Les bases de l'algorithmique
Corrigé

Objectifs

— Comprendre et utiliser les instructions élémentaires et les structures de contrôle

Exercice 1 : Exécution d'un programme

On considère le programme suivant qui affiche « Premier » ou « Non premier » en réponse à un entier naturel saisi par l'utilisateur.

```
1  Programme Premier Est
2  Variables
3      N : Entier
4      C : Entier
5      P : Booléen
6  Début
7      Écrire("Un␣entier␣?␣")
8      Lire(N)
9      C <- 2
10     P <- N >= 2
11     TantQue P Et Alors C * C < N Faire
12         Si N mod C = 0 Alors
13             P <- FAUX
14         Sinon
15             C <- C + 1
16         FinSi
17     FinTQ
18     Si P Alors
19         Écrire("Premier")
20     Sinon
21         Écrire("Non␣premier")
22     FinSi
23 Fin.
```

1. Indiquer comment s'exécute ce programme quand l'utilisateur saisit 2. On indiquera dans la grille, à chaque étape de l'exécution, la nouvelle valeur des variables modifiées et l'instruction suivante à exécuter.

Solution :

instr.	7	8	9	10	11	18	19	23																
N	?		2																					
C	?			2																				
P	?				V																			

Sur le terminal on a :

```
Un entier ? 2
Premier
```

On remarque que les structures de contrôle, conditionnelles ou répétitions, ne modifient pas l'état de la mémoire. Leur but est de déterminer l'instruction suivante à exécuter.

L'instruction `Écrire` ne modifie pas l'état du programme mais a un effet de bord sur le terminal : affichage des expressions en paramètre.

Les seules instructions qui modifient la mémoire sont les affectations... et les saisies qui cachent en fait une affectation (on affecte la variable avec la valeur obtenue à partir des caractères saisis au clavier).

2. Même question quand l'utilisateur saisit 13.

Solution :

instr.	7	8	9	10	11	12	15	11	12	15	11	18	19	23										
N	?		13																					
C	?			2				3			4													
P	?				V																			

Sur le terminal on a :

```
Un entier ? 13
Premier
```

3. Même question quand l'utilisateur saisit 303.

Solution :

instr.	7	8	9	10	11	12	15	11	12	13	11	18	21	23										
N	?		303																					
C	?			2				3																
P	?				V						F													

Sur le terminal on a :

```
Un entier ? 303
Non premier
```

4. Qu'est-ce que les exécutions précédentes nous permettent de conclure concernant la correction de ce programme.

Solution : On peut en déduire que le programme a fonctionné sur les exemples précédents. On ne peut pas pour autant en déduire que le programme est correct. La question suivante en est la preuve.

5. Exécuter le programme, l'utilisateur saisissant 25.

Solution :

instr.	7	8	9	10	11	12	15	11	12	15	11	12	15	11	18	19	23		
N	?		25																
C	?			2				3			4		5						
P	?				V						F								

Sur le terminal on a :

```
Un entier ? 25
Premier
```

6. Que conclure de l'exécution précédente. Que doit-on faire ensuite ?
Solution : Le programme affiche que le nombre est premier alors qu'il ne l'est pas. 25 est divisible par 5 ($25 = 5^2$).

Le programme est donc faux.

Il faut donc :

1. corriger le programme,
2. vérifier avec 25 qu'il donne le bon résultat,
3. vérifier que les exécutions précédentes continuent à donner les résultats attendus (tests de non régression).

7. Quelles autres exécutions serait-il pertinent de faire ?
Solution : Il serait judicieux d'essayer 0 et 1 qui doivent être non premier (cas limite). Toujours dans les cas limites, il faudrait essayer des grands entiers. Dans ce cas, $C * C$ risque de provoquer un débordement des entiers. Il est donc préférable de réécrire l'inégalité en $C <= N/C$.

Enfin, on pourrait essayer avec un nombre négatif. On peut alors soit signaler que l'on ne traite que les entiers naturels, soit travailler sur la valeur absolue de l'entier.

On pourrait améliorer l'efficacité du programme. Par exemple, une fois qu'on a testé 2, il est inutile d'essayer le nombre pairs (4, 6, 8, 10, etc.). Il suffirait donc d'essayer les nombres impairs 3, 5, 7, 9, etc.

Pour aller plus loin, on sait qu'un nombre est premier s'il n'est pas divisible par un nombre premier plus petit que lui. Il suffirait donc de stocker les nombres premiers déjà calculés et les essayer. Ceci suppose de savoir mémoriser ces nombres premiers. Pour l'instant, on ne sait pas faire...

Ces évolutions du programme nous amèneraient certainement à identifier de nouveaux tests à faire en plus de ceux déjà identifiés.

8. Pourquoi les identifiants sont-ils mal choisis ? Proposer de nouveaux identifiants.
Solution : Il faut éviter les identifiants limités à une lettre car ils ne sont pas significatifs. Ici on pourrait prendre `Nombre` au lieu de `N`, `Candidat` au lieu de `C` et `Est_Premier` au lieu de `P`.

9. Quelles informations importantes manque-t-il dans le programme précédent ?
Solution : Les commentaires qui expliquent la structure du programme :

```
-- Demander un entier naturel
-- Déterminer la primalité de cet entier
-- Afficher la primalité de cet entier
```

On pourrait aussi expliquer la signification des conditions, par exemple « N divisible par D » et « D est un diviseur possible ».

instr.	7																								
N	?																								
C	?																								
P	?																								

instr.	7																								
N	?																								
C	?																								
P	?																								

instr.	7																								
N	?																								
C	?																								
P	?																								

instr.	7																								
N	?																								
C	?																								
P	?																								

instr.	7																								
N	?																								
C	?																								
P	?																								

instr.	7																								
N	?																								
C	?																								
P	?																								

instr.	7																								
N	?																								
C	?																								
P	?																								

instr.	7																								
N	?																								
C	?																								
P	?																								

instr.	7																								
N	?																								
C	?																								
P	?																								

instr.	7																							
N	?																							
C	?																							
P	?																							
instr.	7																							
N	?																							
C	?																							
P	?																							

Exercice 2 : TantQue et Répéter

Écrire la répétition **`Répéter`** à partir du **`TantQue`** et réciproquement.

Solution : Écrire un **`Répéter`** en utilisant le **`TantQue`** :

```
Répéter
    séquence
JusquÀ condition
```

devient :

```
séquence
TantQue Non condition Faire
    séquence
FinTQ
```

Inversement,

```
TantQue condition Faire
    séquence
FinTQ
```

devient :

```
Si condition Alors
    Répéter
        séquence
    JusquÀ Non condition
FinSi
```

Remarque : On constate que **`TantQue`** et **`Répéter`** sont donc équivalents. On peut réécrire l'un avec l'autre.

Cependant, dans les deux cas, la formulation initiale est plus concise (et donc plus facile à lire et comprendre). Aussi, si on se retrouve face à la deuxième formulation, on la replacera par la première !

Notons que si les initialisations faites avant le **`TantQue`** garantissent que la condition est initialement vrai (et donc que l'on va passer dans la boucle), c'est une indication qu'il aurait peut-être fallu écrire un **`Répéter`**.

Exercice 3 : TantQue et Pour

1. Donner la forme générale d'un **Pour**.

Solution : On se limite au cas où le pas est positif.

La forme générale du Pour est :

```
-- { V_Pas > 0 }
Pour I De V_Début À V_Fin Pas V_Pas Faire
    Instructions
FinPour
```

2. Dans une répétition **Pour**, peut-on modifier la variable de boucle ?

Solution : Non, ceci est interdit par le compilateur. Il affiche le message « *assignment to loop parameter not allowed* ».

3. Dans une répétition **Pour** où la fin est donnée par une variable, peut-on modifier cette variable, par exemple pour terminer la répétition ?

Solution : On peut modifier la valeur de cette variable mais ceci n'aura pas d'effet sur l'exécution de la boucle. En effet, l'intervalle sur lequel la variable de boucle prendra ses valeurs est évalué une et une seule fois au début de la boucle.

4. Écrire la répétition **Pour** à partir du **TantQue**.

Solution : Elle peut se réécrire avec un **TantQue** de la manière suivante :

```
I <- V_Début
Copie_V_Fin <- V_Fin
Incrément <- V_Pas
TantQue I <= Copie_V_Fin Faire
    Instructions
    I <- I + Incrément
FinTQ
```

Les variabes `Copie_V_Fin` et `Incrément` ont été introduites pour le cas où les `Instructions` modifieraient `V_Fin` ou `V_Pas`. Ce sont bien sûr de nouvelles variables ; leur nom doit être différent des variables utilisées dans le programme.

Remarque : Il faudrait adapter dans le cas où le pas est négatif !

5. Peut-on réécrire un **TantQue** avec un **Pour** ?

Solution : La réponse est non car on ne sait pas quand le condition deviendra fausse dans la cas général. On ne peut donc pas définir l'intervalle nécessaire au **Pour** : on ne connait pas *a priori* le nombre de fois que devront être exécutées les instructions de la boucle. Un **TantQue** pourrait ne jamais s'arrêter. Un **Pour** s'arrêtera toujours.

Exercice 4 Écrire un programme qui affiche le rang du premier terme négatif ou nul de la suite :

$$U_{n+1} = 1/2\, U_n - 3n \quad (1)$$
$$U_0 = a \quad (2)$$

Solution : Dans la solution de cet exercice, nous utilisons la méthode des raffinages qui sera l'objet du prochain cours. Ceci permet de donner un exemple de son application. Pour la réponse à cet exercice, c'est le programme qui en découle qui est attendu.

On applique donc la méthode des raffinages.

Comprendre le problème

```
R0 : Afficher le rang du premier terme d'une suite dont la valeur est négative.

Exemple :
    A = 10 => U0 = 10, U1 = 5, U2 =  -0.5 donc la réponse est RANG = 2.
```

Trouver une solution informelle On calcule les termes successifs de la suite. On s'arrête dès qu'un terme négatif est obtenu. Son rang est la réponse cherchée.

Formaliser cette solution Si on sait que A est positif, on peut écrire un Répéter car il faudra au moins calculer un terme de plus. Si A est négatif ou si on n'a aucune hypothèse sur A, il faut utiliser un TantQue car le premier terme et donc le rang 0 est (peut-être) la solution.

Dans le doute, on choisit ici la deuxième solution (pas d'hypothèse sur A).

```
R1 : Comment « Afficher le rang du premier terme... »
    Demander la valeur de A                      A : out Réel
    Déterminer le rang du premier terme négatif  A : in ; Rang : out Entier
    Afficher ce rang                             Rang : in

R2 : Comment « Déterminer le rang du premier terme négatif » ?
out Entier
    Terme <- A
    Rang <- 0
    TantQue Terme >= 0 Faire    -- pas encore de terme négatif
        Terme <- Terme / 2.0 - 3.0 * Réel (Rang)
        Rang <- Rang + 1
    FinTQ
```

```
-- Afficher le Rang du premier Terme d'une suite dont la valeur est négative.
-- Auteur : Xavier Crégut <nom@n7.fr>

with Text_IO; use Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
with Ada.Float_Text_IO; use Ada.Float_Text_IO;

procedure suite_premier_negatif is
```

```
9       Rang  : Integer;    -- Rang du Terme courant de la série
10      Terme : Float;       -- Terme de la série de Rang 'Rang'
11      A     : Float;       -- l'élément 'a' de la définition de la suite
12  begin
13      -- saisir la valeur de A
14      Put ("Valeur de A : ");
15      Get (A);
16
17      -- calculer le Rang du premier Terme négatif ou nul de la suite
18      -- initialiser avec le premier Terme
19      Rang := 0;
20      Terme := A;
21
22      while Terme > 0.0 loop
23          -- calculer le Terme suivant
24          Terme := Terme / 2.0 - 3.0 * Float (Rang);
25          Rang := Rang + 1;
26      end loop;
27      -- { Terme <= 0 Et "le Terme précédent était >= 0" }
28
29      -- Afficher le résultat
30      Put ("Rang du premier terme négatif ou nul : ");
31      Put (Rang, 1);
32      New_Line;
33      Put ("Terme correspondant : ");
34      Put (Terme, Exp => 0);
35      New_Line;
36  end suite_premier_negatif;
```

Exercice 5 Écrire en langage Ada le programme correspondant à l'algorithme précédent.
Solution : Voir fin du corrigé de l'exercice précédent.

Exercice 6 : Drone commandé par un menu
On s'intéresse à la commande à distance d'un drone qui se déplace uniquement selon son axe vertical. La commande à distance se fait via un « menu textuel », un affichage du type suivant, permettant de prendre en compte les choix successifs de l'utilisateur.

```
Altitude : 3

Que faire ?
    d -- Démarrer
    m -- Monter
    s -- Descendre
    q -- Quitter
Votre choix : _
```

Les propriétés suivantes doivent être satisfaites par le programme gérant le menu textuel :

1. Le drone ne peut monter et descendre que s'il a été démarré au préalable.

2. En fonctionnement nominal, l'action `monter` augmente l'altitude du drone d'une unité et l'action `descendre` la diminue d'une unité.
3. Le drone ne peut pas descendre à une altitude négative.
4. Le programme affiche le menu et traite chaque choix de l'utilisateur du programme jusqu'à ce que l'utilisateur choisisse de quitter (avec l'option 'q') ou jusqu'à ce que le drone monte à une altitude supérieure ou égale à 5 (le drone est alors hors de portée).
5. L'utilisateur pourra utiliser les minuscules ou les majuscules.

Écrire le programme correspondant. Utiliser des variables booléennes est conseillé.

Solution :

```
with Ada.Text_IO;
use Ada.Text_IO;
with Ada.Integer_Text_IO;
use Ada.Integer_Text_IO;

-- Piloter un drone au moyen d'un menu textuel.
procedure Drone is
    ALTITUDE_MAX : constant Integer := 5;    -- altitude à partir de laquelle
                             --  le drone n'est plus à porter (et donc perdu)

    Altitude  : Integer;    -- l'altitude du drone
    En_Route  : Boolean;    -- Est-ce que le drone a été démarré ?
    Est_Perdu : Boolean;    -- Est-ce que le drone est perdu ?

    Choix: Character;    -- le choix de l'utilisateur
    Quitter: Boolean;    -- Est-ce que l'utilisateur veut quitter ?
begin
    -- Initialiser le drone
    En_Route := False;
    Est_Perdu := False;
    Altitude := 0;

    Quitter := False;
    loop
        -- Afficher l'altitude du drone
        New_Line;
        Put ("Altitude : ");
        Put (Altitude, 1);
        New_Line;

        -- Afficher le menu
        New_Line;
        Put_Line ("Que faire ?");
        Put_Line ("    d -- Démarrer");
        Put_Line ("    m -- Monter");
        Put_Line ("    s -- Descendre");
        Put_Line ("    q -- Quitter");

```

```
39          -- Demander le choix de l'utilsateur
40          Put ("Votre choix : ");
41          Get (Choix);
42          Skip_Line;
43
44          -- Traiter le choix de l'utilisateur
45          case Choix is
46
47              when 'd' | 'D' =>    -- Démarrer
48                  -- Mettre de drone en route
49                  En_Route := True;
50
51              when 'm' | 'M' =>    -- Monter
52                  -- Faire monter le drone
53                  if En_Route then
54                      Altitude := Altitude + 1;
55                  else
56                      Put_Line ("Le drone n'est pas démarré.");
57                  end if;
58                  Est_Perdu := Altitude >= ALTITUDE_MAX;
59
60              when 's' | 'S' =>    -- Descendre
61                  -- Faire descendre le drone
62                  if En_Route then
63                      if Altitude > 0 then
64                          Altitude := Altitude - 1;
65                      else
66                          Put_Line ("Le drône est déjà posé.");
67                      end if;
68                  else
69                      Put_Line ("Le drone n'est pas démarré.");
70                  end if;
71
72              when 'q' | 'Q' | '0' =>    -- Quitter
73                  Quitter := True;
74
75              when others =>        -- Ordre inconnu
76                  Put_Line ("Je n'ai pas compris !.");
77
78          end case;
79          exit when Quitter or else Est_Perdu;
80      end loop;
81
82      -- Afficher les raisons de l'arrêt
83      New_Line;
84      if Est_Perdu then
85          Put_Line ("Le drone est hors de portée... et donc perdu !");
86      else
87          Put_Line ("Au revoir...");
88      end if;
```

```
89  end Drone;
```

D'autres solutions sont possibles. Par exemple, on pourrait attendre que le drone soit démarré puis gérer les actions sur le drone.

On pourrait aussi commencer par tester l'état du drone et ensuite l'ordre fourni par l'utilisateur...