Devoir surveillé

novembre 2014 - Durée 1h15

Documents autorisés : Fiche Traduction Algo-ADA et *Mémento ADA*

Les différentes parties sont indépendantes et peuvent être traitées dans un ordre quelconque.

1 - Assertions - Couverture d'un programme

[*barême indicatif : 7 pts*]

On a le programme suivant :

```
1  with ada.assertions;
2  use  ada.assertions;
3
4  procedure partie1 is
5
6      a,b,c,d,e,res : integer;
7
8  begin
9      get(a); get(b); get(c);
10     d := a-b;
11     if d>0 and c>0 then
12         assert(b/=0);
13         e := c/b;
14         if e>1 then
15             res := e;
16         else
17             res := 0;
18         end if;
19     elsif a>1 then
20         e := c+d;
21         if e>0 then
22             assert(d>0);
23             res := d;
24         else
25             res := a;
26         end if;
27     else
28         res := b;
29     end if;
30     put(res); new_line;
31 end partie1;
```

Question 1-1 :

La compilation du programme donne les erreurs suivantes :

```
partie1.adb:9:09: "get" is undefined (more references follow)
partie1.adb:30:09: "put" is undefined
partie1.adb:30:19: "new_line" is undefined
```

Que faut-il ajouter au programme pour corriger ces erreurs ?

Chaque test (exécution) du programme consiste à entrer 3 valeurs `a`, `b`, `c`.

Question 1-2 :

Donner un test mettant en défaut l'assertion `b/=0` (ligne 12)

Question 1-3 :

Donner un test mettant en défaut l'assertion `d>0` (ligne 22).

Question 1-4 :

Construire un jeu de tests minimal couvrant l'ensemble des instructions du programme et sans mettre en défaut les différentes assertions.

2 - Entrée-sortie - Exceptions

[*barême indicatif : 6 pts*]

On dispose de fichiers contenant des informations sur différents articles de bureau.

Chaque fichier a la structure suivante :
– première ligne : le nombre d'articles présents dans le fichier (nombre entier inférieur ou égal à 10000),
– suivie des différents articles, chacun d'eux étant décrit par quatre champs dans l'ordre suivant :

1. le numéro de référence (entier strictement positif),
2. le prix unitaire (réel positif entre 0,01 et 100000),
3. le stock disponible (entier positif ou nul),
4. le nom (chaine de caractères avec au plus 100 caractères), le nom correspond à une ligne entière du fichier.

Par exemple l'article `"Lampe de bureau - 60 W max."` a pour numéro de référence `201055`, pour prix unitaire `45.00` et un stock disponible de `30` unités.

Par exemple le fichier suivant contient 4 articles :

```
4

104891    3.50  1000
Ramette 500 feuilles A4 / 80 gr

104576    2.75   500
Boite de 200 trombones

105001    0.80   200
Stylo noir pour tableau blanc

201055   45.00    30
Lampe de bureau - 60 W max.
```

On définit d'abord le type *enregistrement* `Article` :

```
1 type Article is record
2     nr : positive;          -- numero de reference
3     pu : float;             -- prix unitaire
4     st : natural;           -- stock disponible
5     nom : string(1..100); -- nom
6     l_nom : natural;        -- longueur du nom
7 end record;
```

ensuite le type `TableauArticle` permettant de stocker au plus 10000 articles :

```
1 type VecteurArticle is array(1..10000) of Article;
2 type TableauArticle is record
3     tab : VecteurArticle; -- tableau pour stocker les articles
4     n : natural;          -- nombre d'articles
5 end record;
```

et la procédure `lire_fichier_article` suivante :

```
1  -- lecture d'un tableau d'article dans le fichier nommé nom_f
2  function lire_fichier_article(nom_f : string) return TableauArticle is
3
4      T : TableauArticle;
5      f : file_type;
6      -- PARTIE A COMPLETER EVENTUELLEMENT --
7
8  begin
9      -- PARTIE A COMPLETER --
10     return T;
11 end lire_fichier_article;
```

Question 2-1 :

Dans un premier temps, on suppose que le fichier existe et est sans erreur, c'est à dire vérifie bien les spécifications ci-dessus.
Complétez la fonction `lire_fichier_article`.

Question 2-2 :

Dans un second temps, on souhaite gérer les erreurs possibles lors de la lecture du fichier à l'aide d'exceptions.
Les fonctions et procédures standard d'entrée-sortie comme `open`, `get`, `skip_line`, ... lèvent des exceptions dans le cas de certaines erreurs (par exemple nom de fichier incorrect, nombre à lire formé de caractères non numériques, ...).
Cependant d'autres erreurs possibles doivent être détectées par programme : comment compléter/modifier la fonction `lire_fichier_article` ?.

3 - Abstraction - Généricité

[*barême indicatif : 7 pts*]

On a le programme suivant ***partie3.adb*** suivant :

```
1  with ada.integer_text_io, ada.numerics.discrete_random;
2  use  ada.integer_text_io;
3
4  -- le paquetage de géneration de nombres entre 3 et 20 inclus
5  procedure partie3 is
6
7      package alea_entier is new ada.numerics.discrete_random(integer);
8      G : generator;
9      v : integer;
10
11 begin
12     -- initialisation du générateur aléatoire
13     reset(G);
14
15     -- génération aléatoire et écriture de 10 nombres compris entre 3 et 20
16     for i in 1..10 loop
17         v := random(G) mod (20-3+1) + 3; -- genere un nombre entre 3 et 20
18         put(v);
19     end loop;
20 end partie3;
```

Question 3-1 :

La compilation de ce programme échoue :
- pour quelle raison ?
- que faut-il ajouter au programme précédent pour corriger l'erreur de compilation.

On souhaite écrire un paquetage ***aleatoire_3_20*** permettant d'initialiser un générateur aléatoire et générer des entiers entre 3 et 20 (inclus) afin de réécrire le programme ***partie3.adb*** ainsi :

```
1  with ada.integer_text_io, aleatoire_3_20;
2  use  ada.integer_text_io;
3
4  -- géneration de nombres entre 3 et 20 inclus
5  procedure partie3 is
6
7      v : integer;
8
9  begin
10     -- initialisation du générateur aléatoire
11     aleatoire_3_20.init;
12
13     -- génération aléatoire et écriture de 10 nombres compris entre 3 et 20
14     for i in 1..10 loop
15         v := aleatoire_3_20.valeur; -- genere un nombre entre 3 et 20 (inclus)
16         put(v);
17     end loop;
18 end partie3;
```

Question 3-2 :

Ecrire les fichiers ***aleatoire_3_20.ads*** et ***aleatoire_3_20.adb***.

A partir du paquetage précedent ***aleatoire_3_20***, on souhaite écrire un paquetage générique ***aleatoire_generique*** permettant de générer des nombres compris entre deux entiers `A` et `B` (inclus)

Question 3-3 :

Comment modifier les fichiers ***aleatoire_3_20.ads*** et ***aleatoire_3_20.adb*** de l'exercice **3-2** afin d'avoir un paquetage générique *paramétré* par deux entiers `A` et `B` ?

Corrigé

1 - Assertions - Couverture d'un programme

[*barême indicatif : 7 pts*]

Question 1-1 :

Pour mettre en défaut l'assertion ligne 16, il faut que $n = j2 - i2 \leq 0$ et ($i2 < 0$ ou $k2 = j1 - i1 < 2$) et $i1 \leq j1$ c'est à dire : $i2 < 0$, $j2 <= i2$, $i1 \geq j1 < i1 - 2$ par exemple $\boxed{i1 = 3,\ i2 = -1,\ j1 = 0,\ j2 = -1}$

Question 1-2 :

Pour mettre en défaut l'assertion ligne 22, il faut que $n \geq k2$ et $n = i2 + j2 > 0$ et $k1 =\geq 5$ et $i2 \geq 0$ et $k2 \geq 2$ par exemple : $\boxed{a = 2,\ b = 2,\ c = 2}$

Question 1-3 :

5 tests sont nécessaires et suffisent à couvrir l'ensemble des instructions :

1. $\boxed{a = 3,\ b = 2,\ c = 4}$ ("couvre" la ligne 15)
2. $\boxed{a = 3,\ b = 2,\ c = 2}$ ("couvre" la ligne 17)
3. $\boxed{a = 3,\ b = 0,\ c = 0}$ ("couvre" la ligne 23)
4. $\boxed{a = 3,\ b = 3,\ c = 0}$ ("couvre" la ligne 25)
5. $\boxed{a = 0,\ b = 0,\ c = 0}$ ("couvre" la ligne 28)

2 - Entrée-sortie - Exception

[*barême indicatif : 6 pts*]

Question 2-1 :

```
1   function lire_fichier_article(nom_f : string) return TableauArticle is
2
3       T : TableauArticle;
4       f : file_type;
5
6   begin
7       -- ouverture du fichier
8       open(f, in_file, nom_f);
9
10      -- lecture du nombre d'articles
11      get(f, T.n);
12
13      -- lecture des articles
14      for i in 1..T.n loop
15          get(f, T.tab(i).nr);
16          get(f, T.tab(i).pu);
17          get(f, T.tab(i).st);
18          skip_line(f);
19          get_line(f, T.tab(i).nom, T.tab(i).l_nom);
20      end loop;
21
22      -- fermeture du fichier
23      close(f);
24
25      return T;
26  end lire_fichier_article;
```

Question 2-2 :

Il faut tester la validité des champs `T.n` et `T.tab(i).pu` après lecture :

```
...
        get(f, T.n);
        if T.n>10000 then
            raise NOMBRE_ARTICLE_INCORRECT;
```

```
        end if;
...
            get(f, T.tab(i).pu);
            if T.tab(i).pu<0.01 or T.tab(i).pu>100000.0 then
                raise PRIX_UNITAIRE_INCORRECT;
            end if;
...
```

et avoir déclaré les exceptions au préalable :

```
    NOMBRE_ARTICLE_INCORRECT, PRIX_UNITAIRE_INCORRECT : exception;
```

3 - Abstraction - Généricité

[*barême indicatif : 7 pts*]

Question 3-1 :

La compilation échoue car il faut préciser à quel paquetage appartiennent le type `generator` et les procédures/fonctions `reset` et `random`.

Pour corriger cette erreur :

- soit ajouter après la ligne 7 :

```
    use alea_entier;
```

- soit spécifier le paquetage `alea_entier` quand cela est nécessaire :

```
    G : alea_entier.generator;
...
    alea_entier.reset;
...
    v = alea_entier.random(G) ...
```

Question 3-2 :

Fichier `aleatoire_3_20.ads` :

```
-- le paquetage de géneration de nombres entre 3 et 20 inclus
package aleatoire_3_20 is

    -- initialisation du paquetage
    procedure init;

    -- renvoie un nombre entre 3 et 20 inclus
    function valeur return integer;
end aleatoire_3_20;
```

Fichier `aleatoire_3_20.adb` :

```
with ada.numerics.discrete_random;

-- le paquetage de géneration de nombres entre 3 et 20 inclus
package body aleatoire_3_20 is

    package alea_entier is new ada.numerics.discrete_random(integer);
    use alea_entier;
    G : generator;

    -- initialisation du paquetage
    procedure init is
    begin
        reset(G);
    end init;

    -- renvoie un nombre entre 3 et 20 inclus
    function valeur return integer is
    begin
        return random(G) mod (20-3+1) + 3;
    end valeur;
end aleatoire_3_20;
```

Question 3-3 :

Dans le fichier `aleatoire_3_20.ads`, remplacer

```
package aleatoire_3_20 is
```

par

```
generic
    A, B : integer;
package aleatoire_3_20 is
```

Dans le fichier `aleatoire_3_20.ads`, remplacer

```
        return random(G) mod (20-3+1) + 3;
```

par

```
        if A<B then
            return random(G) mod (B-A+1) + A; -- cas A inférieur à B
        else
            return random(G) mod (A-B+1) + B; -- cas A supérieur à B
        end if;
```