Examen

2 janvier 2017 — Durée 2h

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. Tests (11 pt)

On considère le programme Ada suivant :

```
1  with Ada.Text_IO, Ada.Integer_Text_IO;
2  use Ada.Text_IO, Ada.Integer_Text_IO;
3
4  procedure Max_Intervalle is
5
6     BInf, BSup : Positive;
7     EC : Natural;
8     Existe_Max : Boolean;
9     Max : Positive;
10
11 begin
12
13    Put("Borne inférieure de l'intervalle : ");
14    Get(Binf);
15    Put("Borne supérieure de l'intervalle : ");
16    Get(Bsup);
17    if not (BInf <= BSup) then
18       raise Constraint_Error;
19    end if;
20
21    Existe_Max := False;
22    Get(EC);
23    while EC /= 0 loop
24       if (EC >= BInf) and (EC <= Bsup) then
25          if Existe_Max then
26             if EC > Max then
27                Max := EC;
28             end if;
29          else
30             Max := EC;
31             Existe_Max := True;
32          end if;
33       end if;
34
35       Get(EC);
36    end loop;
37
38    if Existe_Max then
39       Put("Valeur maximum : "); Put(Max); New_Line;
40    else
41       Put_line("Pas de valeur maximum");
42    end if;
43
44 end Max_Intervalle;
```

Exercice 1. (1 pt) Expliquer en quelques phrases ce que fait ce programme.

Ce programme lit un intervalle sur l'entrée standard, puis une séquence d'entiers positifs, terminée par la valeur 0*. Il affiche ensuite, si elle existe, la valeur maximum des valeurs de la séquence appartenant à l'intervalle donné. Si cette valeur n'existe pas le texte «Pas de valeur maximum» est affiché.*

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

Une entrée de ce programme consiste en un intervalle $[I, S]$*, et une séquence d'entiers strictement positifs. Le format de cette entrée est :*

1. *un entier* $I > 0$
2. *un entier* $S > 0$
3. *une séquence d'entiers strictement positifs*
4. *l'entier* 0*.*

Exercice 3. (3 pt) Décrire un jeu de tests pour ce programme.

Chaque test est un couple (intervalle, séquence d'entiers).

- *Tests affichant «Pas de valeur maximum» :*
  - *Séquence vide : par exemple, le test* $([1,3],[])$
  - *Séquence non vide : par exemple, le test* $([2,3],[1,4,6])$
- *Tests affichant «Valeur maximum ...» :*
  - *Cas limite : intervalle comportant un seul élément*
    - *Cas limite : séquence d'un élément :* $([2,2],[2])$
    - *Séquences de plusieurs éléments :*
      - *Cas limite : tous les éléments de la séquence appartiennent à l'intervalle :* $([2,2],[2,2])$
      - *Cas général :* $([2,2],[1,2,3])$
  - *Cas général : intervalle comportant plusieurs éléments*
    - *Cas limite : séquence d'un élément :*
      - *Cas limite : l'élément de la séquence est la borne inférieure de l'intervalle :* $([1,3],[1])$
      - *Cas limite : l'élément de la séquence est la borne supérieure de l'intervalle :* $([1,3],[3])$
      - *Cas général :* $([1,3],[2])$
    - *Séquences de plusieurs éléments :*
      - *Cas limite : l'élément maximum de la séquence est la borne inférieure de l'intervalle :* $([1,3],[1,4])$
      - *Cas limite : l'élément maximum de la séquence est la borne supérieure de l'intervalle :* $([1,3],[3,4])$
      - *Cas général :* $([1,3],[2,4])$

Exercice 4. (2 pt) Modifier le programme afin qu'une exception spécifique (une exception par cas d'erreur) soit levée lorsqu'une entrée non valide est fournie.

```
1  with Ada.Text_IO, Ada.Integer_Text_IO;
2  use Ada.Text_IO, Ada.Integer_Text_IO;
3
4  procedure Max_Intervalle is
5
6     BInf, BSup : Positive;
7     EC : Natural;
8     Existe_Max : Boolean;
9     Max : Positive;
10
11    BInf_Error,
12    BSup_Error,
13    BInf_Sup_BSup,
14    Elem_Error : exception;
15
16 begin
17
18    Put("Borne inférieure de l'intervalle : ");
19    begin
20       Get(Binf);
21    exception
22       when Constraint_Error => raise BInf_Error;
23    end;
24    Put("Borne supérieure de l'intervalle : ");
25    begin
26       Get(Bsup);
27    exception
28       when Constraint_Error => raise BSup_Error;
29    end;
30    if not (BInf <= BSup) then
31       raise BInf_Sup_BSup;
32    end if;
33
34    Existe_Max := False;
35    begin
36       Get(EC);
37       while EC /= 0 loop
38          if (EC >= BInf) and (EC <= Bsup) then
39             if Existe_Max then
40                if EC > Max then
41                   Max := EC;
42                end if;
43             else
44                Max := EC;
45                Existe_Max := True;
46             end if;
47          end if;
48
49          Get(EC);
50       end loop;
51    exception
52       when Constraint_Error => raise Elem_Error;
53    end;
54
55    if Existe_Max then
56       Put("Valeur maximum : "); Put(Max); New_Line;
57    else
58       Put_line("Pas de valeur maximum");
59    end if;
60
61 end Max_Intervalle;
```

Exercice 5. (3 pt) Écrire un programme permettant de générer de manière aléatoire N fichiers tests pour le programme `Max_Intervalle`. Le nombre N de fichiers générés est fourni en argument de la ligne de commande.

```
1  with Ada.Text_IO, Ada.Integer_Text_IO, Ada.Command_Line;
2  use Ada.Text_IO, Ada.Integer_Text_IO, Ada.Command_Line;
3  with Ada.Numerics.Discrete_Random;
4
5  procedure Gen_Test_Max_Intervalle is
6
7     package Alea is new Ada.Numerics.Discrete_Random(Positive);
8
9
10    G : Alea.Generator;
11    F : File_Type;
12    BInf, BSup : Positive;
13    Tmp : Positive;
14    Nb_Elem_Max : constant Natural := 10000;
15    Nb_Elem : Natural;
16    EC : Positive;
17
18 begin
19    Alea.Reset(G);
20    for I in 1..Integer'Value(Argument(1)) loop
21       Create(F, Out_File, "Test_" & Integer'Image(I));
22       -- Génération de l'intervalle
23       BInf := Alea.Random(G);
24       Bsup := Alea.Random(G);
25       if BInf > BSup then
26          -- Échange de BInf et BSup
27          Tmp := BInf;
28          BInf := BSup;
29          BSup := Tmp;
30       end if;
31       Put(F, BInf);New_Line(F);
32       Put(F, BSup);New_Line(F);
33       -- Séquence d'éléments
34       -- Nombre d'éléments de la séquence
35       Nb_Elem := Alea.Random(G) mod Nb_Elem_Max;
36       -- Éléments de la séquence
37       for I in 1..Nb_Elem loop
38          EC := Alea.Random(G);
39          Put(F, EC); New_Line(F);
40       end loop;
41       -- Fin de la séquence
42       Put(F, 0); New_Line(F);
43       Close(F);
44    end loop;
45 end Gen_Test_Max_Intervalle;
```

NB :

- L'expression `Integer'Value(S)` donne la valeur d'un entier représenté par la chaîne de caractères `S` (`S` de type `String`). L'expression `Integer'Image(X)` donne la représentation sous forme de chaîne de caractères de l'entier `X`.
- La concaténation de deux chaînes `S1` et `S2` s'obtient avec `S1 & S2`.
- La spécification du paquetage `Ada.Numerics.Discrete_Random` est fournie en annexe.

2. Paquetages et généricité (9 pt)

On considère le paquetage générique `Ensemble_Paq` dont la spécification est donnée ci-dessous (implémentation fournie en annexe). Ce paquetage fournit un type `Ensemble`, implémentant un ensemble d'objets au sens mathématique : il ne peut y avoir plusieurs occurrences d'un même objet dans un ensemble.

```
1  generic
2
3     -- Type des objets de l'ensemble
4     type Objet is private;
5     -- Comparaison entre deux objets
6     with function Egal(O1,O2:Objet) return Boolean;
7
8  package Ensemble_Paq is
9
10    -- type Ensemble d'éléments de type Objet
11    type Ensemble is private;
12
13    -- Retourne un ensemble vide
14    function Ensemble_Vide return Ensemble;
15
16    -- Ajouter l'élément X dans l'ensemble E.
17    -- Si X est présent dans E, E n'est pas modifié.
18    procedure Ajouter(E : in out Ensemble; X : in Objet);
19
20    -- Supprimer l'élément X de l'ensemble E.
21    procedure Supprimer(E : in out Ensemble; X : in Objet);
22
23    -- Tester si l'ensemble E est vide
24    function Est_Vide(E : Ensemble) return Boolean;
25
26    -- Tester si l'élément X appartient à E
27    function Appartient(X : Objet; E : Ensemble) return Boolean;
28
29    -- Renvoie le nombre d'éléments de l'ensemble E
30    function Nb_Elements(E : Ensemble) return Natural;
31
32 private
33
34    LMAX : constant Natural := 1000;
35    type TObjet is array (Natural range 1..LMAX) of Objet;
36    type Ensemble is record
37       T : TObjet;
38       N : Natural range 0..LMAX;
39    end record;
40
41 end Ensemble_Paq;
```

Exercice 6. (2 pt) Soit le programme Ada suivant, instanciant et utilisant le paquetage `Ensemble_Paq` pour créer l'ensemble $\{42\}$:

```
1  with Ensemble_Paq;
2
3  procedure Test_Ensemble_Paq is
4     [...]
5     E : Ensemble;
6  begin
7     E.N := 1;
8     E.T(1) := 42;
9  end Test_Ensemble_Paq;
```

On suppose l'instanciation du paquetage `Ensemble_Paq` (non montrée ci-dessus) correcte.

Ce programme est-il correct ? Justifier votre réponse et indiquez, s'il n'est pas correct :

1. si l'erreur apparaîtra à la compilation ou à l'exécution ;

2. à quelle ligne se trouve l'erreur en question ?

Le programme n'est pas correct : il tente d'accéder au champ `N` de la variable `E` de type `Ensemble`, alors que ce type est privé.
L'erreur apparaîtra à la ligne 7 à la compilation.

Exercice 7. (2 pt) Écrire l'instanciation du paquetage `Ensemble_Paq` pour que le type `Ensemble` corresponde à un *ensemble d'entiers*.

```
1  -- Fonction de comparaison de deux entiers
2  function EgalEntiers(X,Y : Integer) return Boolean is
3  begin
4     return X = Y;
5  end EgalEntiers;
6
7  -- Instanciation du paquetage Ensemble_Paq
8  package Ensemble_Entiers is new Ensemble_Paq(Integer, EgalEntiers);
```

Exercice 8. (3 pt) Écrire une fonction Ada, utilisant le paquetage instancié à l'exercice 7, permettant de lire un ensemble d'entiers dans un fichier. Le nom du fichier est fourni en paramètre de la fonction. Le fichier lu contient sur la première ligne le nombre d'éléments à lire, suivi d'une séquence d'entiers.

```
1  function Lire_Ensemble(Filename : String)
2                    return Ensemble_Entiers.Ensemble is
3     F : File_Type;
4     Nb_Elem : Natural;
5     EC : Integer;
6     E : Ensemble_Entiers.Ensemble;
7  begin
8     -- Initialisation de l'ensemble
9     E := Ensemble_Entiers.Ensemble_Vide;
10    -- Ouverture du fichier en lecture
11    Open(F, In_File, Filename);
12    -- Lecture du nombre d'éléments de la séquence
13    Get(F, Nb_Elem);
14    -- Lecture des éléments de la séquence
15    for I in 1..Nb_Elem loop
16       Get(F, EC);
17       Ensemble_Entiers.Ajouter(E, EC);
18    end loop;
19    Close(F);
20    return E;
21 end Lire_Ensemble;
```

Exercice 9. (2 pt) Utiliser les fonctions et instanciations des exercices précédents pour écrire un programme Ada qui lit une séquence d'entiers dans un fichier dont le nom est fourni en argument de la ligne de commande, et affiche le nombre d'entiers distincts de cette séquence.

```
1     E : Ensemble_Entiers.Ensemble;
2
3  begin
4     E := Lire_Ensemble(Argument(1));
5     Put(Ensemble_Entiers.Nb_Elements(E)); New_Line;
6  end Nb_Entiers_Distincts;
```

Annexes

A. Spécification du paquetage `Ada.Numerics.Discrete_Random`

```
1  generic
2          type Result_Subtype is (<>);
3          -- le type paramètre <> désigne n'importe quel type discret
4  package Ada.Numerics.Discrete_Random is
5          type Generator is limited private;
6          function Random (Gen : Generator) return Result_Subtype;
7          procedure Reset (Gen : in Generator;
8                           Initiator : in Integer);
9          procedure Reset (Gen : in Generator);
10         -- [...]
11 private
12         ... -- not specified by the language
13 end Ada.Numerics.Discrete_Random;
```

B. Implémentation du paquetage Ensemble_Paq

```
1  package body Ensemble_Paq is
2
3     function Ensemble_Vide return Ensemble is
4        E : Ensemble;
5     begin
6        E.N := 0;
7        return E;
8     end Ensemble_Vide;
9
10    procedure Ajouter(E : in out Ensemble; X : in Objet) is
11       I : Integer;
12    begin
13       -- Recherche de l'élément X
14       I := 1;
15       while (I <= E.N) and then not Egal(X,E.T(I)) loop
16          I := I + 1;
17       end loop;
18       if I > E.N then
19          -- X n'est pas dans E : Ajout de X
20          E.N := E.N + 1;
21          E.T(E.N) := X;
22       end if;
23    end Ajouter;
24
25    procedure Supprimer(E : in out Ensemble; X : in Objet) is
26       I : Integer;
27    begin
28       -- Recherche de l'élément X
29       I := 1;
30       while (I <= E.N) and then not Egal(X,E.T(I)) loop
31          I := I + 1;
32       end loop;
33       if I <= E.N then
34          -- X est présent dans E : suppression de X
35          E.T(I) := E.T(E.N);
36          E.N := E.N + 1;
37       end if;
38    end Supprimer;
39
40    function Est_Vide(E : Ensemble) return Boolean is
41    begin
42       return E.N = 0;
43    end Est_Vide;
44
45    function Appartient(X : Objet; E : Ensemble) return Boolean is
46       I : Integer;
47    begin
48       -- Recherche de l'élément X
49       I := 1;
50       while (I <= E.N) and then not Egal(X,E.T(I)) loop
51          I := I + 1;
52       end loop;
53       return I <= E.N;
54    end Appartient;
55
56    function Nb_Elements(E : Ensemble) return Natural is
57    begin
58       return E.N;
59    end Nb_Elements;
60
61 end Ensemble_Paq;
```