Raffinages

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)

R3 : Comment [déterminer] « na et nb différents » ?
    Résultat <- na <> nb

R3 : Comment « Soustraire au plus grand le plus petit » ?
    Si na > nb Alors
        na <- na - nb
    Sinon
        nb <- nb - na
    FinSi
```

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

```
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
    LIMITE_PORTEE : 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");

        -- Demander le choix de l'utilisateur
        Put ("Votre choix : ");
        Get (Choix);
        Skip_Line;

        -- Traiter le choix de l'utilisateur
        case Choix is

            when 'd' | 'D' =>   -- Démarrer
                -- Mettre le drone en route
                En_Route := True;

            when 'm' | 'M' =>   -- Monter
                -- Faire monter le drone
                if En_Route then
                    Altitude := Altitude + 1;
                else
                    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;
```

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.

1. Écrire les raffinages des deux programmes demandés.

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

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