

Module BASES DE L'ALGORITHMIQUE

TD 3 (2e partie)

Les boucles

Hervé Owsinski
Herve.Owsinski@uphf.fr

Semestre 1



2026-2027

Résumé des trois boucles précédentes

Recréer un tant que à partir des autres

VERSION **Tant que...faire** -->

```
tant que (cond_poursuite) faire  
    action1 ;  
    action2 ;  
fin tant que
```

VERSION **Faire...Tant que**

VERSION **Répéter...jusqu'à**

Pour obtenir l'un des deux autres à partir d'un Tant Que :

Résumé des trois boucles précédentes

Recréer un tant que à partir des autres

VERSION **Tant que...faire** -->

```
tant que (cond_poursuite) faire  
    action1 ;  
    action2 ;  
fin tant que
```

VERSION **Faire...Tant que**

```
Si (cond_poursuite) alors  
    faire  
        action1;  
        action2;  
    tant que (cond_poursuite);  
Fin Si
```

VERSION **Répéter...jusqu'à**

Pour obtenir l'un des deux autres à partir d'un Tant Que :

- Transformation de la condition de poursuite en condition d'arrêt.

Attention : $x < y$ deviendrait $x \geq y$ par exemple

- Rajout d'une alternative pour obtenir 0 itération si nécessaire ;

Si la cond_poursuite est **Vrai** au début, on peut omettre l'alternative.

Résumé des trois boucles précédentes

Recréer un tant que à partir des autres

VERSION **Tant que...faire** -->

```
tant que (cond_poursuite) faire  
    action1 ;  
    action2 ;  
fin tant que
```

VERSION **Faire...Tant que**

```
Si (cond_poursuite) alors  
    faire  
        action1;  
        action2;  
    tant que (cond_poursuite);  
Fin Si
```

VERSION **Répéter...jusqu'à**

```
Si (cond_poursuite) alors  
    répéter  
        action1;  
        action2;  
    jusqu'à (cond_arret);  
Fin Si
```

Pour obtenir l'un des deux autres à partir d'un Tant Que :

- Transformation de la condition de poursuite en condition d'arrêt.

Attention : $x < y$ deviendrait $x \geq y$ par exemple

- Rajout d'une alternative pour obtenir 0 itération si nécessaire ;

Si la cond_poursuite est **Vrai** au début, on peut omettre l'alternative.

Pour

Définition 4 : Boucle Pour

Le schéma **pour** consiste à répéter une ou plusieurs actions **en comptant** le nombre d'itérations qui sont exécutées.

Ce schéma nécessite l'utilisation d'une variable, dite **compteur**, qui va être incrémentée ou décrémentée **automatiquement** à chaque passage dans la boucle.

compteur $\leftarrow$ **compteur** + **pas** automatiquement

Plusieurs éléments sont nécessaires à la définition de cette boucle :

- les deux **bornes** de l'intervalle fermé $[a; b]$ des valeurs possibles du compteur ;
- le **pas** avec lequel le compteur est modifié après chaque itération.

Définition 4 : Boucle Pour (suite), compteur qui augmente "à"

```
actions avant le pour ;  
pour cpt de départ à arrivée pas le_pas faire  
    action1 ;  
    action2 ;  
fin pour  
actions après le pour ;
```

Exemple

```
pour cpt de 10 à 23 pas 5 faire  
    action1 ;                /* Compteur prenant les valeurs */  
    action2 ;                /* 10 15 20   mais pas 25           */  
fin pour
```

Définition 4 : Boucle Pour (suite), compteur qui diminue "bas"

```
actions avant le pour ;  
pour cpt de départ bas arrivée pas le_pas faire  
    action1 ;  
    action2 ;  
fin pour  
actions après le pour ;
```

Exemple

```
pour cpt de 20 bas 9 pas -5 faire  
    action1 ;           /* Compteur prenant les valeurs */  
    action2 ;           */ 20 15 10    mais pas 5           */  
fin pour
```

Attention, le poly suit une syntaxe arrivée puis départ. Je préfère garder la logique départ puis arrivée très répandu dans les langages. Les deux choix sont néanmoins possibles. On trouve aussi la forme **pour cpt de 20 bas 9 pas 5 faire** signifiant -5 puisque bas est présent.

Exemples

Exemple 1

```
pour cpt de 12 à 26 pas 4 faire  
    action1 ;                      /* Compteur prenant les valeurs */  
    action2 ;                      /* 12 16 20 24    mais pas 28    */  
fin pour
```

Exemple 2

```
pour cpt de 2 bas -4 pas -2 faire  
    action1 ;                      /* Compteur prenant les valeurs */  
    action2 ;                      /* 2 0 -2 -4    mais pas -6    */  
fin pour
```

Exercice 5.a

Expliquer ce que font les trois algorithmes suivants.

Programme exo5a

VAR i, n : Entier ;

début

Écrire("Entrer un entier : ") ;

Lire(n) ;

pour i **de** n + 1 **à** n + 10 **pas** 1 **faire**

 Écrire(i) ;

fin pour

fin

```

1  Programme exo5a
2      VAR i, n : Entier ;
3  début
4      Écrire("Entrer un entier : ") ;
5      Lire(n) ;
6      pour i de n + 1 à n + 10 pas 1 faire
7          Écrire(i) ;
8      fin pour
9  fin

```

Le déroulé complet est :

04 - 05(n=5)

06(i=6) - 07(Affiche(6))

06(i=7) - 07(Affiche(7))

...

06(i=14) - 07(Affiche(14))

06(i=15) - 07(Affiche(15))

06(on vérifie qu'il n'y a plus de i possible) - 08(fin Pour) - 09 (fin)

Exercice 5.a

```
1  Programme exo5a
2  VAR i, n : Entier ;
3  début
4      Écrire("Entrer un entier : ") ;
5      Lire(n) ;
6      pour i de n + 1 à n + 10 pas 1 faire
7          Écrire(i) ;
8      fin pour
9  fin
```

Correction

Cet algorithme affiche un par un les entiers compris dans $[n+1 ; n+10]$.
Il affiche donc toujours 10 entiers.

Exercice 5.b

```
1  Programme exo5b
2      VAR i, n, t : Entier ;
3  début
4      Écrire("Entrer un entier : ") ;
5      Lire(n) ;
6      t ← 0;
7      pour i de 2 à n pas 2 faire
8          t ← t + i;
9      fin pour
10     Écrire(t) ;
11 fin
```

```

4   Écrire("Entrer un entier : ") ;   Lire(n) ;   t ← 0;
5   pour i de 2 à n pas 2 faire
6       t ← t + i;
7   fin pour
8   Écrire(t) ;

```

On initialise une variable t à 0. La première itération permet de l'incrémenter de 2. A chaque itération, on y ajoute le nombre pair suivant si on a bien $i \leq n$.

Intention : cet algorithme calcule et affiche la somme des entiers naturels pairs appartenant à l'intervalle $[2; n]$.

Selon que n soit pair ou impair, nous avons $n = 2k + 0$ ou $n = 2k + 1$.

$$t = \sum_{i=1}^p 2k = 2 + 4 + 6 + \dots + 2k = 2 \times (1 + 2 + 3 + \dots k)$$

Exercice 5.b

Or, rappel sur les séries arithmétiques :

$$\sum_{i=0}^k i = 0 + 1 + 2 + 3 + \dots + k = \frac{k \times (k+1)}{2}$$

Si on utilise ce résultat sur notre cas particulier des entiers pairs :

$$t = \sum_{i=1}^k 2i = 2 \times \sum_{i=1}^k i = k \times (k+1)$$

Exo 5

Exercice 5.c

```
1  Programme exo5c
2      VAR i, n : Entier ;
3  début
4      Écrire("Entrer un entier : ") ;
5      Lire(n) ;
6      pour i de 100 bas n pas -1 faire
7          si (i MOD 3 = 1) alors
8              écrire(i) ;
9          fin si
10     fin pour
11 fin
```

```

1  Programme exo5c
2      VAR i, n : Entier ;
3  début
4      Écrire("Entrer un entier : ") ;
5      Lire(n) ;
6      pour i de 100 bas n pas -1 faire
7          si (i MOD 3 = 1) alors
8              écrire(i) ;
9          fin si
10     fin pour
11 fin

```

La condition **$i \text{ MOD } 3 = 1$** signifie qu'on veut savoir si le reste de la division de i par 3 vaut 1. Les entiers pour lesquels la condition est vraie s'écrivent donc $n = 3k + 1$.

De tels entiers sont 1, 4, 7, 11, 14, ... 94, 97, 100, 103,... (exemple $100 = 3 \cdot 33 + 1$).

Attention, on lit dans l'autre sens donc on affiche 100, 97, 94, ...

Exo 6

- ❶ Écrire un algorithme qui affiche tous les nombres entiers impairs supérieurs à 0 et inférieurs à 100 avec une boucle **tant que**.
- ❷ Transformer cet algorithme pour utiliser une boucle **jusqu'à**, puis une boucle **faire ... tant que**.
- ❸ Réécrire l'algorithme mais en utilisant une boucle **pour**.

Exercice 6 : version Tant Que ... faire

```
1  Programme exo6
2      VAR i, n : Entier ;
3  début
4      n ← 1;
5      tant que (n < 100) faire
6          écrire(n);
7          n ← n + 2;
8      fin tant que
9  fin
```

Exo 6

Le **Tant que...faire** précédent réalise toujours au moins une itération.
La transformation en **faire...tant** que est donc très facile puisqu'on ne doit pas gérer le cas où aucun tour de boucle n'est nécessaire.

```
1  Programme exo6
2      VAR i, n : Entier ;
3  début
4      n ← 1;
5      faire
6          écrire(n);
7          n ← n + 2;
8      tant que (n < 100);
9  fin
```

Exo 6

Passer du **faire...tant que** au **répéter...jusqu'à** ne nécessite que de changer la condition :
une condition de poursuite en une condition d'arrêt.

```
1  Programme exo6
2      VAR i, n : Entier ;
3  début
4      n ← 1;
5      répéter
6          écrire(n);
7          n ← n + 2;
8      jusqu'à (n ≥ 100);
9  fin
```

Exercice 6 : version pour...faire

On peut utiliser un **Pour** puisque le nombre de tours est clairement défini.

```
1  VAR i, n : Entier ;  
2  début  
3      pour i de 1 à 100 pas 2 faire  
4          écrire(n);  
5      fin pour  
6  fin
```

Un peu tout en même temps

Transformer un Pour en Tant que

[Pour] $\rightarrow$ [Tant Que]

Version **Pour**

```
...  
Lire(n);  
t  $\leftarrow$  0 ;  
pour i de 2 à n pas 2 faire  
    t  $\leftarrow$  t + i ;  
fin pour  
Écrire(t) ;  
...  
...  
...
```

Version **Tant que**

```
...  
Lire(n);  
t  $\leftarrow$  0 ;  
...  
...
```

Transformer un Pour en Tant que

[Pour] $\rightarrow$ [Tant Que]

Version **Pour**

```
...  
Lire(n);  
t  $\leftarrow$  0 ;  
pour i de 2 à n pas 2 faire  
    t  $\leftarrow$  t + i ;  
fin pour  
Ecrire(t) ;  
...  
...  
...
```

Version **Tant que**

```
...  
Lire(n);  
t  $\leftarrow$  0 ;  
  
tant que (i  $\leq$  n) faire    # i ?  
    t  $\leftarrow$  t + i ;  
  
fin tant que  
Ecrire(t);  
...
```

Transformer un Pour en Tant que

[Pour] $\rightarrow$ [Tant Que]

Version **Pour**

```
...  
Lire(n);  
t  $\leftarrow$  0 ;  
pour i de 2 à n pas 2 faire  
    t  $\leftarrow$  t + i ;  
fin pour  
Écrire(t) ;  
...  
...  
...
```

Version TQ

```
...  
Lire(n);  
t  $\leftarrow$  0 ;  
i  $\leftarrow$  2 ; # Création du compteur  
tant que (i  $\leq$  n) faire  
  
    fin tant que  
    Écrire(t);  
...  
...
```

Transformer un Pour en Tant que

[Pour] $\rightarrow$ [Tant Que]

Version **Pour**

```
...  
Lire(n);  
t  $\leftarrow$  0 ;  
pour i de 2 à n pas 2 faire  
    t  $\leftarrow$  t + i ;  
fin pour  
Ecrire(t) ;  
...  
...  
...
```

Version TQ

```
...  
Lire(n);  
t  $\leftarrow$  0 ;  
i  $\leftarrow$  2 ; # Création du compteur  
tant que (i  $\leq$  n) faire  
    t  $\leftarrow$  t + i ;  
  
fin tant que  
Ecrire(t);  
...
```

Transformer un Pour en Tant que

[Pour] $\rightarrow$ [Tant Que]

Version **Pour**

```
...  
Lire(n);  
t  $\leftarrow$  0 ;  
pour i de 2 à n pas 2 faire  
    t  $\leftarrow$  t + i ;  
fin pour  
Ecrire(t) ;  
...  
...  
...
```

Version TQ

```
...  
Lire(n);  
t  $\leftarrow$  0 ;  
i  $\leftarrow$  2 ; # Création du compteur  
tant que (i  $\leq$  n) faire  
    t  $\leftarrow$  t + i ;  
    i  $\leftarrow$  i + 2 ; # Incrémentation  
fin tant que  
Ecrire(t);  
...
```

Transformer un Pour en Tant que

[Pour] $\rightarrow$ [Tant Que]

Version **Pour**

```
...  
Lire(n);  
t  $\leftarrow$  0 ;  
pour i de 2 à n pas 2 faire  
    t  $\leftarrow$  t + i ;  
fin pour  
Écrire(t) ;  
...  
...  
...
```

Version TQ

```
...  
Lire(n);  
t  $\leftarrow$  0 ;  
i  $\leftarrow$  2 ; # Création du compteur  
tant que (i  $\leq$  n) faire  
    t  $\leftarrow$  t + i ;  
    i  $\leftarrow$  i + 2 ; # Incrémentation  
fin tant que  
Ecrire(t);  
...
```

- N'oubliez pas de créer le compteur manuellement ;
- N'oubliez pas d'incrémenter le compteur manuellement.
- Ces deux services sont offerts directement par la boucle Pour

Exercice 7

On souhaite écrire un algorithme qui demande à l'utilisateur de saisir un nombre entier positif.

La saisie doit être répétée jusqu'à ce que le nombre entré soit bien positif.

Écrire une version de cet algorithme pour chacune des boucles de notre pseudo-langage (tant que, jusqu'à, faire ... tant que, pour), lorsque cela est possible.

Exercice 7 - Version Tant Que...faire V1

Version avec une fausse valeur avant la boucle : cette fausse première valeur doit permettre de réaliser une première itération.

```
1  Programme exo7tq1
2      VAR n : Entier ;
3  début
4      n ← -1;
5      tant que (n < 0) faire
6          Écrire("Fournir un entier positif : ") ;
7          Lire(n) ;
8      fin tant que
9  fin
```

Exercice 7 - Version Tant Que...faire V2

Version avec actions avant la boucle puis dans la boucle :

```
1  Programme exo7tq2
2      VAR n : Entier ;
3  début
4      Écrire("Fournir un entier positif : ") ;
5      Lire(n) ;
6      tant que (n < 0) faire
7          Écrire("Fournir un entier positif : ") ;
8          Lire(n) ;
9      fin tant que
10 fin
```

Exercice 7 - Version faire...tant que

En utilisant la version 2 de **tant que...faire**, le passage à **faire...tant que** est immédiat :

```
1  Programme exo7ftq
2      VAR n : Entier ;
3  début
4      faire
5          Écrire("Fournir un entier positif : ") ;
6          Lire(n) ;
7      tant que (n < 0);
8  fin
```

Exercice 7 - Version répéter...jusqu'à

Condition de poursuite du **faire...tant que** en condition d'arrêt.

Ne pas oublier de remplacer **faire** par **répéter** :

```
1  Programme exo7jq
2      VAR n : Entier ;
3  début
4      répéter
5          Écrire("Fournir un entier positif : ") ;
6          Lire(n) ;
7      jusqu'à (n ≥ 0);
8  fin
```

Exercice 7 - Version pour ?

Impossible : on ne peut pas prévoir à l'avancer combien de fois il faudra donner au maximum.

Or fixer un nombre maximum de tentatives ne peut pas garantir de parvenir à obtenir un nombre positif.

Exercice 8

Écrire un algorithme dans lequel on fait la somme de valeurs entières saisies et qui s'arrête lorsque cette somme dépasse 50 ou que l'utilisateur a saisi 10 valeurs.

L'algorithme affichera à la fin la somme obtenue et le nombre de valeurs saisies.

Exercice 8 - Version répéter...jusqu'à

La version la plus naturelle serait le **répéter...jusqu'à** puisque l'énoncé fournit deux conditions d'arrêt possible : la condition d'arrêt sera évaluée **Vrai** si la somme dépasse 51 ou si le nombre d'entiers déjà fournis est de 10.

```
1  Programme exo8jq
2      VAR n, somme, nombre : Entier ;
3  début
4      somme ← 0; /* Ne pas oublier l'initialisation ! */
5      nbr ← 0;
6      répéter
7          Écrire("Fournir un entier : ") ;
8          Lire(n) ;
9          somme ← somme + n;
10         nbr ← nbr + 1;
11     jusqu'à ((somme > 50) OU (nbr = 10));
12     Écrire("Somme des entiers fournis : ", somme) ;
13     Écrire("Nombre d'entiers fournis : ", nbr) ;
14 fin
```

Exercice 8 - Version faire...tant que

En utilisant l'opérateur logique unaire NON, on peut facilement obtenir le complément de la condition d'arrêt qui est la condition de poursuite. La condition de poursuite est moins évidente car le problème correspond plus à un jusqu'à .

```
1  Programme exo8ftq
2      VAR n, somme, nombre : Entier ;
3  début
4      somme ← 0;
5      nbr ← 0;
6      faire
7          Écrire("Fournir un entier : ") ;
8          Lire(n) ;
9          somme ← somme + n;
10         nbr ← nbr + 1;
11     tant que ( NON ( (somme > 50) OU (nbr = 10) ));
12     Écrire("Somme des entiers fournis : ", somme) ;
13     Écrire("Nombre d'entiers fournis : ", nbr) ;
14 fin
```

Exercice 8 - Version faire...tant que 2

En réfléchissant à la condition de poursuite, on peut aboutir à ceci :

- Arrêt : ($\text{somme} > 50$) **OU** ($\text{nbr} = 10$) devient en recherchant sa négation :
- Poursuite : ($\text{somme} \leq 50$) **ET** ($\text{nbr} \neq 10$)

```
1  Programme exo8ftq2
2      VAR n, somme, nombre : Entier ;
3  début
4      somme ← 0;
5      nbr ← 0;
6      faire
7          Écrire("Fournir un entier : ") ;
8          Lire(n) ;
9          somme ← somme + n;
10         nbr ← nbr + 1;
11     tant que ((somme ≤ 50) ET (nbr ≠ 10));
12     Écrire("Somme des entiers fournis : ", somme) ;
13     Écrire("Nombre d'entiers fournis : ", nbr) ;
14 fin
```

Exercice 8 - Version tant que...faire

La condition de poursuite est nécessairement Vrai au départ, la transformation en Tant Que...faire est donc facile :

```
1  Programme exo8ftq2
2  VAR n, somme, nombre : Entier ;
3  début
4  somme ← 0;
5  nbr ← 0;
6  tant que ((somme ≤ 50) ET (nbr ≠ 10)) faire
7  Écrire("Fournir un entier : ") ;
8  Lire(n) ;
9  somme ← somme + n;
10  nbr ← nbr + 1;
11 fin tant que
12 Écrire("Somme des entiers fournis : ", somme) ;
13 Écrire("Nombre d'entiers fournis : ", nbr) ;
14 fin
```

Exo 8 - complément optionnel

Lois de De Morgan avec les opérateurs ET, OU et NON

$NON(a \text{ ET } b)$
est équivalent à
 $(NON \text{ } a) \text{ OU } (NON \text{ } b)$

$NON(a \text{ OU } b)$
est équivalent à
 $(NON \text{ } a) \text{ ET } (NON \text{ } b)$

Lois de De Morgan avec les opérateurs conjonction, disjonction et négation

$\neg(a \wedge b)$
est équivalent à
 $(\neg a) \vee (\neg b)$

$\neg(a \vee b)$
est équivalent à
 $(\neg a) \wedge (\neg b)$

Exercice 9

- 1 Écrire un algorithme qui demande successivement 10 valeurs réelles à l'utilisateur, et qui lui indique ensuite quelle est la plus grande valeur saisie.
- 2 Réécrire l'algorithme précédent en changeant la boucle utilisée et en affichant en plus le rang auquel le plus grand nombre a été saisi (i.e., en premier, second, ...).

Exo 9

Exercice 9 - valeur max (exemple de réponse)

```
1  Programme exo9A
2      VAR n, valeur_maximum, nbr : Entier ;
3  début
4      Écrire("Fournir un entier : ") ;
5      Lire(valeur_maximum) ;          /* pour le moment, ce nombre est le maximum */
6      nbr ← 1;
7      tant que (nbr ≤ 9) faire
8          Écrire("Fournir un entier : ") ;
9          Lire(n);
10         nbr ← nbr + 1;
11         si (n > valeur_maximum) alors
12             valeur_maximum ← n;
13         fin si
14     fin tant que
15     Écrire("Le maximum est : ", valeur_maximum) ;
16 fin
```

Exercice 9 - valeur et rang max (exemple de réponse)

```
1  Programme exo9B
2      VAR n, valeur_maximum, rang_maximum, i : Entier ;
3  début
4      Écrire("Fournir un entier : ") ;
5      Lire(valeur_maximum) ;          /* pour le moment, ce nombre est le maximum */
6      rang_maximum ← 1;
7      pour i de 2 à 10 pas 1 faire
8          Écrire("Fournir un entier : ") ;
9          Lire(n);
10         si (n > valeur_maximum) alors
11             valeur_maximum ← n;
12             rang_maximum ← i;
13         fin si
14     fin pour
15     Écrire("Le maximum est : ", valeur_maximum) ;
16     Écrire("Son rang est : ", rang_maximum) ;
17 fin
```

Exo 10

Exercice 10.a

Expliquer ce que font les deux algorithmes suivants, dans lesquels l'instruction **aller_à_la_ligne** est une instruction prédéfinie du pseudo-langage permettant de passer à la ligne dans l'affichage.

```
1  Programme exo10a
2      VAR i, j : Entier ;
3  début
4      i ← 1 ;
5      tant que(i ≠ 4) faire
6          pour j de 1 à 5 pas 1 faire
7              Écrire(i, " ") ;
8          fin pour
9          aller_à_la_ligne ;
10         i ← i + 1;
11     fin tant que
12 fin
```

On considère donc que `écrire()` ne passe pas automatiquement à la ligne.

Exercice 10.a

```
1  Programme exo10a
2      VAR i, j : Entier ;
3  début
4      i ← 1 ;
5      tant que(i ≠ 4) faire
6          pour j de 1 à 5 pas 1 faire
7              Écrire(i, " ") ;
8          fin pour
9          aller_à_la_ligne ;
10         i ← i + 1;
11     fin tant que
12 fin
```

- $i=1$, TQ condition Vrai donc itération : affiche 1 1 1 1 1, passage à la ligne puis i à 2 ;
- $i=2$, TQ condition Vrai donc itération : affiche 2 2 2 2 2, passage à la ligne puis i à 3 ;
- $i=3$, TQ condition Vrai donc itération : affiche 3 3 3 3 3, passage à la ligne puis i à 4 ;
- $i=4$, TQ condition Faux : fin du tant que.

```
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
```

Exercice 10.a

```
1  Programme exo10a
2      VAR i, j : Entier ;
3  début
4      i ← 1 ;
5      tant que(i ≠ 4) faire
6          pour j de 1 à 5 pas 1 faire
7              Écrire(i, " ") ;
8          fin pour
9          aller_à_la_ligne ;
10         i ← i + 1;
11     fin tant que
12 fin
```

- $i=1$, TQ condition Vrai donc itération : affiche 1 1 1 1 1, passage à la ligne puis i à 2 ;
- $i=2$, TQ condition Vrai donc itération : affiche 2 2 2 2 2, passage à la ligne puis i à 3 ;
- $i=3$, TQ condition Vrai donc itération : affiche 3 3 3 3 3, passage à la ligne puis i à 4 ;
- $i=4$, TQ condition Faux : fin du tant que.

```
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
```

Exercice 10.a

```
1  Programme exo10a
2      VAR i, j : Entier ;
3  début
4      i ← 1 ;
5      tant que(i ≠ 4) faire
6          pour j de 1 à 5 pas 1 faire
7              Écrire(i, " ") ;
8          fin pour
9          aller_à_la_ligne ;
10         i ← i + 1;
11     fin tant que
12 fin
```

- $i=1$, TQ condition Vrai donc itération : affiche 1 1 1 1 1, passage à la ligne puis i à 2 ;
- $i=2$, TQ condition Vrai donc itération : affiche 2 2 2 2 2, passage à la ligne puis i à 3 ;
- $i=3$, TQ condition Vrai donc itération : affiche 3 3 3 3 3, passage à la ligne puis i à 4 ;
- $i=4$, TQ condition Faux : fin du tant que.

```
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
```

Exercice 10.a

```
1  Programme exo10a
2      VAR i, j : Entier ;
3  début
4      i ← 1 ;
5      tant que(i ≠ 4) faire
6          pour j de 1 à 5 pas 1 faire
7              Écrire(i, " ") ;
8          fin pour
9          aller_à_la_ligne ;
10         i ← i + 1;
11     fin tant que
12 fin
```

- $i=1$, TQ condition Vrai donc itération : affiche 1 1 1 1 1, passage à la ligne puis i à 2 ;
- $i=2$, TQ condition Vrai donc itération : affiche 2 2 2 2 2, passage à la ligne puis i à 3 ;
- $i=3$, TQ condition Vrai donc itération : affiche 3 3 3 3 3, passage à la ligne puis i à 4 ;
- $i=4$, TQ condition Faux : fin du tant que.

```
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
```

Exercice 10.b

Expliquer ce que font les deux algorithmes suivants, dans lesquels l'instruction **aller_à_la_ligne** est une instruction prédéfinie du pseudo-langage permettant de passer à la ligne dans l'affichage.

```
1  Programme exo10b
2      VAR i, j : Entier ;
3  début
4      pour i de 1 à 9 pas 1 faire
5          Écrire(i, " : ") ;
6          j ← 1;
7          tant que (j ≤ 10) faire
8              Écrire(i * j, " ") ;
9              j ← j + 1 ;
10         fin tant que
11         aller_à_la_ligne ;
12     fin pour
13 fin
```

Exercice 10.b

```
1  Programme exo10b
2  VAR i, j : Entier ;
3  début
4      pour i de 1 à 9 pas 1 faire
5          Écrire(i, " : " ) ;
6          j ← 1 ;
7          tant que (j ≤ 10) faire
8              Écrire(i * j, " ") ;
9              j ← j + 1 ;
10         fin tant que
11         aller_à_la_ligne ;
12     fin pour
13 fin
```

Pour i valant d'abord 1 puis 2 ... puis 9 :

- affiche la valeur de i suivi de la chaîne " : " sans passer à la ligne ;
- valeur 1 affectée à la variable j ;
- j=1, condition TQ Vrai donc itération : affiche la valeur i*1, un espace puis 2 affectée à j ;
- j=2, condition TQ Vrai donc itération : affiche la valeur i*2, un espace puis 3 affectée à j ;
- ...
- j=10, condition TQ Vrai donc itération : affiche la valeur i*10, un espace puis 11 affectée à j ;
- j=11, condition TQ Faux donc fin du tant que ;
- Passage à la ligne et passage à la prochaine valeur de i.

Exo 10

Exercice 10.b

```
1  Programme exo10b
2  VAR i, j : Entier ;
3  début
4      pour i de 1 à 9 pas 1 faire
5          Écrire(i, " : " ) ;
6          j ← 1;
7          tant que (j ≤ 10) faire
8              Écrire(i * j, " ") ;
9              j ← j + 1 ;
10         fin tant que
11         aller_à_la_ligne ;
12     fin pour
13 fin
```

Pour i valant d'abord 1 puis 2 ... puis 9 :

- affiche i suivi de la chaîne " : " sans passer à la ligne ;
- affiche i*1, un espace, i*2, un espace, ... i*10, un espace ;
- Passage à la ligne et passage à la prochaine valeur de i.

On obtient donc les tables de multiplication suivantes :

1 : 1 2 3 4 5 6 7 8 9 10

2 : 2 4 6 8 10 12 14 16 18 20

3 : 3 6 9 12 15 18 21 24 27 30

...

9 : 9 18 27 36 45 54 63 72 81 90

Exercice 11

On veut dessiner à l'écran un 'L' et un 'I' sur plusieurs lignes à l'aide du caractère '*' comme dans les exemples ci-dessous :

```
*      *      *****      ***      *
*      *      *          *      ***
*      *      *          *      *****
*      ***     *          ***     *****
*              *          *      *****
*****        *****      *****
```

- 1 Écrire un algorithme qui demande à l'utilisateur la **lettre** à reproduire à l'écran, puis la **largeur** (qui doit être impaire) et la **hauteur** de la lettre, puis réalise le "dessin".
- 2 Écrire un algorithme qui dessine à l'écran un carré plein, rempli du caractère '*', d'une **largeur** saisie par l'utilisateur.
- 3 Écrire un algorithme qui affiche à l'écran un triangle forme d'"*", dont la **hauteur** est saisie au clavier, et qui respecte le format suivant (pour une hauteur de 6 par exemple, voir l'exemple ci-dessus).

Exo 11

Exercice 11-1

```
1  Programme exo11Q1
2      VAR lettre : Caractère ;
3          largeur, hauteur : Entier ;
4  début
5      répéter
6          Écrire("Choisir L ou I : ") ; Lire(lettre);
7      jusqu'à (lettre = 'L' OU lettre = 'I');
8      répéter
9          Écrire("Choisir une largeur positive impaire : ") ; Lire(largeur);
10     jusqu'à (largeur > 0 ET largeur MOD 2 = 1);
11     faire
12         Écrire("Choisir une hauteur positive : ") ; Lire(hauteur);
13     tant que (hauteur < 0);
14
15     si (lettre = "L") alors
16         ...
17     fin si
18     si (lettre = "I") alors
19         ...
20     fin si
21 fin
```

Exercice 11-1 suite

```
1  Programme exo11Q1
2      VAR lettre : Caractère ;
3          largeur, hauteur, k : Entier ;
4  début
5      ...
6      si (lettre = "L") alors
7
8          pour k de 1 à (hauteur - 1) pas 1 faire
9              Écrire("*"); aller_à_la_ligne ;
10         fin pour
11
12         pour k de 1 à largeur pas 1 faire
13             Écrire("*");
14         fin pour
15         aller_à_la_ligne ;
16     fin si
17     ...
18 fin
```

Exo 11

Exercice 11-1 et fin

```
1  Programme exo11Q1
2      VAR lettre : Caractère ;
3          largeur, hauteur, k, espace : Entier ;
4  début
5      ...
6      si (lettre = "I") alors
7          pour k de 1 à largeur pas 1 faire
8              Écrire("*");
9          fin pour
10         aller_à_la_ligne ;
11
12         espace ← ((largeur-1) DIV 2) ;
13         pour k de 1 à espace pas 1 faire
14             Écrire(" ");
15         fin pour
16         Écrire("*"); aller_à_la_ligne ;
17
18         pour k de 1 à largeur pas 1 faire
19             Écrire("*");
20         fin pour
21         aller_à_la_ligne ;
22     fin si
23 fin
```

Exercice 11-2

```
1  Programme exo11Q2
2      VAR largeur, ligne, colonne : Entier ;
3  début
4      faire
5          Écrire("Choisir une largeur positive : ") ;
6          Lire(largeur);
7      tant que (largeur < 0);
8
9      pour ligne de 1 à largeur pas 1 faire
10         pour colonne de 1 à largeur pas 1 faire
11             Écrire("*");
12         fin pour
13         aller_à_la_ligne ;
14     fin pour
15 fin
```

Exercice 11-3-étude du problème

```
      *      étage 1
     ***     étage 2
    *****  étage 3
   *         étage 4
  *         étage 5
 *         étage 6 ; fin donc hauteur 6
```

Commençons par noter que :

- Le nombre d'étoiles est de 1 pour le premier étage puis augmente de 2 à chaque étage.
- Le nombre d'étoiles sur un étage est donc : $1 + 2 \times (\text{numro_tage} - 1)$.
- La largeur maximale d'étoiles est donc sur le dernière étage : $1 + 2 \times (h - 1)$

Il faut réaliser un algorithme qui :

- 1 Demande la hauteur voulue h
- 2 Calcule la largeur maximale $lgmax = 1 + 2 \times (h - 1)$
- 3 Pour chaque *etage* de 1 à h
 - 1 Calcule le nombre d'étoiles à tracer sur l'étage : $nbe = 1 + 2 \times (\text{etage} - 1)$
 - 2 Calcule le nombre d'espaces blancs à placer devant puis derrière : $nbb = (lgm - nbe)/2$
 - 3 Ecrire nbb espaces blancs (devant) ;
 - 4 Ecrire nbe étoiles ;
 - 5 Ecrire nbb espaces blancs (derrière) [INUTILE] ;
 - 6 Passer à la ligne.

Exercice 11-3-étude du problème

```
      *      étage 1
     ***     étage 2
    *****  étage 3
   *         étage 4
  *          étage 5
 *           étage 6 ; fin donc hauteur 6
```

Commençons par noter que :

- Le nombre d'étoiles est de 1 pour le premier étage puis augmente de 2 à chaque étage.
- Le nombre d'étoiles sur un étage est donc : $1 + 2 \times (\text{numro_tage} - 1)$.
- La largeur maximale d'étoiles est donc sur le dernière étage : $1 + 2 \times (h - 1)$

Il faut réaliser un algorithme qui :

- 1 Demande la hauteur voulue h
- 2 Calcule la largeur maximale $lgmax = 1 + 2 \times (h - 1)$
- 3 Pour chaque *etage* de 1 à h
 - 1 Calcule le nombre d'étoiles à tracer sur l'étage : $nbe = 1 + 2 \times (\text{etage} - 1)$
 - 2 Calcule le nombre d'espaces blancs à placer devant puis derrière : $nbb = (lgm - nbe)/2$
 - 3 Ecrire nbb espaces blancs (devant) ;
 - 4 Ecrire nbe étoiles ;
 - 5 Ecrire nbb espaces blancs (derrière) [INUTILE] ;
 - 6 Passer à la ligne.

Exercice 11-3-étude du problème

```
      *      étage 1
     ***     étage 2
    *****  étage 3
   *         étage 4
  *          étage 5
 *           étage 6 ; fin donc hauteur 6
```

Commençons par noter que :

- Le nombre d'étoiles est de 1 pour le premier étage puis augmente de 2 à chaque étage.
- Le nombre d'étoiles sur un étage est donc : $1 + 2 \times (\text{numro_tage} - 1)$.
- La largeur maximale d'étoiles est donc sur le dernière étage : $1 + 2 \times (h - 1)$

Il faut réaliser un algorithme qui :

- 1 Demande la hauteur voulue h
- 2 Calcule la largeur maximale $lgmax = 1 + 2 \times (h - 1)$
- 3 Pour chaque *etage* de 1 à h
 - 1 Calcule le nombre d'étoiles à tracer sur l'étage : $nbe = 1 + 2 \times (\text{etage} - 1)$
 - 2 Calcule le nombre d'espaces blancs à placer devant puis derrière : $nbb = (lgm - nbe)/2$
 - 3 Ecrire nbb espaces blancs (devant) ;
 - 4 Ecrire nbe étoiles ;
 - 5 Ecrire nbb espaces blancs (derrière) [INUTILE] ;
 - 6 Passer à la ligne.

Exercice 11-3-étude du problème

```
      *      étage 1
     ***     étage 2
    *****  étage 3
   *         étage 4
  *         étage 5
 *         étage 6 ; fin donc hauteur 6
```

Commençons par noter que :

- Le nombre d'étoiles est de 1 pour le premier étage puis augmente de 2 à chaque étage.
- Le nombre d'étoiles sur un étage est donc : $1 + 2 \times (\text{numro_tage} - 1)$.
- La largeur maximale d'étoiles est donc sur le dernière étage : $1 + 2 \times (h - 1)$

Il faut réaliser un algorithme qui :

- 1 Demande la hauteur voulue h
- 2 Calcule la largeur maximale $lgmax = 1 + 2 \times (h - 1)$
- 3 Pour chaque *etage* de 1 à h
 - 1 Calcule le nombre d'étoiles à tracer sur l'étage : $nbe = 1 + 2 \times (\text{etage} - 1)$
 - 2 Calcule le nombre d'espaces blancs à placer devant puis derrière : $nbb = (lgm - nbe)/2$
 - 3 Ecrire nbb espaces blancs (devant) ;
 - 4 Ecrire nbe étoiles ;
 - 5 Ecrire nbb espaces blancs (derrière) [INUTILE] ;
 - 6 Passer à la ligne.

Exercice 11-3-étude du problème

```
      *      étage 1
     ***     étage 2
    *****  étage 3
   *         étage 4
  *          étage 5
 *           étage 6 ; fin donc hauteur 6
```

Commençons par noter que :

- Le nombre d'étoiles est de 1 pour le premier étage puis augmente de 2 à chaque étage.
- Le nombre d'étoiles sur un étage est donc : $1 + 2 \times (\text{numro_tage} - 1)$.
- La largeur maximale d'étoiles est donc sur le dernière étage : $1 + 2 \times (h - 1)$

Il faut réaliser un algorithme qui :

- 1 Demande la hauteur voulue h
- 2 Calcule la largeur maximale $lgmax = 1 + 2 \times (h - 1)$
- 3 Pour chaque *etage* de 1 à h
 - 1 Calcule le nombre d'étoiles à tracer sur l'étage : $nbe = 1 + 2 \times (\text{etage} - 1)$
 - 2 Calcule le nombre d'espaces blancs à placer devant puis derrière : $nbb = (lgm - nbe)/2$
 - 3 Ecrire nbb espaces blancs (devant) ;
 - 4 Ecrire nbe étoiles ;
 - 5 Ecrire nbb espaces blancs (derrière) [INUTILE] ;
 - 6 Passer à la ligne.

Exercice 11-3-étude du problème

```
      *      étage 1
     ***     étage 2
    *****  étage 3
   *         étage 4
  *          étage 5
 *           étage 6 ; fin donc hauteur 6
```

Commençons par noter que :

- Le nombre d'étoiles est de 1 pour le premier étage puis augmente de 2 à chaque étage.
- Le nombre d'étoiles sur un étage est donc : $1 + 2 \times (\text{numro_tage} - 1)$.
- La largeur maximale d'étoiles est donc sur le dernière étage : $1 + 2 \times (h - 1)$

Il faut réaliser un algorithme qui :

- 1 Demande la hauteur voulue h
- 2 Calcule la largeur maximale $lgmax = 1 + 2 \times (h - 1)$
- 3 Pour chaque *etage* de 1 à h
 - 1 Calcule le nombre d'étoiles à tracer sur l'étage : $nbe = 1 + 2 \times (\text{etage} - 1)$
 - 2 Calcule le nombre d'espaces blancs à placer devant puis derrière : $nbb = (lgm - nbe)/2$
 - 3 Ecrire nbb espaces blancs (devant) ;
 - 4 Ecrire nbe étoiles ;
 - 5 Ecrire nbb espaces blancs (derrière) [INUTILE] ;
 - 6 Passer à la ligne.

Exercice 11-3-étude du problème

```
      *      étage 1
     ***     étage 2
    *****  étage 3
   *         étage 4
  *         étage 5
 *         étage 6 ; fin donc hauteur 6
```

Commençons par noter que :

- Le nombre d'étoiles est de 1 pour le premier étage puis augmente de 2 à chaque étage.
- Le nombre d'étoiles sur un étage est donc : $1 + 2 \times (\text{numro_tage} - 1)$.
- La largeur maximale d'étoiles est donc sur le dernière étage : $1 + 2 \times (h - 1)$

Il faut réaliser un algorithme qui :

- 1 Demande la hauteur voulue h
- 2 Calcule la largeur maximale $lgmax = 1 + 2 \times (h - 1)$
- 3 Pour chaque *etage* de 1 à h
 - 1 Calcule le nombre d'étoiles à tracer sur l'étage : $nbe = 1 + 2 \times (\text{etage} - 1)$
 - 2 Calcule le nombre d'espaces blancs à placer devant puis derrière : $nbb = (lgm - nbe)/2$
 - 3 Ecrire nbb espaces blancs (devant) ;
 - 4 Ecrire nbe étoiles ;
 - 5 Ecrire nbb espaces blancs (derrière) [INUTILE] ;
 - 6 Passer à la ligne.

Exo 11

Exercice 11-3-algo

```
1  Programme exo11Q3
2      VAR h, lgmax, etage, nbe, nbb, k : Entier ;
3  début
4      répéter                                     /* Etape 1 */
5          Écrire("Choisir une hauteur positive 1+ : ") ; Lire(h);
6      jusqu'à (h>0);
7
8      lgmax ← 1 + 2 × (h - 1) ;                     /* Etape 2 */
9
10     pour etage de 1 à h pas 1 faire                /* Etape 3 */
11         nbe ← 1 + 2 × (etage - 1) ;
12         nbb ← (lgm - nbe) DIV 2 ;
13         pour k de 1 à nbb pas 1 faire
14             Écrire(" ");
15         fin pour
16         pour k de 1 à nbe pas 1 faire
17             Écrire("*");
18         fin pour
19         aller_à_la_ligne ;
20     fin pour
21 fin
```

Une serre agricole utilise des capteurs pour surveiller les températures tout au long de la journée afin de garantir des conditions optimales pour la croissance des plantes. La responsable de la serre souhaite savoir s'il y a des écarts critiques par rapport aux températures idéales et souhaite calculer la moyenne des températures mesurées sur une journée. On suppose qu'un relevé de températures est fait toutes les heures dans la serre.

- ❶ Écrire un programme permettant de saisir les températures chaque heure d'une journée (de 24h), d'en calculer et d'en afficher la moyenne en utilisant le schéma répétitif pour.
- ❷ Réécrire le programme en modifiant le schéma répétitif utilisé (la boucle).
- ❸ Écrire un programme permettant de saisir les températures d'une journée, d'en définir et d'en afficher la température minimum et la température maximum, en utilisant un autre schéma répétitif qu'aux questions 1 et 2.
- ❹ Écrire un programme permettant de saisir les températures d'une journée, la température idéale ainsi que l'écart acceptable de température choisi par la responsable de la serre. Lorsque l'écart entre une température saisie et la température idéale est supérieur à l'écart acceptable souhaité par la responsable, un message sera affiché avec l'heure de la journée concernée et une précision "température trop basse" ou "température trop élevée".
- ❺ Compléter le programme précédent pour compter le nombre d'heures concernées par un dépassement de l'écart acceptable.

Exercice 12-1

```
1  Programme exo12Q1
2      VAR somme, moyenne, temperature : Réel ;
3          heure : Entier;
4  début
5
6      somme ← 0 ;
7
8      pour heure de 0 à 23 pas 1 faire
9          Écrire("Entrer la température mesurée à ", heure, " h : ") ;
10         Lire(temperature);
11         somme ← somme + temperature ;
12      fin pour
13
14      moyenne ← somme / 24 ;
15      Écrire("Température moyenne : ", moyenne) ;
16
17  fin
```

Exercice 12-2 (avec choix d'un Tant que...faire)

```
1  Programme exo12Q2
2      VAR somme, moyenne, temperature : Réel ;
3          heure : Entier;
4  début
5
6      somme ← 0 ;
7      heure ← 0 ;
8
9      tant que (heure < 24) faire
10         Écrire("Entrer la température mesurée à ", heure, " h : ") ;
11         Lire(temperature);
12         somme ← somme + temperature ;
13         heure ← heure + 1 ;
14     fin tant que
15
16     moyenne ← somme / 24 ;
17     Écrire("Température moyenne : ", moyenne) ;
18
19 \motcle{fin}
```

Exercice 12-3 (avec choix d'un répéter...jusqu'à)

```
1  Programme exo12Q3
2      VAR tmin, tmax, temperature : Réel ;
3          heure : Entier;
4  début
5
6      heure ← 0 ;
7      Écrire("Entrer la température mesurée à ", heure, " h : ") ;
8      Lire(temperature);
9      tmin ← temperature ; tmax ← temperature ;
10
11     répéter
12         heure ← heure + 1 ;
13         Écrire("Entrer la température mesurée à ", heure, " h : ") ;
14         Lire(temperature);
15         si (temperature > tmax) alors tmax ← temperature ; fin si
16         si (temperature < tmin) alors tmin ← temperature ; fin si
17     jusqu'à (heure ≥ 23);
18
19     Écrire("Température minimale : ", tmin) ;
20     Écrire("Température maximale : ", tmax) ;
21 fin
```

Exercice 12-4

Écrire un programme permettant de saisir **les températures** d'une journée, la **température idéale** ainsi que l'**écart acceptable** de température choisi par la responsable de la serre. Lorsque l'écart entre une température saisie et la température idéale est supérieur à l'écart acceptable souhaité par la responsable, un message sera affiché avec l'heure de la journée concernée et une précision "température trop basse" ou "température trop élevée".

Attention, si on gère la saisie dans cet ordre, on ne pourrait pas agir puisque chaque nouvelle température saisie écraserait la précédente ! Pour le moment, on ne pourra pas séparer la saisie des données et leurs traitements.

Ou alors, il faudrait 24 variables de températures. Mais l'algorithme ne permettrait plus alors aucune automatisation particulière avec les boucles. Deux solutions :

- 1 Inverser la saisie des données -> solution choisie ici ;
- 2 Utiliser une structure de données type tableau pour séparer saisie / traitement et garder de l'automatisation ;

Exercice 12-4

```
1  Programme exo12Q3
2      VAR tOptimale, ecartAcc, ecart, temperature : Réel ;
3          heure : Entier;
4  début
5
6      Écrire("Entrer la température optimale : ") ; Lire(tOptimale);
7      Écrire("Entrer l'écart acceptable : ") ; Lire(ecartAcc);
8
9      heure ← 0 ;
10
11     répéter
12         Écrire("Entrer la température mesurée à ", heure, " h : ") ;
13         Lire(temperature);
14         ecart ← temperature - tOptimale;
15         si (ecart > ecartAcc) alors
16             Écrire("Température trop haute à ", heure, " h.") ;
17         fin si
18         si (ecart < (- ecartAcc)) alors
19             Écrire("Température trop basse à ", heure, " h.") ;
20         fin si
21         heure ← heure + 1 ;
22     jusqu'à (heure ≥ 24);
23
24 fin
```

Exo 12

Exercice 12-5

```
1  Programme exo12Q3
2      VAR tOptimale, ecartAcc, ecart, temperature : Réel ;
3          heure, nb : Entier;
4  début
5
6      Écrire("Entrer la température optimale : ") ; Lire(tOptimale);
7      Écrire("Entrer l'écart acceptable : ") ; Lire(ecartAcc);
8
9      heure ← 0 ;
10     nb ← 0 ;
11
12     répéter
13         Écrire("Entrer la température mesurée à ", heure, " h : ") ;
14         Lire(temperature);
15         ecart ← temperature - tOptimale;
16         si (ecart > ecartAcc) alors
17             Écrire("Température trop haute à ", heure, " h.") ;
18             nb ← nb + 1 ;
19         fin si
20         si (ecart < (- ecartAcc)) alors
21             Écrire("Température trop basse à ", heure, " h.") ;
22             nb ← nb + 1 ;
23         fin si
24         heure ← heure + 1 ;
25     jusqu'à (heure ≥ 24);
26
27     Écrire("Nombre d'heures problématiques : ", nb) ;
28
29 fin
```

Bilan

- Connaître la structure du TQ...faire (0+ itération(s)) : condition de poursuite puis itération possible ;
- Connaître la structure du Faire..TQ (1+ itération(s)) : itération puis condition de poursuite ;
- Connaître la structure du Répéter..jusqu'à (1+ itération(s)) : itération puis condition d'arrêt ;
- Savoir transformer une condition d'arrêt en poursuite et inversement ;
- Pouvoir passer d'une des trois boucles à l'une des autres ;
- Connaître la structure du Pour...de...à...pas...faire ;
- Connaître la structure du Pour...de...bas...pas...faire ;
- Savoir passer d'un Pour à l'une des trois structures TQ, faire...TQ et répéter...jusqu'à ;
- Comprendre et savoir qu'on ne peut pas passer systématiquement d'un TQ/Jusqu'à à un Pour ;
- Savoir suivre le déroulé d'un algorithme contenant une boucle ;
- Savoir gérer les algorithmes comportant des structures imbriquées l'une dans l'autre ;
- Comprendre que la Primitive Ecrire peut ne pas imposer le passage à la ligne ;
- Comprendre que les mots clés et les points-virgules suffisent à comprendre un algo (mais qu'une écriture tabulée sur plusieurs lignes est néanmoins plus claire) ;
- ALGO : recherche du maximum, du minimum ;
- ALGO : recherche d'une moyenne ;

Pour la semaine prochaine

Avec Moodle :

- ➊ Récupérer le PDF du TD 3 ;
- ➋ Récupérer le PDF support de cours n° ;
- ➌ Revoir le TD, notamment les deux exercices en autonomie ;
- ➍ Réaliser **sérieusement** le QCM final de la leçon 3.
- ➎ Réviser pour l'interrogation ;

Exo supplémentaire

Pour s'entraîner à lire des algorithmes :
<https://execubot.fr/>

Résumé des trois boucles précédentes

[Faire...Tant ou Répéter...jusqu'à] —> [Tant Que...faire]

VERSION Tant que...faire <--

VERSION Faire...Tant que

VERSION Répéter...jusqu'à

faire

action1;

action2;

tant que (cond_poursuite);

répéter

action1;

action2;

jusqu'à (cond_arret);

Pour obtenir un Tant Que à partir d'un des deux autres :

Résumé des trois boucles précédentes

[Faire...Tant ou Répéter...jusqu'à] —> [Tant Que...faire]

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

Pour obtenir un Tant Que à partir d'un des deux autres :

- Au besoin, transformation de la condition d'arrêt en condition de poursuite.

Attention : $x \leq y$ deviendrait $x > y$ par exemple

- Rajout du bloc préliminaire d'actions avant le Tant Que (de façon à pouvoir faire ces actions une fois ou plus);

Si on peut prouver que la condition de poursuite est nécessairement **Vrai** lors du premier passage, on peut omettre de placer ce bloc préliminaire.

CONCLUSION : transformation facile si il y a forcément au moins une itération imposée.