Devoir surveillé

novembre 2015 - Durée 1h15

Documents autorisés : **Fiche Traduction Algo-ADA** et **Mémento ADA** vierges de toute annotation manuscrite

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

1 - Type abstrait

[*barême indicatif : 6 pts*]

Le paquetage **cartes_paq** décrit dans le fichier `cartes_paq.ads` un type `Carte` représentant une carte d'un jeu de 32 cartes. Une carte est représentée par sa `Valeur` et sa `Couleur`.

Un exemple d'utilisation du paquetage **cartes_paq** est donné dans le fichier `prog1.adb`.

```
1  -- definition du type Carte
2  -- fichier interface : cartes_paq.ads
3
4  package cartes_paq is
5
6    -- type enumere representant l'ensemble des valeurs
7    type Valeur is (sept, huit, neuf, dix, valet, dame, roi, as);
8
9    -- type enumere representant l'ensemble des couleurs
10   type Couleur is (pique, coeur, carreau, trefle);
11
12   -- type enregoistrement representant une carte
13   type Carte is record
14     v : Valeur;
15     c : Couleur;
16   end record;
17
18 end cartes_paq;
```

```
1  -- fichier : prog1.adb
2  with ada.text_io;
3  use ada.text_io;
4
5  procedure prog1 is
6
7    c1, c2 : Carte;
8
9  begin
10    -- c1 : huit de trefle
11    c1.v := huit;
12    c1.c := trefle;
13    -- c2 : dame de coeur
14    c2.v := dame;
15    c2.c := coeur;
16
17    -- afficher la valeur de la carte c1
18    put_line ("Valeur de c1 " & Valeur'image(c1.v));
19    -- afficher la couleur de la carte c1
20    put_line ("Couleur de c1 " & Couleur'image(c1.c));
21
22    -- comparer les cartes c1 et c2 suivant leurs valeurs
23    if c1.v < c2.v then put_line ("Valeur de c1 < Valeur de c2");
24    elsif c1.v > c2.v then put_line ("Valeur de c1 > Valeur de c2");
25    else put_line ("Valeur de c1 et de c2 identiques");
26    end if;
27
28 end prog1;
```

La compilation du programme donne les erreurs suivantes :

```
prog1.adb:6:12: "Carte" is undefined
prog1.adb:10:04: invalid prefix in selected component "c1"
prog1.adb:10:12: "huit" is undefined
prog1.adb:11:04: invalid prefix in selected component "c1"
prog1.adb:11:12: "trefle" is undefined
prog1.adb:13:04: invalid prefix in selected component "c2"
prog1.adb:13:12: "dame" is undefined
prog1.adb:14:04: invalid prefix in selected component "c2"
prog1.adb:14:12: "coeur" is undefined
prog1.adb:17:32: "Valeur" is undefined
prog1.adb:19:33: "Couleur" is undefined
prog1.adb:22:07: invalid prefix in selected component "c1"
prog1.adb:22:14: invalid prefix in selected component "c2"
prog1.adb:23:10: invalid prefix in selected component "c1"
prog1.adb:23:17: invalid prefix in selected component "c2"
```

Question 1-1 :

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

Afin de rendre le type `Carte` privé, on souhaite modifier le paquetage **cartes_paq** ainsi :

```
1  -- definition du type Carte
2  -- fichier interface : cartes_paq.ads
3
4  package cartes_paq is
5
6    -- type enumere representant l'ensemble des valeurs
7    type Valeur is (sept, huit, neuf, dix, valet, dame, roi, as);
8
9    -- type enumere representant l'ensemble des couleurs
10   type Couleur is (pique, coeur, carreau, trefle);
11
12   type Carte is private;
13
14 private
15   -- type enregoistrement representant une carte
16   type Carte is record
17     v : Valeur;
18     c : Couleur;
19   end record;
20
21 end cartes_paq;
```

Question 1-2 :

En utilisant le concept de **type abstrait** vu en cours, complétez le paquetage **cartes_paq** (interface et corps) avec les fonctions et procédures nécessaires et modifier le programme prog1 en conséquence pour garder le même comportement lors de l'éxécution (ne récrivez que ce qui est nécessaire en vous aidant des numéros de ligne).

2 - Entrées-Sorties

[*barême indicatif : 8 pts*]

Indication : Dans cette question vous devez programmer des entrées-sorties pour les types énumérés `Valeur` et `Couleur`. Avant de traiter la question, lisez l'annexe concernant les entrées-sorties pour les types énumérés. Vous y trouverez de quelle façon instancier un sous-paquetage générique afin de pouvoir lire et écrire une valeur d'un type énuméré.

Le paquetage **donne_paq** définit un type `Donne` pour représenter la répartition des cartes entre 4 joueurs.

La procédure `lire_donne` lit une donne depuis un fichier. Ce fichier comporte 32 lignes qui représentent successivement les 8 cartes du joueur 1, puis les 8 cartes du joueur 2, puis les 8 cartes du joueur 3 et enfin les 8 cartes du joueur 4. Chaque carte est écrite sur une ligne sous la forme de deux chaines de caractères, la première indiquant la valeur de la carte et la seconde indiquant sa couleur. Par exemple une ligne peut être : DIX PIQUE.

La procédure `ecrire_donne` écrit une donne dans un fichier, au format décrit précédemment.

```
1  -- paquetage definissant le type Donne
2  -- fichier donne_paq.ads
3
4  with cartes_paq;
5  use cartes_paq;
6
7  package donne_paq is
8     subtype Joueur is integer range 1..4;
9     subtype Numcarte is integer range 1..8;
10    type Donne is array (Joueur, NumCarte) of Carte;
11
12    procedure lire_donne (nom_f: in string; d: out Donne);
13    procedure ecrire_donne (nom_f: in string; d: in Donne);
14
15 end donne_paq;
```

Question 2-1 :

Ecrivez le contenu du fichier **donne_paq.adb** (corps du paquetage).

Question 2-2 :

Ecrivez un programme **prog2.adb** qui attend deux arguments, (deux noms de fichier) et qui lit une donne depuis le premier fichier et écrit cette donne dans le second fichier. Vous utiliserez les procédures `lire_donne` et `ecrire_donne`. S'il n'y a pas deux arguments votre programme doit afficher un message d'erreur.

3 - Assertions, couverture d'un programme

[*barême indicatif : 6 pts*]

Considérons le programme ci-dessous pour lequel chaque exécution consiste à entrer 4 valeurs pour les variables `a`, `b`, `c` et `d`.

```
1  with ada.integer_text_io, ada.text_io, ada.assertions;
2  use ada.integer_text_io, ada.text_io, ada.assertions;
3
4  procedure prog3 is
5
6     a,b,c,d,e,f,g : integer;
7
8  begin
9
10    get(a); get(b); get(c); get(d);
11    e := a*b;
12    if d<e then
13       f := c-a;
14       if f>0 then
15          d := d*2;
16          g := e-b;
17          if g>a then
18             b := b+a;
19          else
20             assert (d>=f, ..... );
21             d := d-5;
22          end if;
23          g := a/2;
24       else
25          d := d-1;
26          g := a+1;
27       end if;
28       b := b+g;
29       assert (e>f, ..... );
30    end if;
31
32 end prog3;
```

Question 3-1 :

Indiquez de quelle manière compléter les assertions (lignes 20 et 29).

Question 3-2 :

1. Donnez les propriétés que doit avoir un test qui met en défaut l'assertion `d>=f`. Donnez un exemple d'un tel test.
2. Donnez les propriétés que doit avoir un test qui met en défaut l'assertion `e>f`. Donnez un exemple d'un tel test.
3. Expliquer comment construire un jeu de tests minimal permettant de couvrir l'ensemble des instructions du programme sans mettre en défaut aucune assertion. Donnez les propriétés que doit avoir un tel jeu de tests. On ne vous demande pas d'exemple.

Annexe : entrées-sorties pour un type énuméré

Le programme **exemple_es_type_enumere** ci-dessous montre comment utiliser le sous-paquetage générique **ada.text_io.enumeration_io** afin d'effectuer des entrées-sortie pour un type énuméré. Comme le sous-paquetage générique **ada.text_io.enumeration_io** est déclaré dans le paquetage **ada.text_io**, il suffit d'inclure le seul paquetage **ada.text_io** avec la clause `with`.

```
1  -- entree/sortie pour type enumeré
2  with ada.text_io;
3
4  procedure exemple_es_type_enumere is
5
6     -- declaration d'un type enumere
7      type Piece is (Pile,Face);
8
9     -- instanciation des entrees-sorties pour le type Piece
10     package ES_Piece is new ada.text_io.enumeration_io(Piece);
11
12     p : Piece;
13
14 begin
15
16    while true loop
17       ada.text_io.put("Entrer pile ou face : ");
18       ES_Piece.get(p);
19       ada.text_io.put("La valeur entree est :");
20       ES_Piece.put(p);
21       ada.text_io.new_line;
22       ada.text_io.new_line;
23    end loop;
24
25 end exemple_es_type_enumere;
```

Et voila un exemple d'exécution :

```
Entrer pile ou face : PILE
La valeur entree est :PILE

Entrer pile ou face : pile
La valeur entree est :PILE

Entrer pile ou face : FACE
La valeur entree est :FACE

Entrer pile ou face : face
La valeur entree est :FACE

Entrer pile ou face : fAcE
La valeur entree est :FACE

Entrer pile ou face : FACCE

raised ADA.IO_EXCEPTIONS.DATA_ERROR : a-tienio.adb:57 instantiated at
exemple_es_type_enumere.adb:11
```