Devoir surveillé

24 octobre 2015 - Durée 1h30

Document autorisé : **Mémento Ada** vierge de toute annotation manuscrite

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

Éléments de correction

1. Types abstraits (6 pt)

On considère le paquetage `Type_Complexe` dont la spécification est la suivante :

```
1  package Type_Complexe is
2
3     type Complexe is record
4        R : Float;
5        I : Float;
6     end record;
7
8  end Type_Complexe;
```

Ce paquetage est utilisé par le programme `Test_Complexe` :

```
1  with Ada.Text_IO, Ada.Float_Text_IO, Type_Complexe;
2  use Ada.Text_IO, Ada.Float_Text_IO, Type_Complexe;
3
4  procedure Test_Complexe is
5     C1, C2, C : Complexe;
6  begin
7     Put_Line("Saisie de C1 :");
8     Put("Saisie de la partie réelle : ");
9     Get(C1.R);
10    put("Saisie de la partie imaginaire : ");
11    Get(C1.I);
12    -- C2 = 3 + 5i
13    C2.R := 3;
14    C2.I := 5;
15    C.R := C1.R + C2.R;
16    C.I := C1.I + C2.I;
17    Put("C1 + C2 = ");
18    Put("("); Put(C.R); Put(", "); Put(C.I); Put(")");
19    New_Line;
20 end Test_Complexe;
```

Exercice 1. (3 pt) Modifier le paquetage `Type_Complexe` pour rendre le type `Complexe` abstrait. Quels sont les opérations à ajouter à la spécification ? Donner l'implémentation de ces opérations.

```
1  package Type_Complexe is
2     type Complexe is private;
3
4     procedure Get(C : out Complexe);
5
6     function Creer_Complexe(R : Float; I : Float) return Complexe;
7
8     function Somme(C1, C2 : Complexe) return Complexe;
9
10    procedure Put(C : in Complexe);
11
12 private
13
14    type Complexe is record
15       R : Float;
16       I : Float;
17    end record;
18
19 end Type_Complexe;
```

```
1  with Ada.Text_IO, Ada.Float_Text_IO;
2  use Ada.Text_IO, Ada.Float_Text_IO;
3
4  package body Type_Complexe is
5
6     procedure Get(C : out Complexe) is
7     begin
8        Put("Partie réelle :");
9        Get(C.R);
10       Put("Partie imaginaire :");
11       Get(C.I);
12    end Get;
13
14    function Creer_Complexe(R : Float; I : Float) return Complexe is
15    begin
16       return (R, I);
17    end Creer_Complexe;
18
19    function Somme(C1, C2 : Complexe) return Complexe is
20       C : Complexe;
21    begin
22       C.R := C1.R + C2.R;
23       C.I := C1.I + C2.I;
24       return C;
25    end Somme;
26
27    procedure Put(C : in Complexe) is
28    begin
29       Put("("); Put(C.R); Put(", "); Put(C.I); Put(")");
30    end Put;
31
32 end Type_Complexe;
```

Exercice 2. (1 pt) Une fois le paquetage ainsi modifié, le programme `Test_Complexe` est-il correct ? À quelle ligne la compilation devrait-elle échouer et pourquoi ?

*Le programme n'est pas correct. La compilation va échouer à la ligne 11, du fait que l'accès au champ `C1.R` est interdit par le mot-clé **`private`**.*

Exercice 3. (2 pt) Modifier le programme `Test_Complexe` pour prendre en compte les modifications apportées au paquetage `Type_Complexe` à l'exercice 1.

```
1  with Ada.Text_IO, Type_Complexe;
2  use Ada.Text_IO, Type_Complexe;
3
4  procedure Test_Complexe is
5
6     C1, C2, C : Complexe;
7
8  begin
9     Put_Line("Saisie de C1 :");
10    Get(C1);
11    C2 := Creer_Complexe(3.0,5.0);
12    C := Somme(C1, C2);
13    Put("C1 + C2 = ");
14    Put(C);
15    New_Line;
16 end Test_Complexe;
```

2. Tests et généricité (14 pt)

On considère le paquetage `Tri_Comptage`, implémentant l'algorithme de «tri comptage», dont la spécification et l'implémentation sont fournies en annexe.

Ce paquetage est utilisé par le programme suivant :

```
1  with Ada.Integer_Text_IO, Tri_Comptage;
2  use Ada.Integer_Text_IO, Tri_Comptage;
3
4  procedure Test_Tri_Comptage is
5     -- Taille du tableau
6     N : Natural;
7  begin
8     Get(N);
9     declare
10       -- Tableau à trier
11       A : TabIntervalle(1..N);
12    begin
13       -- Lecture des valeurs du tableau
14       for I in A'Range loop
15          Get(A(I));
16       end loop;
17       -- Tri du tableau
18       Trier(A);
19       -- Affichage du tableau
20       for I in A'Range loop
21          Put(A(I));
22       end loop;
23    end;
24 end Test_Tri_Comptage;
```

Exercice 4. (2 pt) Quel est le format des entrées de ce programme ? Décrire précisément le domaine de validité des entrées.

Les entrées sont constituées :

1. *d'un entier* $N \geq 0$*;*
2. *d'une séquence de N entiers compris dans l'intervalle* $[0, 1000]$.

Exercice 5. (2 pt) Donner une trace d'exécution de la procédure `Trier` (valeurs successives de `I` et des tableaux `T` et `Compte`) pour `T` $= [1, 4, 6, 3, 2]$.

Valeur initiale :

	1	2	3	4	5
`T` =	1	4	6	3	2

	0	1	2	3	4	5	6	7	⋯	1000
`Compte` =	0	0	0	0	0	0	0	0	⋯	0

Première boucle :

— $I = 1$

	1	2	3	4	5
`T` =	1	4	6	3	2

	0	1	2	3	4	5	6	7	⋯	1000
`Compte` =	0	1	0	0	0	0	0	0	⋯	0

— $I = 2$

	1	2	3	4	5
`T` =	1	4	6	3	2

	0	1	2	3	4	5	6	7	⋯	1000
`Compte` =	0	1	0	0	1	0	0	0	⋯	0

— $I = 3$

	1	2	3	4	5
`T` =	1	4	6	3	2

	0	1	2	3	4	5	6	7	⋯	1000
`Compte` =	0	1	0	0	1	0	1	0	⋯	0

— $I = 4$

	1	2	3	4	5
`T` =	1	4	6	3	2

	0	1	2	3	4	5	6	7	⋯	1000
`Compte` =	0	1	0	1	1	0	1	0	⋯	0

— $I = 5$

	1	2	3	4	5
`T` =	1	4	6	3	2

	0	1	2	3	4	5	6	7	⋯	1000
`Compte` =	0	1	1	1	1	0	1	0	⋯	0

Deuxième boucle :

— $I = 1, J = 0$

	1	2	3	4	5
`T` =	1	4	6	3	2

	0	1	2	3	4	5	6	7	⋯	1000
`Compte` =	0	1	1	1	1	0	1	0	⋯	0

— $I = 1, J = 1$

	1	2	3	4	5
`T` =	1	4	6	3	2

	0	1	2	3	4	5	6	7	⋯	1000
`Compte` =	0	1	1	1	1	0	1	0	⋯	0

— $I = 1, J = 2$

	1	2	3	4	5
`T` =	1	2	6	3	2

	0	1	2	3	4	5	6	7	⋯	1000
`Compte` =	0	1	1	1	1	0	1	0	⋯	0

— $I = 1, J = 3$

	1	2	3	4	5
`T` =	1	2	3	3	2

	0	1	2	3	4	5	6	7	⋯	1000
`Compte` =	0	1	1	1	1	0	1	0	⋯	0

— $I = 1, J = 4$

	1	2	3	4	5
`T` =	1	2	3	4	2

	0	1	2	3	4	5	6	7	⋯	1000
`Compte` =	0	1	1	1	1	0	1	0	⋯	0

— $I = 1, J = 5$

	1	2	3	4	5
`T` =	1	2	3	4	2

	0	1	2	3	4	5	6	7	⋯	1000
`Compte` =	0	1	1	1	1	0	1	0	⋯	0

— $I = 1, J = 6$

	1	2	3	4	5
`T` =	1	2	3	4	6

	0	1	2	3	4	5	6	7	⋯	1000
`Compte` =	0	1	1	1	1	0	1	0	⋯	0

⋮

— $I = 1, J = 1000$

	1	2	3	4	5
`T` =	1	2	3	4	6

	0	1	2	3	4	5	6	7	⋯	1000
`Compte` =	0	1	1	1	1	0	1	0	⋯	0

Exercice 6. (2 pt) Décrire un jeu de tests pour le programme `Test_Tri_Comptage`.

Cf CTD n° 2 et 3...

On souhaite maintenant rendre ce paquetage générique : le **type des objets triés** doit être un paramètre générique `Objet` de ce paquetage.

Exercice 7. (1 pt) Quels sont les opérations de la procédure `Trier` nécessitant une modification suite au passage en paramètre générique du type des objets triés ?

Les deux opérations nécessitant une modification sont :
- *ligne 15 : l'accès au tableau `Compte` par l'indice `T(I)` (le type de `T(I)` étant un paramètre générique il ne peut être utilisé comme indice) ;*
- *ligne 22 : l'affectation de l'indice `J` à la case `T(I)`.*

Exercice 8. (3 pt) Modifier (en indiquant très précisément quels fichiers et quelles lignes sont modifiées) le paquetage `Tri_Comptage` afin que le **type des objets triés** soit un paramètre générique `Objet` de ce paquetage.

Indications :
- cet algorithme ne permet de trier que des ensembles objets E tels qu'il existe une *bijection* de cet ensemble dans un intervalle entier ;
- un paramètre générique `I` de type intervalle entier peut être déclaré avec la syntaxe **`type I is range`** <>.

```
1  generic
2     type Objet is private;
3     type Intervalle is range <>;
4     type TabIntervalle is array(Integer range <>) of Objet;
5     with function Indice(X : Objet) return Intervalle;
6     with function Valeur(I : Intervalle) return Objet;
7
8  package Tri_Comptage is
9
10    procedure Trier(T : in out TabIntervalle);
11
12 end Tri_Comptage;
```

```
1  package body Tri_Comptage is
2
3     procedure Trier(T : in out TabIntervalle) is
4
5        -- Compte(I) contient le nombre de valeurs égales à I dans T
6        Compte : array(Intervalle) of Natural;
7        I : Integer;
8
9     begin
10       -- Initialisation de Compte à 0
11       Compte := (others => 0);
12
13       for I in T'Range loop
14          -- Incrémenter le nombre de valeurs T(I) lues
15          Compte(Indice(T(I))) := Compte(Indice(T(I))) + 1;
16       end loop;
17
18       -- Mettre les valeurs de l'intervalle en ordre croissant dans T
19       I := T'First;
20       for J in Intervalle'Range loop
21          for K in 1..Compte(J) loop
22             T(I) := Valeur(J);
23             I := I + 1;
24          end loop;
25       end loop;
26    end Trier;
27
28 end Tri_Comptage;
```

Exercice 9. (2 pt) Indiquer les modifications à apporter au programme `Test_Tri_Comptage` pour utiliser le paquetage `Tri_Comptage` modifié à l'exercice 8.

```
1  with Ada.Integer_Text_IO, Tri_Comptage;
2  use Ada.Integer_Text_IO;
3
4  procedure Test_Tri_Comptage is
5
6     subtype IntervalleEntier is Integer range 1..100;
7
8     type TabEntier is array(Integer range <>) of IntervalleEntier;
9
10    function Id(X:IntervalleEntier) return IntervalleEntier is
11    begin
12       return X;
13    end Id;
14
15    package TriCEntiers is new Tri_Comptage(IntervalleEntier,
16                                            IntervalleEntier,
17                                            TabEntier, Id, Id);
18
19
20    -- Taille du tableau
21    N : Natural;
22 begin
23    Get(N);
24    declare
25       -- Tableau à trier
26       A : TabEntier(1..N);
27    begin
28       -- Lecture des valeurs du tableau
29       for I in A'Range loop
30          Get(A(I));
31       end loop;
32       -- Tri du tableau
33       TriCEntiers.Trier(A);
34       -- Affichage du tableau
35       for I in A'Range loop
36          Put(A(I));
37       end loop;
38    end;
39 end Test_Tri_Comptage;
```

Exercice 10. (2 pt) Écrire un programme, utilisant le paquetage générique `Tri_Comptage`, permettant de trier un tableau de caractères.

Indications :

- le code ASCII sous forme entière d'un caractère `C` peut être obtenu par `Character'Pos(C)`
- le caractère de code ASCII `V` peut être obtenu par `Character'Val(V)`.

```
1  with Ada.Text_IO, Ada.Integer_Text_IO, Tri_Comptage;
2  use Ada.Text_IO, Ada.Integer_Text_IO;
3
4  procedure Tri_Caracteres is
5
6     subtype IntervalleASCII is Integer range 0..255;
7
8     type TabCar is array(Integer range <>) of Character;
9
10    function Pos(C:Character) return IntervalleASCII is
11    begin
12       return Character'Pos(C);
13    end Pos;
14
15    function Val(V:IntervalleASCII) return Character is
16    begin
17       return Character'Val(V);
18    end Val;
19
20    package TriCaracteres is new Tri_Comptage(Character,
21                                              IntervalleASCII,
22                                              TabCar, Pos, Val);
23
24
25    -- Taille du tableau
26    N : Natural;
27
28 begin
29    Get(N);
30    declare
31       -- Tableau à trier
32       A : TabCar(1..N);
33    begin
34       -- Lecture des valeurs du tableau
35       for I in A'Range loop
36          Get(A(I));
37       end loop;
38       -- Tri du tableau
39       TriCaracteres.Trier(A);
40       -- Affichage du tableau
41       for I in A'Range loop
42          Put(A(I));
43       end loop;
44    end;
45 end Tri_Caracteres;
```

A. Paquetage Tri_Comptage

A.1 Spécification

```
1  package Tri_Comptage is
2
3     subtype Intervalle is Integer range 0..1000;
4
5     type TabIntervalle is array(Integer range <>) of Intervalle;
6
7     procedure Trier(T : in out TabIntervalle);
8
9  end Tri_Comptage;
```

A.2 Implémentation

```
1  package body Tri_Comptage is
2
3     procedure Trier(T : in out TabIntervalle) is
4
5        -- Compte(I) contient le nombre de valeurs égales à I dans T
6        Compte : array(Intervalle) of Natural;
7        I : Integer range Intervalle'First..Intervalle'Last+1;
8
9     begin
10       -- Initialisation de Compte à 0
11       Compte := (others => 0);
12
13       for I in T'Range loop
14          -- Incrémenter le nombre de valeurs T(I) lues
15          Compte(T(I)) := Compte(T(I)) + 1;
16       end loop;
17
18       -- Mettre les valeurs de l'intervalle en ordre croissant dans T
19       I := T'First;
20       for J in Intervalle'Range loop
21          for K in 1..Compte(J) loop
22             T(I) := J;
23             I := I + 1;
24          end loop;
25       end loop;
26    end Trier;
27
28 end Tri_Comptage;
```