1 : Pas= -1 (explicite)
    Instr1 ;
    Instr2 ;
    ….
    InstrK ;
}
```

Exemple précédent en langage C :

```
#include <stdio.h>
int main(){
int i,n,som;
printf("Valeur de n : ");
scanf("%d",&n) ;
som=0;
for (i=n; i>=1; i--){
    som += i;
    printf("\n Boucle  = %d   som = %d",n-i+1,som);
}
printf("\n Somme de 1 a  %d",n);
printf(" est : %d", som);
}
```

Exécution :

```
Valeur de n : 6
Boucle  = 1   som = 6
Boucle  = 2   som = 11
Boucle  = 3   som = 15
Boucle  = 4   som = 18
Boucle  = 5   som = 20
Boucle  = 6   som = 21
Somme de 1 a  6 est : 21
```

Version 3: Dans cette version :

- Nous avons v1≤ v2
- Nous partons donc de v1 vers v2 en incrémentant le compteur de boucles de *pas = 2*.

Syntaxe en langage algorithmique :

Pour (vari= v1 **jusqu'à** v2 pas =2) **faire**

Action 1

…

Action k

Finpour

Exemple : Reprendre l'exemple précédent en calculant cette fois la somme des nombres impairs, le numéro de chaque boucle et le nombre impair correspondent.

```
Algorithme SomPairImpair
Données n : entier
Résultats simpair : entier
Variables i,j : entier
Début
   Ecrire(''Donnez n = '')
   Lire(n)
   simpair ← 0
   j ← 0                                   // Nombre valeurs impaires et Boucles
   pour (i=v1 jusqu'à v2 pas = 2) faire
      simpair ← simpair +i                 // somme des valeurs impaires
      j ← j+1
      ecrire(" Boucle :'',j, '' Nombre impair'',i, '' , ''simpair ='',simpair)
   finpour
  fin
```

Programme en langage C :

```
#include <stdio.h>
int main(){
int i,j,n,simpair;
printf("Valeur de n : ");
scanf("%d",&n) ;
simpair=0;
j=0;
for (i=1;i<=n; i+=2){
    simpair += i;
    j++;
    printf("\n Boucle  = %d   Nombre impair: %d simpair = %d   ",j,i,simpair);
}
printf("\n Somme Nombres impairs de 1 a  %d",n);
printf(" est : %d", simpair);
}
```

Exécution :

```
Valeur de n : 6
Boucle  = 1   Nombre impair: 1 simpair = 1
Boucle  = 2   Nombre impair: 3 simpair = 4
Boucle  = 3   Nombre impair: 5 simpair = 9
Somme Nombres impairs de 1 a  6 est : 9
```

4.3 La boucle Tantque…Faire :

Cette structure algorithmique de contrôle est la plus générale des trois formes de répétition. La répétition ici est plutôt basée sur une condition : *Cond* (*Expression arithmétique et Logique*). Elle se présente comme suit :

Syntaxe algorithmique :

```
Tantque Cond Faire
    Action1 ;
    Action2 ;
    …
FinTantque
```

Effet :

lorsque *Cond==Vrai*, il y a exécution de toutes les actions entre *Faire* et *FinTanque*.

Lorsque *Cond==Faux*, il y à saut en avant, après le *FinTantque*.

Syntaxe en langage C :

```
While (Cond) {
    Actio1 ;
    Action2 ;
    ….
}
```

Clairement, ici, il y a seulement remplacement de *faire* par { et *FinTanque* par }.

Exemple :

Nous répétons le même problème précédent : somme arithmétique des nombres : 1, 2,…,n.

Voici un algorithme basé sur la *boucle Tanque…Faire* :

```
Algorithme SomArithmetiqueTanque
Donnée : n : entier
Résultats : som : entier
Variables :
    i : entier                    // compteur de la boucle for
Lire(n)
i ← 1
Som ← 0
Tantque i≤ n faire
    som ← som+i
    i ← i+1                  // incrémentation explicite du compteur i
finpour
Ecrire('Somme de 1 à n=', som)
Fin SomArithm
```

Programme équivalent en langage C :

```
#include <stdio.h>
main(){
int i,n,som;
printf("Valeur de n :");
scanf("%d",&n) ;
i=1 ;
som=0;
while (i<=n) {
   som= som + i;
   i++;                          //  i++: Incrémentation explicite du compteur i
}
printf("somme de 1 a  %d",n);
printf(" est : %d", som);
}
```

Exécution :

```
Valeur de n :100
somme de 1 a  100 est : 5050
```

4.3 La boucle Répéter…Jusqu'à :

Cette structure algorithmique de contrôle réalise aussi la répétition sur la base d'une condition logique, soit, *Cond*. La différence entre les deux structures est que :

- Pour le ***Tanque***, la condition est à l'entrée du bloc d'actions à répéter. Conséquences : La boucle peut ne pas s'exécuter du tout.
- Pour la boucle ***Répéter***, la condition est après ce bloc. Conséquences : La boucle est exécutée toujours, au moins une fois.

Ces trois structures assurent alors un choix complet pour toutes les situations de répétition.

Syntaxe algorithmique :

```
Répéter
    Action1 ;
    Action2 ;
    …
Jusqu'à Cond
```

Effet :

- Il y a exécution du bloc d'actions *inconditionnellement* dans le premier passage.
- Après chaque passage par le bloc d'actions :

  Lorsque *Cond == Faux* : Il y a exécution de la boucle une fois de plus ;

  Lorsque *Cond == Vrai* : Il y a arrêt de la boucle (continuation après *jusqu'à Cond*).

Remarque : Pour s'arrêter :

- La boucle *Tantque* vérifie que Cond==Faux
- La boucle *répéter* vérifie que Cond==Vrai.

Syntaxe en langage C :

```
Do {
    Action1 ;
    Action2 ;
    …
}
While (Cond) ;
```

Exemple :

Nous reprenons l'exemple précédent de la somme arithmétique (som = 1+2+…n), par la boucle *répéter* :

Forme en langage algorithmique :

```
Algorithme SomArithmetiqueRépéter
Données : n : entier
Résultats : som : entier
Variables :
    i : entier                          // compteur de la boucle répéter
Lire(n)
Som ← 0
i ← 1
répéter
    som ← som+i
    i ← i+1                             // incrémentation explicite du compteur i
jusqu'à i>n
Ecrire('Somme de 1 à n=', som)
Fin SomArithm
```

Programme en langage C :

```
#include <stdio.h>
void main(){
int i,n,som;
printf("Valeur de n :");
scanf("%d",&n) ;
som=0;
i=1;
do {
    som += i;                           // som+=i;  est:  som=som+i;
        i++;
    }
while (i<=n);
printf("somme de 1 a  %d",n);
printf(" est : %d", som);
}
```

Exécution :

```
Valeur de n :100
somme de 1 a  100 est : 5050
```

4.5 Les boucles imbriquées :

Ce genre de boucles est composé d'une boucle externe, à l'intérieur de laquelle il y a une autre boucle (boucle interne). Il peut y avoir encore une 3ième boucle dans la 2ième boucle (interne) et ainsi de suite.

Evidemment, ce genre de boucle est plus compliqué à gérer, mais il permet de résoudre des problèmes encore plus difficiles. Ce principe est illustré sur la Figure 4.1

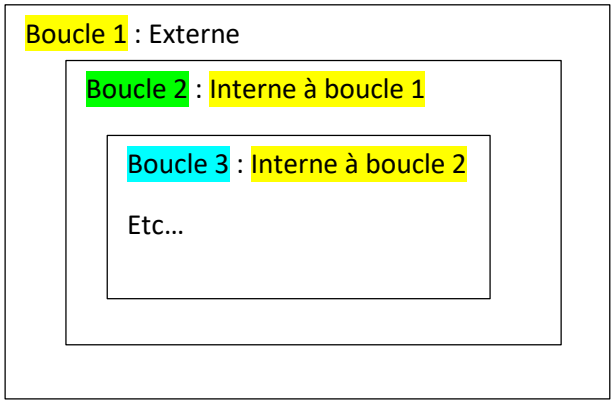

Figure 4.1 Principe des boucles imbriquées

Dans ce chapitre, nous considérons le cas de deux boucles imbriquées, à travers l'exemple suivant.

Exemple 1: Ecrire un algorithme qui écrit tous les nombres suivants :

- Ligne 1 : 1 à 1
- Ligne 2 : 1 à 2
- …
- Ligne i : 1 à i
- …
- Ligne n : 1 à n

n étant un nombre entier positif lu en entrée.

Algorithme en langage algorithmique :

```
Algorithme Lignes
Données n : entier
Résultats
Variable i,j : entier
Début
    Pour i=1 à n faire              // Boucle 1 (externe) : Compteur: i
      Pour j=1 à i faire
        Ecrire(j)                   // Boucle 2 (interne) : écriture des nombres j ∈ 1 … i
                                    // Compteur : j
      Finpour
      Ecrire('\n ' )                // Saut de ligne en fin de chaque  boucle 2
    Finpour
Fin Lignes
```

2. **En langage C :**

```
#include <stdio.h>
int main(){
int i,j,n;
printf("Valeur de n :");
scanf("%d",&n) ;

for (i=1;i<=n;i++){             // Boucle 1 (externe): Compteur: i
   for (j=1;j<=i;j++){
      printf("%d ",j);          // Boucle 2 (interne) : écriture des nombres j ∈ 1 … i
                                // Compteur : j
   }
   printf("\n");                // Saut de ligne en fin de chaque  boucle 2
}
}
```

Exécution :

```
Valeur de n :10
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
```

Exemple 2 : Reprendre le même exemple, cette fois-ci, en comptabilisant la somme et le produit de Tous les nombres j, $1 \leq j \leq i$, avec $1 \leq i \leq n$.

Nous demandons ici d'appliquer la méthodologie de la programmation, cycle complet (toutes les étapes) :

Solution proposée :
Rappel : **Méthodologie de la programmation**
La méthodologie de la programmation est un ensemble d'étapes méthodiques à appliquer dans un ordre précis pour résoudre un problème donné P par ordinateur.

Etapes de la méthodologie de la programmation :

1. Expression (énoncé) du problème P
   Cette étape est donné dans le texte de l'exemple (ou de l'exercice).
2. Analyse du problème P
   Cette étape consiste à déterminer :
   Données ?
   Résultats ?
3. Proposition d'une solution S (Algorithme)
   Cette étape consiste à trouver une succession d'étapes claires menant des Données aux résultats (*Chemin* : Données $\rightarrow$ Résultats)
4. Expression de l'algorithme S en langage algorithmique ou en organigramme
5. Programmation de l'algorithme S dans le langage C

Etapes de la solution :

1. Expression du problème : Reprendre le même exemple, cette fois-ci, en comptabilisant la somme et le produit de tous les nombres j, $1 \leq j \leq i$, avec $1 \leq i \leq n$.

2. Analyse du problème :
   Données : n (nombre entier positif).
   Résultats : somj = 1+ 2 + … j et prodj= 1*2*…*j, avec : $1 \leq j \leq i$, et $1 \leq i \leq n$.

3. Algorithme :

- Nous utilisons une boucle externe (boucle 1), de compteur i, $1 \leq i \leq n$.
- Une boucle interne (boucle 2), de compteur j, $1 \leq j \leq i$.
- A l'intérieur de la boucle 1, pour chaque valeur i, $1 \leq i \leq n$, nous (ré-)initialisons les valeurs somi à 0 et prodj à 1.
- A l'intérieur de la boucle 2, nous mettons à jour somj et prod j, comme suit :
  somj $\leftarrow$ somj + j, j, $1 \leq j \leq I$,
  prodj $\leftarrow$ prodj * j, j, $1 \leq j \leq i$.
- A la sortie de chaque boucle interne, nous écrivons les valeurs somj et prodj.
- Fin

4. Expression de l'algorithme en langage algorithmique :

```
Algorithme SomjProdj
Données  n : entier
Résultats somj, prodj : entier
Variables j : entier
Début
  Lire(n)
  Pour (i=1 jusqu'à n) faire                    // Boucle 1 : Externe
      Somj ← 0
      Prodj ← 1
      Pour (j=1 jusqu'à i) faire                // Boucle 2 : Interne à la boucle 1
         Somj ← somj + j
         Prodj ← prodj * j
      Finpour                                   // Fin de la boucle interne
      Ecrire(''Ligne : '',j, '' somj= '', somj, ''prodj= '', prodj)
  Finpour                                       // Fin de la boucle externe
  fin
```

5. Expression de l'algorithme en langage C :

```
#include <stdio.h>
int main(){
int i,j,n,somj,prodj ;
printf("Valeur de n : ");
scanf("%d",&n) ;

for (i=1;i<=n;i++){            // Boucle 1 (externe): Compteur: i
   somj=0;
   prodj=1;
   for (j=1;j<=i;j++){         // Boucle 2 (interne): Compteur : j
      somj +=j;
     prodj *=j;
   }
   printf(" Somme de: 1 a: %d =  %d  -- produit de: 1 a: %d = %d \n",i,somj,i,prodj);   // somme 1 à j et produit 1 à j
}
}
```

Exécution :

```
Valeur de n : 10
 Somme de: 1 a: 1 =  1  -- produit de: 1 a: 1 = 1
 Somme de: 1 a: 2 =  3  -- produit de: 1 a: 2 = 2
 Somme de: 1 a: 3 =  6  -- produit de: 1 a: 3 = 6
 Somme de: 1 a: 4 =  10  -- produit de: 1 a: 4 = 24
 Somme de: 1 a: 5 =  15  -- produit de: 1 a: 5 = 120
 Somme de: 1 a: 6 =  21  -- produit de: 1 a: 6 = 720
 Somme de: 1 a: 7 =  28  -- produit de: 1 a: 7 = 5040
 Somme de: 1 a: 8 =  36  -- produit de: 1 a: 8 = 40320
 Somme de: 1 a: 9 =  45  -- produit de: 1 a: 9 = 362880
 Somme de: 1 a: 10 =  55  -- produit de: 1 a: 10 = 3628800
```