

Module BASES DE L'ALGORITHMIQUE

TD 3 (1er partie)

Les boucles

Hervé Owsinski
Herve.Owsinski@uphf.fr

Semestre 1



2026-2027

Tant Que

Rappel - Définitions

Rappel (définition) : Structure de contrôle

Les **structures de contrôle** sont des outils qui permettent de **contrôler l'ordre** dans lequel sont exécutées les instructions de l'algorithme. Deux éléments sont directement liées aux structures de contrôle :

- la notion de condition (TD2) ;
- la notion de boucle (aujourd'hui, TD3).

Cours - La répétition

Souvent nécessaire de répéter plusieurs fois un bloc d'actions.

Exemple : compte à rebours 10 9 8 7....

Trop fastidieux de procéder en écrivant Écrire("10"); Écrire("9"); ...

Les ... ne sont pas reconnus comme une instruction par la machine !

Par contre, il existe des schémas de répétition.

Définition Tant Que ... faire

Définition 1 : Boucle Tant que

Le schéma répétitif (ou boucle) **tant que ... faire** ou simplement **tant que** permet de **répéter** un bloc de lignes **tant qu'une condition est vérifiée**.

Le tant que se traduit en pseudo-langage sous la forme suivante :

```
10 /* actions avant... */
11
12 tant que (condition) faire
13     action1 ;
14     action2 ;
15 fin tant que
16
17 /* actions après... */
```

(VOCABULAIRE) **Une itération** : réaliser une fois le bloc du schéma répétitif. Faire deux itérations revient ici à suivre les lignes 12-13-14 - 12-13-14.

Définition Tant Que ... faire

... Définition 1 : Boucle Tant que ...

```
12 tant que (condition) faire  
13     action1 ;  
14     action2 ;  
15 fin tant que
```

Fonctionnement du tant que

- 1 On évalue d'abord en ligne 12 la condition booléenne ;
- 2 Si **Vrai**, on réalise les lignes 13-14 puis PUIS on revient en ligne 12 ;
- 3 Si **Faux**, on saute directement en ligne 15 pour sortir de la structure.

La condition d'un TantQue est une **condition de poursuite d'itération** : si elle est Vrai, on réalise une itération.

Ici, faire 3 itérations veut dire exécuter 12-13-14 - 12-13-14 - 12-13-14 - 12-15.

Définition Tant Que ... faire

Exemple d'utilisation du Tant Que

On imagine disposer d'une fonction **alea(a, b)** qui renvoie un entier aléatoirement compris dans l'intervalle [a; b]. Imaginons qu'on tire 3, 8 puis 7.

```
1  Programme TrouverNombreMystere
2      VAR nb_mystere, nb_alea, tentative : Entier ;
3  debut
4      Écrire("Entrez un nombre entre 1 et 10") ;
5      Lire(nb_mystere) ;                               /* On tape 7 */
6      nb_alea ← alea(1, 10) ;                          /* On tire 3 au début */
7      tentative ← 1 ;
8      Écrire("Tentative numéro", tentative) ;
9      tant que (nb_alea ≠ nb_mystere) faire             /* Si Vrai : on boucle */
10         nb_alea ← alea(1, 10) ;
11         tentative ← tentative + 1 ;
12         Écrire("Tentative numéro ", tentative) ;
13     fin tant que
14     Écrire("L'algorithme vient de trouver ", nb_alea) ;
15 fin
```

L 1-2-3-4-5-6-7-8 - 9(Vrai)-10-11-12 - 9(Vrai)-10-11-12 - 9(Faux)-13-14-15

Définition Tant Que ... faire : remarques importantes

... Définition 1 : Boucle Tant que ...

```
12 tant que (condition) faire  
13     action1 ;  
14     action2 ;  
15 fin tant que
```

Remarques

- **condition** est une condition de poursuite des itérations.
- Pour faire au moins une itération, la condition doit être **Vrai au départ**.
- Une fois dans la boucle, les instructions doivent pouvoir modifier les évaluations de la condition, sinon on ne sortira jamais : boucle infinie.
- Avec TantQue, on peut avoir **0, 1 ou plusieurs itérations**.

TD - Exercices

Exo 1

Exo 1 - Enoncé A

Dérouler les trois algorithmes suivants en indiquant les valeurs successives des variables et les affichages réalisés.

```
1  Programme exo1a
2      VAR i : Entier ;
3  début
4      i ← 1;
5      tant que (i ≤ 7) faire
6          Écrire(i) ;
7          i ← i + 1;
8      fin tant que
9  fin
```

Exo 1

Exo 1 - Correction A

Dérouler les trois algorithmes suivants en indiquant les valeurs successives des variables et les affichages réalisés

```
1  Programme exo1a
2      VAR i : Entier ;
3  début
4      i ← 1;
5      tant que (i ≤ 7) faire
6          Écrire(i) ;
7          i ← i + 1;
8      fin tant que
9  fin
```

Cet algorithme affiche 1 2 3 4 5 6 7.

Ou

1

2

3

...

Exo 1

Exo 1 - Enoncé B

Dérouler les trois algorithmes suivants en indiquant les valeurs successives des variables et les affichages réalisés.

```
1  Programme exo1b
2      VAR a, b, c : Entier ;
3  début
4      Écrire("Entrer un entier : ") ; Lire(a) ;
5      b  $\leftarrow$  a; c  $\leftarrow$  0 ;
6      tant que (b  $\geq$  5) faire
7          b  $\leftarrow$  b - 5;
8          c  $\leftarrow$  c + 1;
9      fin tant que
10     Écrire(b, c) ;
11 fin
```

Exo 1

Exo 1 - Correction B

Dérouler les trois algorithmes suivants en indiquant les valeurs successives des variables et les affichages réalisés.

```
1  Programme exo1b
2      VAR a, reste, quotient : Entier ;
3  début
4      Écrire("Entrer un entier : ") ; Lire(a) ;
5      reste  $\leftarrow$  a; quotient  $\leftarrow$  0 ;
6      tant que (reste  $\geq$  5) faire
7          reste  $\leftarrow$  reste - 5;
8          quotient  $\leftarrow$  quotient + 1;
9      fin tant que
10     Écrire(reste, quotient) ;
11 fin
```

Exo 1

Exo 1 - Enoncé C

Dérouler les trois algorithmes suivants en indiquant les valeurs successives des variables et les affichages réalisés.

```
1  Programme exo1c
2      VAR c : Caractère ;
3          b : Booléen ;
4          n : Entier ;
5  début
6      b ← FAUX ; n ← 0 ;
7      tant que (b = FAUX) faire
8          Lire(c) ;
9          n ← n + 1;
10         si(c ≥ 'a' ET c ≤ 'z') alors
11             b ← VRAI ;
12         fin si
13     fin tant que
14     Écrire(c, n) ;
15 fin
```

Exo 1

Exo 1 - Enoncé C

Dérouler les trois algorithmes suivants en indiquant les valeurs successives des variables et les affichages réalisés.

```
1  Programme exo1c
2      VAR c : Caractère ;
3          b : Booléen ;
4          n : Entier ;
5  début
6      b ← FAUX ; n ← 0 ;
7      tant que (b = FAUX) faire
8          Lire(c) ;
9          n ← n + 1 ;
10         si (c ≥ 'a' ET c ≤ 'z') alors
11             b ← VRAI ;
12         fin si
13     fin tant que
14     Écrire(c, n) ;
15 fin
```

Cet algorithme compte et affiche le nombre de tentatives pour obtenir une lettre minuscule.

Plus précisément, il affiche la lettre minuscule obtenue et le nombre de tentatives pour l'obtenir.

Exo 2.1

Dérouler l'algorithme suivant en considérant que les 5 séquences de valeurs suivantes (une valeur est associée à chaque instruction Lire(c)) :

1. 5

```
1  Programme exo2
2      VAR n, c : Entier ;
3  début
4      Lire(c) ; n ← 0 ;
5      tant que(c > 0 ET c < 4) faire
6          tant que(c > 0 ET c < 4) faire
7              Lire(c) ;
8          fin tant que
9          tant que(c < 8 ET c > 1) faire
10             n ← n + 1;
11             Lire(c) ;
12         fin tant que
13         Lire(c) ;
14     fin tant que
15     Écrire(n) ;
16 fin
```

Exo 2.2

Dérouler l'algorithme suivant en considérant que les 5 séquences de valeurs suivantes (une valeur est associée à chaque instruction Lire(c)) :

2. **3 9 10**

```
1  Programme exo2
2      VAR n, c : Entier ;
3  début
4      Lire(c) ; n ← 0 ;
5      tant que(c > 0 ET c < 4) faire
6          tant que(c > 0 ET c < 4) faire
7              Lire(c) ;
8          fin tant que
9          tant que(c < 8 ET c > 1) faire
10             n ← n + 1;
11             Lire(c) ;
12         fin tant que
13         Lire(c) ;
14     fin tant que
15     Écrire(n) ;
16 fin
```

Exo 2.3

Dérouler l'algorithme suivant en considérant que les 5 séquences de valeurs suivantes (une valeur est associée à chaque instruction Lire(c)) :

3. **3 6 8 4**

```
1  Programme exo2
2      VAR n, c : Entier ;
3  début
4      Lire(c) ; n ← 0 ;
5      tant que(c > 0 ET c < 4) faire
6          tant que(c > 0 ET c < 4) faire
7              Lire(c) ;
8          fin tant que
9          tant que(c < 8 ET c > 1) faire
10             n ← n + 1;
11             Lire(c) ;
12         fin tant que
13         Lire(c) ;
14     fin tant que
15     Écrire(n) ;
16 fin
```

Exo 2.4

Dérouler l'algorithme suivant en considérant que les 5 séquences de valeurs suivantes (une valeur est associée à chaque instruction Lire(c)) :

4. **2 2 1 3 4 0 4**

```
1  Programme exo2
2      VAR n, c : Entier ;
3  début
4      Lire(c) ; n ← 0 ;
5      tant que(c > 0 ET c < 4) faire
6          tant que(c > 0 ET c < 4) faire
7              Lire(c) ;
8          fin tant que
9          tant que(c < 8 ET c > 1) faire
10             n ← n + 1;
11             Lire(c) ;
12         fin tant que
13         Lire(c) ;
14     fin tant que
15     Écrire(n) ;
16 fin
```

Exo 2.5

Dérouler l'algorithme suivant en considérant que les 5 séquences de valeurs suivantes (une valeur est associée à chaque instruction Lire(c)) :

5. **1 4 2 6 10 2 6 12 3 6 8 1 11 0**

```
1  Programme exo2
2      VAR n, c : Entier ;
3  début
4      Lire(c) ; n ← 0 ;
5      tant que(c > 0 ET c < 4) faire
6          tant que(c > 0 ET c < 4) faire
7              Lire(c) ;
8          fin tant que
9          tant que(c < 8 ET c > 1) faire
10             n ← n + 1;
11             Lire(c) ;
12         fin tant que
13         Lire(c) ;
14     fin tant que
15     Écrire(n) ;
16 fin
```

Exo 2.6

Dérouler l'algorithme suivant en considérant que les 5 séquences de valeurs suivantes (une valeur est associée à chaque instruction Lire(c)) :

6. **1 9 2 -1 2 8 1 0 2 2 12 7**

```
1  Programme exo2
2      VAR n, c : Entier ;
3  début
4      Lire(c) ; n ← 0 ;
5      tant que(c > 0 ET c < 4) faire
6          tant que(c > 0 ET c < 4) faire
7              Lire(c) ;
8          fin tant que
9          tant que(c < 8 ET c > 1) faire
10             n ← n + 1;
11             Lire(c) ;
12         fin tant que
13         Lire(c) ;
14     fin tant que
15     Écrire(n) ;
16 fin
```

Correction 2

Cet algorithme affiche 0 si on tape un chiffre différent de 1, 2 ou 3 au début.

Etape 1 : Il accepte ensuite toute séquence de 1, 2 ou 3 sans rien en faire.

Etape 2 est activé une fois qu'on tape un chiffre différent de 1, 2, 3 : il compte alors le nombre de chiffres successifs parmi 2-3-4-5-6-7.

Etape 3 : Si le chiffre derrière la séquence précédente est un 1, on repart à l'étape 1. Sinon, on stoppe la machine.

Automate ?

Répéter jusqu'à

Définition Répéter Jusqu'à

Définition 2 : Boucle Répéter Jusqu'à

Le schéma **répéter . . . jusqu'à** consiste à répéter plusieurs actions jusqu'à ce qu'une condition soit vérifiée.

Le tant que se traduit en pseudo-langage sous la forme suivante :

```
10 /* actions avant le jusqu'à */
11
12 répéter          /* notez que c'est "répéter", pas "faire" */
13     action1 ;
14     action2 ;
15 jusqu'à (condition) ;
16
17 /* actions après le jusqu'à */
```

Définition Répéter Jusqu'à

... Définition 2 : Boucle Répéter Jusqu'à ...

```
12 répéter          /* notez que c'est "répéter", pas "faire" */  
13     action1 ;  
14     action2 ;  
15 jusqu'à (condition) ;
```

Fonctionnement du jusqu'à

- ➊ On réalise au moins une fois les actions du **répéter** (L12-L13-L14) ;
- ➋ On évalue ensuite la condition (L15) ;
- ➌ Si la condition est **Vrai**, on sort de la structure (L16+) ;
- ➍ Si la condition est **Faux**, on revient en L12.

La condition du Jusqu'à est une **condition d'arrêt** des itérations : si Vrai, on sort.

On réalise **toujours au moins une itération**, jamais 0.

Définition Répéter ... jusqu'à

Exemple d'utilisation du Jusqu'à

On imagine disposer d'une fonction **alea(a, b)** qui renvoie un entier aléatoirement compris dans l'intervalle $[a; b]$. Imaginons qu'on tire 3, 8 puis 7.

```
1  Programme TrouverNombreMystere
2      VAR nb_mystere, nb_alea, tentative : Entier ;
3  debut
4      Écrire("Entrez un nombre entre 1 et 10") ;
5      Lire(nb_mystere) ;                               /* on tape 7 */
6      tentative ← 0 ;
7      Répéter
8          nb_alea ← alea(1, 10) ;
9          tentative ← tentative + 1 ;
10         Écrire("Tentative numéro", tentative) ;
11     jusqu'à (nb_alea = nb_mystere) ;
12     Écrire("L'algorithme vient de trouver ", nb_alea) ;
13 fin
```

L 1-2-3-4-5-6 - 7-8-9-10-11(Faux) - 7-8-9-10-11(Faux) - 7-8-9-10-11(Vrai) -12-13

Comparaison Tant que et Jusqu'à

Rappel : exemple d'utilisation du Jusqu'à

```
1  Programme TrouverNombreMystere
2      VAR nb_mystere, nb_alea, tentative : Entier ;
3  debut
4      Écrire("Entrez un nombre entre 1 et 10") ;
5      Lire(nb_mystere) ;
6      tentative ← 0 ;
7      Répéter
8          nb_alea ← alea(1, 10) ;          /* toutes les tentatives sont ici */
9          tentative ← tentative + 1 ;
10         Écrire("Tentative numéro", tentative) ;
11     jusqu'à (nb_alea = nb_mystere) ; /* Jusqu'à ce que les 2 soient identiques */
12     Écrire("L'algorithme vient de trouver ", nb_alea) ;
13 fin
```

Les actions à faire ne sont présentes que dans le bloc de la structure..
La condition d'arrêt correspond ici à une EGALITE.

Comparaison Tant que et Jusqu'à

Rappel : exemple d'utilisation du Tant Que

```
1  Programme TrouverNombreMystere
2      VAR nb_mystere, nb_alea, tentative : Entier ;
3  debut
4      Écrire("Entrez un nombre entre 1 et 10") ;
5      Lire(nb_mystere) ;
6      nb_alea ← alea(1, 10) ;                /* 1er tentative ici */
7      tentative ← 1 ;
8      Écrire("Tentative numéro", tentative) ;
9      tant que (nb_alea ≠ nb_mystere) faire /* Tant que les 2 sont différents */
10         nb_alea ← alea(1, 10) ;            /* tentatives suivantes */
11         tentative ← tentative + 1 ;
12         Écrire("Tentative numéro ", tentative) ;
13     fin tant que
14     Écrire("L'algorithme vient de trouver ", nb_alea) ;
15 fin
```

Les actions à réaliser sont dans le bloc de structures mais on trouve également une première forme AVANT la boucle puisqu'on veut agir 1 fois au moins mais c'est un TantQue!

La condition de poursuite correspond à une DIFFERENCE, complément du =.

Comparaison Tant que et Jusqu'à

Deux façons de résoudre le même problème

Les structures **Tant Que** et **Jusqu'à** sont interchangeables : on peut toujours transformer l'une en l'autre. Néanmoins, lorsqu'on doit réaliser :

- des actions **1 fois ou plus** autant prendre un **Jusqu'à** ;
- des actions **0 fois ou plus** autant prendre un **Tant que**.

Passage de Condition d'arrêt à Condition de poursuite

L'opérateur NON inverse une expression booléenne :

NON Vrai donne **Faux** et **NON Faux** donne **Vrai**.

On peut donc exprimer l'une des conditions en fonction de l'autre :

- **condition_arrêt** peut s'écrire comme **NON condition_poursuite**.
- **condition_poursuite** peut s'écrire comme **NON condition_arrêt**.

Exemple : exprimons **jusqu'à (nb_alea = nb_mystere)** sous forme d'un Tant Que :
tant que (NON (nb_alea = nb_mystere)) qui donne
tant que (nb_alea $\neq$ nb_mystere).

Exo 3

Exercice 3 :

Expliquer ce que fait l'algorithme ci-dessous.

```
1  Programme exo3
2      VAR u : Réel ;
3  début
4      u ← 2;
5      répéter
6          écrire(u) ;
7          u ← racine(u) ;
8      jusqu'à u ≤ 1.001 ;
9  fin
```

Proposer une autre version pour faire la même chose, mais en utilisant une boucle **tant que**.

Exo 3

Correction 3 :

Expliquer ce que fait l'algorithme ci-dessous.

```
1  Programme exo3
2      VAR u : Réel ;
3  début
4      u ← 2;
5      tant que (u > 1.001) faire
6          écrire(u) ;
7          u ← racine(u) ;
8      fin tant que
9  fin
```

Proposer une autre version pour faire la même chose, mais en utilisant une boucle **tant que**.

Faire Tant Que

Définition Faire ... tant que

Définition 3 : Boucle Faire ... Tant que

Le schéma **faire ... tant que** est une autre variante qui consiste à répéter une ou plusieurs actions tant qu'une condition est vérifiée.

Le faire ... tant que se traduit en pseudo-langage sous la forme suivante :

```
10 /* actions avant le faire ... tant que ; */
11
12 faire
13     action1 ;
14     action2 ;
15 tant que (condition) ;
16
17 /* actions après le faire ... tant que ; */
```

Définition Faire ... Tant Que

... Définition 1 : Boucle Faire ...Tant que

```
12 faire  
13     action1 ;  
14     action2 ;  
15 tant que (condition) ;
```

Fonctionnement du faire ... tant que

- 1 On réalise d'abord les actions (12-13-14) ;
- 2 On évalue ensuite la condition (15) ;
- 3 Si **Vrai**, on réalise une nouvelle itération (retour en 12) ;
- 4 Si **Faux**, on sort de la structure (passage en 16).

La condition est une **condition de poursuite**.

On réalise toujours au moins une itération.

Définition Faire ... Tant Que

Exemple d'utilisation du Faire ... Tant Que

On imagine disposer d'une fonction **alea(a, b)** qui renvoie un entier aléatoirement compris dans l'intervalle [a; b]. Imaginons qu'on tire 3, 8 puis 7.

```
1  Programme TrouverNombreMystere
2      VAR nb_mystere, nb_alea, tentative : Entier ;
3  debut
4      Écrire("Entrez un nombre entre 1 et 10") ;
5      Lire(nb_mystere) ;           /* on tape 7 */
6      tentative ← 0 ;
7      faire
8          nb_alea ← alea(1, 10) ;
9          tentative ← tentative + 1 ;
10         Écrire("Tentative numéro ", tentative) ;
11     tant que (nb_alea ≠ nb_mystere) ;
12     Écrire("L'algorithme vient de trouver ", nb_alea) ;
13 fin
```

Exemple de déroulé :

1-2-3-4-5-6 - 7-8-9-10-11(Vrai) - 7-8-9-10-11(Vrai) - 7-8-9-10-11(Faux) - 12-13

Exo 4

Expliquer ce que fait l'algorithme ci-dessous.

```
1  Programme exo4
2      VAR i : Entier ;
3          s : Réel ;
4  début
5      i ← 0;
6      s ← 0.0 ;
7      faire
8          i ← i + 1;
9          s ← s + racine(i) ;
10     tant que (s ≤ 10) ;
11 fin
```

Proposer une autre version pour faire la même chose, mais en utilisant une boucle **jusqu'à**.

Exo 4

```
1  Programme exo4
2      VAR i : Entier ;
3          s : Réel ;
4  début
5      i ← 0;
6      s ← 0.0 ;
7      faire
8          i ← i + 1;
9          s ← s + racine(i) ;
10     tant que (s ≤ 10) ;
11 fin
```

- **L05-L06** : on initialise i à 0 et s à 0.0;
- **L07-L10** : on incrémente i de 1 et on incrémente s de $\sqrt{1}$. On obtient $s = \sqrt{1}$.
La poursuite est Vrai car ($s \leq 10$) ;
- **L07-L10** : on incrémente i de 1 et on incrémente s de $\sqrt{2}$. On obtient $s = \sqrt{1} + \sqrt{2}$.
La poursuit est Vrai car ($s \leq 10$) ;
- **L07-L10** : on incrémente i de 1 et on incrémente s de $\sqrt{3}$. On obtient $s = \sqrt{1} + \sqrt{2} + \sqrt{3}$.
La poursuit est Vrai car ($s \leq 10$) ; ...

Exo 4

```
1  Programme exo4
2      VAR i : Entier ;
3          s : Réel ;
4  début
5      i ← 0;
6      s ← 0.0 ;
7      faire
8          i ← i + 1;
9          s ← s + racine(i) ;
10     tant que (s ≤ 10) ;
11 fin
```

A chaque itération, on incrémente i et on ajoute à s un terme $\sqrt{i}$, on stope si $s > 10$.

Conclusion : cet algorithme détermine la plus petite valeur i à donner à i telle que :

$$\sqrt{1} + \sqrt{2} + \dots \sqrt{i} > 10$$

$$\sum_{k=1}^i \sqrt{k} > 10$$

Exo 4

Pour info :

$$\sum_{k=1}^5 \sqrt{k} = \sqrt{1} + \sqrt{2} + \sqrt{3} + \sqrt{4} + \sqrt{5} = 8.382332347441762...$$

$$\sum_{k=1}^6 \sqrt{k} = \sqrt{1} + \sqrt{2} + \sqrt{3} + \sqrt{4} + \sqrt{5} + \sqrt{6} = 10.83182209022494$$

Même algorithme mais avec une structure **répéter...jusqu'à** :

- Les deux structures itèrent 1+ fois : pas de changement à ce niveau ;
- Il suffit de transformer la condition de **poursuite** en condition d'**arrêt** ;
- sans oublier de remplacer les mots-clés **faire** par **répéter**.

Exo 4

```
1  Programme exo4BIS
2      VAR i : Entier ;
3          s : Réel ;
4  début
5      i ← 0;
6      s ← 0.0 ;
7      répéter
8          i ← i + 1;
9          s ← s + racine(i) ;
10     jusqu'à (s > 10) ;
11 fin
```

Remarque :

On notera que cet algorithme n'affiche rien.

Il calcule s et i et les place en mémoire.

Si la mémoire disparaît à la fin du programme, on peut dire que le programme ne fait... rien !

Résumé des trois boucles précédentes

Les 3 pseudo-codes

VERSION Tant que...faire	VERSION Faire...Tant que	VERSION Répéter...jusqu'à
tant que (cond_poursuite) faire action1 ; action2 ; fin tant que	faire action1; action2; tant que (cond_poursuite);	répéter action1; action2; jusqu'à (cond_arret);

On notera que :

- Avec condition de poursuite : le Tant Que...faire et le Faire...tant que ;
- Avec condition d'arrêt : Répéter...jusqu'à ;
- Avec 0+ itération(s) : Tant Que...faire ;
- Avec 1+ itération(s) : Faire...tant que et Répéter...jusqu'à ;