# Raffinages
# Corrigé



## Exercice 1 : Des raffinages vers un programme

Écrire le programme Ada qui correspond aux raffinages du listing 1.

Listing 1 – Des raffinages pour le calcul des PGCD

```
R0 : Afficher le pgcd de deux entiers strictement positifs

Exemples :

   a    b       pgcd
-------------------------------------------------------------
   2     4  ==>  2                 -- cas nominal (a < b)
  20    15  ==>  5                 -- cas nominal (a > b)
  20    20  ==>  20                -- cas nominal (a = b)
  20     1  ==>  1                 -- cas limite  (b = 1)
   1     1  ==>  1                 -- cas limite  (a = b = 1)
   0     4  ==>  Erreur : a <= 0   -- cas d'erreur (robustesse)
   4    -4  ==>  Erreur : b <= 0   -- cas d'erreur (robustesse)

R1 : Comment « Afficher le pgcd de deux entiers positifs » ?
    Demander deux entiers a et b            a, b: out
    { (a > 0) Et (b > 0)  -- les deux entiers sont strictement positifs }
    Déterminer le pgcd de a et b            a, b: in; pgcd: out
    Afficher le pgcd                        pgcd: in

R2 : Comment « Déterminer le pgcd de a et b » ?
    na <- a     -- variables auxiliaires car a et b sont en in
    nb <- b     -- et ne doivent donc pas être modifiées.
    TantQue na et nb différents Faire                          na, nb: in
        Soustraire au plus grand le plus petit          na, nb: in out
    FinTQ
    pgcd <- na  -- pgcd était en out, il doit être initialisé.

R2 : Comment « Afficher le pgcd » ?
    Écrire ("pgcd = ")
    Écrire (pgcd)

R2 : Comment « Demander deux entiers » ?
    -- Attention : la spécification n'est pas respectée car cette saisie
    -- ne garantit pas que les deux entiers seront strictement positifs
    -- Ce raffinage n'est donc pas correct et le programme ne sera pas robuste !
    Ecrire ("A et B ? ")
    Lire (a)
    Lire (b)
```

```
40
41 R3 : Comment [déterminer] « na et nb différents » ?
42     Résultat <- na <> nb
43
44 R3 : Comment « Soustraire au plus grand le plus petit » ?
45     Si na > nb Alors
46         na <- na - nb
47     Sinon
48         nb <- nb - na
49     FinSi
```

**Solution :** Le principe est expliqué en cours...

Le R0 est placé avant le programme principal. C'est le commentaire général qui donne l'objectif du programme.

Les variables sont déduites des flots de données. En même temps que l'on construit le raffinage, il faudrait alimenter le dictionnaire des données : pour chaque donnée identifiée, on ajoute au dictionnaire son rôle (qui deviendra son commentaire), son nom et son type.

Les instructions du programme son déduites des raffinages en commençant naturellement par le R1. Chaque action complexe devient un commentaire qui est suivi par la transcription de sa décomposition. Pour une expression complexe, on met le commentaire en fin de la ligne qui contient la définition de cette expression complexe.

Les exemples pourraiennt être écrits à la fin du programme pour en garder un trace mais il est préférable d'essayer de les automatiser.

Voici le programme correspondant.

```
1  with Ada.Text_IO;          use Ada.Text_IO;
2  with Ada.Integer_Text_IO;  use Ada.Integer_Text_IO;
3
4  -- Afficher le pgcd de deux entiers strictement positifs
5  procedure Pgcd_Euclide is
6
7      A, B: Integer;              -- deux entiers saisis au clavier
8      Pgcd: Integer;              -- le pgcd de A et B
9      Na, Nb: Integer;    -- utilisées pour le calcul du pgcd
10
11 begin
12     -- Demander deux entiers
13     --   | Attention : la spécification n'est pas respectée car cette saisie
14     --   | ne garantit pas que les deux entiers seront strictement positifs
15     --   | Ce raffinage n'est donc pas correct et le programme ne sera pas robuste !
16     Put ("A et B ? ");
17     Get (A);
18     Get (B);
19
20     pragma Assert (A > 0);
21     pragma Assert (B > 0);
22
23     -- Déterminer le pgcd de A et B
24     NA := A;    -- variables auxiliaires : ceci permet
25     NB := B;    -- de conserver les valeurs saisies
26     while NA /= NB loop         -- NA et NB différents
27         -- Soustraire au plus grand le plus petit
28         if NA > NB then
29             NA := NA - NB;
30         else
31             NB := NB - NA;
```

```
32                 end if;
33             end loop;
34         Pgcd := NA;
35
36         -- Afficher le pgcd
37         Put ("pgcd = ");
38         Put (Pgcd, 1);
39     end Pgcd_Euclide;
```

**Remarques :**

1. L'expression complexe n'apparaîtra certainement pas dans un raffinage. On pourrait directement mettre l'expression correspondante. En tout état de cause, on ne la mettrait pas en commentaire dans le programme car elle paraphrase l'expression sans plus value.
2. L'assertion entre accolade a été traduite en Ada par deux `pragma Assert`. Ils seront évalués pendant l'exécution du programme à condition d'avoir utiliser l'option `-gnata`. Ils provoqueront l'arrêt du programme si l'expression est fausse. Ces assertions peuvent être désactivées lors de la compilation du programme (non utilisation de l'option `-gnata`). Faire deux assertions plutôt qu'une seule permet d'avoir une indication plus claire en cas d'assertion non vérifiée : on saura si le problème est détecté sur A ou B.

### Exercice 2 : Retrouver un raffinage

Retrouver les raffinages qui sont à l'origine du programme du listing 2. On ne donnera pas le raffinage des actions complexes qui ne contiennent que des actions élémentaires.

Listing 2 – Programme Ada : piloter un drone

```
1  with Ada.Text_IO;           use Ada.Text_IO;
2  with Ada.Integer_Text_IO;   use Ada.Integer_Text_IO;
3
4  -- Piloter un drone au moyen d'un menu textuel.
5  procedure Drone is
6      LIMITE_PORTEE : constant Integer := 5;  -- altitude à partir de laquelle
7                               --  le drone n'est plus à porter (et donc perdu)
8
9      Altitude  : Integer;    -- l'altitude du drone
10     En_Route  : Boolean;    -- Est-ce que le drone a été démarré ?
11     Est_Perdu : Boolean;    -- Est-ce que le drone est perdu ?
12
13     Choix: Character;   -- le choix de l'utilisateur
14     Quitter: Boolean;   -- Est-ce que l'utilisateur veut quitter ?
15 begin
16     -- Initialiser le drone
17     En_Route := False;
18     Est_Perdu := False;
19     Altitude := 0;
20
21     Quitter := False;
22     loop
23         -- Afficher l'altitude du drone
24         New_Line;
25         Put ("Altitude : ");
26         Put (Altitude, 1);
27         New_Line;
28
29         -- Afficher le menu
```

```
30          New_Line;
31          Put_Line ("Que␣faire␣?");
32          Put_Line ("␣␣␣␣d␣--␣Démarrer");
33          Put_Line ("␣␣␣␣m␣--␣Monter");
34          Put_Line ("␣␣␣␣s␣--␣Descendre");
35          Put_Line ("␣␣␣␣q␣--␣Quitter");
36
37          -- Demander le choix de l'utilisateur
38          Put ("Votre␣choix␣:␣");
39          Get (Choix);
40          Skip_Line;
41
42          -- Traiter le choix de l'utilisateur
43          case Choix is
44
45              when 'd' | 'D' =>   -- Démarrer
46                  -- Mettre le drone en route
47                  En_Route := True;
48
49              when 'm' | 'M' =>   -- Monter
50                  -- Faire monter le drone
51                  if En_Route then
52                      Altitude := Altitude + 1;
53                  else
54                      Put_Line ("Le␣drone␣n'est␣pas␣démarré.");
55                  end if;
56                  Est_Perdu := Altitude >= LIMITE_PORTEE;
57
58              when 's' | 'S' =>   -- Descendre
59                  -- Faire descendre le drone
60                  if En_Route then
61                      if Altitude > 0 then
62                          Altitude := Altitude - 1;
63                      else
64                          Put_Line ("Le␣drone␣est␣déjà␣posé.");
65                      end if;
66                  else
67                      Put_Line ("Le␣drone␣n'est␣pas␣démarré.");
68                  end if;
69
70              when 'q' | 'Q' | '0' => -- Quitter
71                  Quitter := True;
72
73              when others =>      -- Ordre inconnu
74                  Put_Line ("Je␣n'ai␣pas␣compris␣!");
75
76          end case;
77          exit when Quitter or else Est_Perdu;
78      end loop;
79
80      -- Afficher les raisons de l'arrêt
81      New_Line;
82      if Est_Perdu then
83          Put_Line ("Le␣drone␣est␣hors␣de␣portée...␣et␣donc␣perdu␣!");
84      elsif not En_Route then
85          Put_Line ("Vous␣n'avez␣pas␣réussi␣à␣le␣mettre␣en␣route␣?");
86      else
87          Put_Line ("Au␣revoir...");
88      end if;
```

```
89 end Drone;
```

**Solution :** Partant du programme, pour retrouver les raffinages, il faut s'intéresser aux commentaires qui vont devenir des actions complexes (ou expressions complexes) et vont structurer les instructions.

On obtient alors les raffinages suivants.

```
1  R0 : Piloter un drone au moyen d'un menu textuel.
2
3  Exemples : Omis car non présents dans le programme fourni.
4
5  R1 : Comment « Piloter un drone au moyen d'un menu textuel. » ?
6      Initialiser le drone        En_Route, Est_Perdu: out Booléen ; Altitude: out Entier
7      Quitter <- False            Quitter: out Booléen
8      Répéter
9          Afficher l'altitude du drone            Altitude: in
10         Afficher le menu
11         Demander le choix de l'utilisateur      Choix: out Caractère
12         Traiter le choix de l'utilisateur       Choix: in; Altitude, En_Route: in out; Quitter, E
13     JusquÀ Quitter OuBien Est_Perdu
14     Afficher les raisons de l'arrêt             Est_Perdu, En_Route: in
15
16 R2 : Comment « Initialiser le drone » ?
17     En_Route <- False
18     Est_Perdu <- False
19     Altitude <- 0
20
21 R2 : Comment « Afficher l'altitude du drone » ?
22     Ecrire ("Atitude : ", 1)
23
24 R2 : Comment « Afficher le menu » ?
25     Ecrire ("Que faire ?")
26     Ecrire ("    d -- Démarrer")
27     Ecrire ("    m -- Monter")
28     Ecrire ("    s -- Descendre")
29     Ecrire ("    q -- Quitter")
30
31 R2 : Comment « Demander le choix de l'utilisateur » ?
32     Ecrire ("Votre choix : ")
33     Lire (Choix)
34
35 R2 : Comment « Traiter le choix de l'utilisateur » ?
36     Selon Choix Faire
37         'd', 'D' => Mettre en route le drone
38         'm', 'M' => Faire monter le drone
39         's', 'S' => Faire descendre le drone
40         'q', 'Q' => Quitter
41         Autres   => Ecrire ("Je n'ai pas compris !")
42     FinSelon
43
44 R2 : Comment « Afficher les raisons de l'arrêt » ?
45     Si Est_Perdu Alors
46         Ecrire ("Le drone est hors de portée... et donc perdu !")
47     Sinon Si Non En_Route Alors
48         Ecrire ("Vous n'avez pas réussi à le mettre en route ?")
49     Sinon
50         Ecrire ("Au revoir...")
51     FinSi
```

Le R1 peut être considéré comme un peu long. On pourrait ajouter une action complexe « Piloter le drone » mais elle aurait dû apparaître en commentaire dans le programme. Le nouveau R1 ainsi obtenu aurait été composé de 3 étapes avec la deuxième qui est beaucoup plus importante que les autres. La deuxième action complexe est très proche de R0 et on peut donc considérer que ce R1 n'apporte pas tellement plus d'information que le R0.

**Exercice 3 : Nombres parfaits et nombres amis**

Un entier naturel est dit *parfait* s'il est égal à la somme de ses diviseurs, lui exclu. Par exemple, 6 est un nombre parfait ($6 = 1 + 2 + 3$), 28 l'est aussi ($28 = 1 + 2 + 4 + 7 + 14$).

Deux nombres $N$ et $M$ sont dits *amis* si la somme des diviseurs de $M$ (en excluant $M$ lui-même) est égale à $N$ et la somme des diviseurs de $N$ (en excluant $N$ lui-même) est égale à $M$. Par exemple, 220 et 284 sont amis. En effet, la somme des diviseurs de 220 hors 220 est $1+2+4+5+10+11+20+22+44+55+110 = 284$ et la somme des diviseurs de 284 hors 284 est $1 + 2 + 4 + 71 + 142 = 220$.

Notre objectif est d'écrire deux sous-programmes. Le premier affiche dans l'ordre croissant et au fur et à mesure les nombres parfaits de 2 à un entier naturel donné. Le deuxième affiche au fur et à mesure tous les nombres amis (N, M) entre 2 et un entier naturel donné MAX tel que $0 < N \leq M \leq$ MAX.

**Attention :** L'énoncé a été un peu changé car il faisait par erreur référence aux sous-programmes qui n'étaient pas le but de l'exercice.

**1.** Écrire les raffinages des deux programmes demandés.

**Solution :** Commençons par le premier qui calcule les nombres parfaits.

```
1  R0 : Afficher les nombres parfaits compris entre 2 et Max, lu au clavier
2
3  R1 : Raffinage De « R0 »
4         | Pour n <- 2 JusquÀ n = Max Faire
5         |    | Si n est parfait Alors
6         |    |    | Afficher n
7         |    | FinSi
8         | FinPour
9
10 R2 : Raffinage De « n est parfait »
11        | Calculer la somme des diviseurs stricts de n     somme: out Entier
12        | Résultat <- n = somme
13
14 R3 : Raffinage De « Calculer la somme des diviseurs de n »
15        | somme <- 1
16        | Pour i <- 2 JusquÀ racine carrée de n Faire
17        |    | Si i diviseur de n Alors
18        |    |    | somme <- somme + i + (n Div i)
19        |    | FinSi
20        | FinPour
21        | Si n est un carré parfait Alors
22        |    | somme <- somme - racine carrée de n
23        | FinSi
```

L'action complexe « Calculer la somme des diviseurs de n » aurait pu être traitée comme une expression complexe « Somme des diviseurs de n ».

On a eu besoin de calculer la somme des diviseurs strict d'un nombre. Naïvement, on peut regarder si les entiers de 2 à $p - 1$ sont des diviseurs de $p$. Une première optimisation consiste à

remarquer que $p$ n'a pas de diviseurs au delà de $p/2$. En fait, il est plus intéressant de remarquer que si $i$ est diviseur de $p$ alors $p/i$ est également un diviseur de $p$. Il suffit donc de ne considérer comme diviseurs potentiels que les entiers compris dans l'intervalle $2..\sqrt{p}$. Il faut toutefois faire attention à ne pas comptabiliser deux fois $\sqrt{p}$.

Continuons avec le deuxième, les nombres amis donc.

```
R0 : Afficher les couples de nombres amis (N, M) avec 1 < N < M <= Max

R1 : Raffinage De « R0 »
       | Pour m <- 2 JusquÀ m = Max Faire
       |    | Pour n <- 2 JusquÀ n = m - 1 Faire
       |    |    | Si n et m amis Alors
       |    |    |    | Afficher le couple (n, m)
       |    |    | FinSi
       |    | FinPour
       | FinPour

R2 : Raffinage De « n et m amis »
       | Résultat <- (somme des diviseurs de N) = M
       |          Et (somme des diviseurs de M) = N
```

On ne détaille pas le raffinage de « somme des diviseurs de p » car nous l'avons déjà traité dans le programme précédent (au nom de l'entier près).

Une solution plus efficace consiste à constater que pour un entier M compris entre 2 et MAX, le seul nombre ami possible est la somme de ses diviseurs que l'on note `somme_m`. Il reste alors à vérifier si la somme des diviseurs de `somme_m` est égale à M pour savoir si `somme_m` et M sont amis. Le fait de devoir afficher les couples dans l'ordre croissant nous conduit à ne considérer que les sommes de diviseurs inférieures à M.

À titre indicatif, pour Max = 1000000, ce deuxième algorithme termine en 1 minute 23 alors que dans le même temps, seuls les sept premiers résultats sont trouvés avec le premier algorithme. Les solutions suivantes sont trouvées au bout d'une minute 32 secondes, 2 minutes 46, 5 minutes 35, 20 minutes, etc.

```
R0 : Afficher les couples de nombres parfaits (N, M) avec 1 < N < M <= Max

R1 : Raffinage De « R0 »
       | Pour m <- 2 JusquÀ m = Max Faire
       |  | Calculer la somme des diviseurs de m
       |  |                                    m: in ; somme_m: out Entier
       |  | n <- somme_m
       |  | Si n < m Alors       { n est un nb amis potentiel }
       |  |  | Calculer la somme des diviseurs de n (somme_n)
       |  |  | Si somme_n = m Alors      { somme_m et m amis }
       |  |  |  | Afficher le couple (n, m)
       |  |  | FinSi
       |  | FinPour
       | FinPour
```

**2.** On ne devrait jamais avoir de code redondant. Comment faire pour éviter d'avoir du code redondant entre les deux programmes précédents.

**Solution :** Avoir du code redondant est mauvais car si on doit intervenir dessus (pour le corriger, l'améliorer ou le changer), il faudra penser à intervenir à l'identique sur toutes les copies. C'est

long, fastidieux et source d'erreur !

Les deux programmes précédents ont besoin de connaître le nombre de diviseurs stricts d'un nombre. Plutôt que d'écrire deux fois le même code dans les deux programmes, il est préférable d'en faire un sous-programme (ici une fonction) pour le factoriser. Comme on veut réutiliser ce sous-programme dans deux programmes différents, on mettra ce sous-programme dans un module.

Avoir des redondances n'est pas bon car si on doit faire évoluer le code dupliqué, il faudra le faire sur toutes ses occurrences. On risque d'en oublier ! Si le code est factorisé (dans un sous-programme par exemple), il suffira de le modifier à un seul endroit. Par exemple, si on est allé jusqu'à $N - 1$ pour calculer la somme des diviseurs stricts de $N$, on n'a pas été efficace. On aurait pu s'arrêté à $\sqrt{N}$ car si $d$ est diviseur de $N$, $N/d$ l'est aussi. Améliorer l'algorithme n'aura à être fait qu'une seule fois si on a écrit le sous-programme correspondant.

**Exercice 4 : Construire un algorithme**

On veut expliquer comment construire un algorithme en utilisant la technique des raffinages. Le R0 est donc "Construire un algorithme" qui, partant d'un problème posé, doit produire le programme correspondant.

Les principales étapes ont été identifiées et sont listées ci-dessous dans l'ordre alphabétique. Il ne reste plus qu'à les structurer en utilisant la technique des raffinages (et donc des structures de contrôles). Pour le premier niveau de raffinage (R1), on fera apparaître le flot de données.

1. Choisir l'étape la moins bien comprise
2. Comprendre le problème
3. Construire le raffinage d'une étape
4. Construire R1
5. Identifier des jeux de tests
6. Identifier des jeux de tests correspondant aux cas hors limites
7. Identifier des jeux de tests correspondant aux cas limites
8. Identifier des jeux de tests représentatifs des cas nominaux
9. Identifier les flots de données
10. Identifier une solution informelle
11. Il y a des étapes non élémentaires
12. Lister les étapes
13. Ordonner les étapes
14. Produire le programme
15. Raffiner les étapes non élémentaires
16. Reformuler le problème
17. Regrouper les étapes
18. Structurer la solution informelle

19. Tester le programme
20. Vérifier l'ensemble de l'algorithme

**Solution :**

```
R0 : Construire un algorithme (pour un problème posé)

R1 : Raffinage De « Construire un algorithme »
     | Comprendre le problème              énoncé: in ; R0, tests: out
     | Identifier une solution informelle  R0, tests: in ; principe: out
     | Structurer cette solution           principe, R0, tests: in ; raffinages: out
     | Produire l'algorithme               raffinages: in ; algorithme: out
     | Tester le programme                 algorithme, tests: in ; erreurs: out

R2 : Raffinage De « Comprendre le problème »
     | Reformuler le problème (R0)              énoncé: in; R0: out
     | Identifier des jeux de tests             énoncé: in; tests: out

R2 : Raffinage De « Structurer cette solution »
     Répéter
     | Construire R1            énoncé, R0, tests: in; R1: out
     | Vérifier R1              R1: in, erreurs: out
     | Raffiner les étapes non élémentaires     raffinages: in out
     | Vérifier l'ensemble de l'algorithme      tests, raffinages: in; erreurs: in out
     JusquÀ pas d'erreurs

R3 : Raffinage De « Identifier des jeux de tests »
     | Identifier des jeux de tests représentatifs des cas nominaux
     | Identifier des jeux de tests correspondant aux cas limites
     | Identifier des jeux de tests correspondant aux cas hors limites

R3 : Raffinage De « Construire R1 »
     | Lister les étapes
     | Ordonner les étapes
     | Regrouper les étapes
     | Identifier les flots de données
     | Compléter le dictionnaire des données

R3 : Raffinage De « Vérifier R1 »
     | Vérifier que les étapes introduites font R0
     | Vérifier que les étapes introduites ne font que R0
     | Vérifier le niveau d'abstraction des étapes
     | Vérifier l'enchaînement des étapes
     | Vérifier la cohérence du flôt de donnée
     | Vérifier la précision du vocabulaire
     | ...

R3 : Raffinage De « Raffiner les étapes non élémentaires »
     | TantQue il y a des étapes non élémentaires Faire
     |   | Choisir l'étape la moins bien comprise
     |   | Construire le raffinage de cette étape
     |   | Vérifier ce raffinage
     | FinTQ
```