

Corrigé

Les téléphones portables doivent être éteints et rangés dans vos sacs.
Le barème (sur 40) est donné à titre indicatif.

Avant propos

Toutes vos réponses doivent être dûment justifiées.

Pensez à présenter clairement vos réponses, en particulier les programmes.

Lisez bien attentivement le sujet avant de répondre aux questions. Toute réponse hors sujet sera considérée comme fausse.

Vous devez impérativement indiquer `.a11` chaque fois que vous aurez affaire à un pointeur. Nous considérerons que l'absence des `.a11` lorsqu'on les attend correspond à une réponse fausse.

Exercice 1 – Un métro à Jussieu (22 points)

Dans le cadre de l'optimisation des enseignements et pour accélérer les déplacements des étudiants et enseignants, on souhaite procéder à la création de lignes de métro sur le campus de Jussieu. Pour cela, il faut disposer de structures de données permettant la description des lignes qui seront installées.

La description des lignes de métro s'appuie sur le type `Une_station` présenté ci après

```
type Une_Station is
  record
    Nom : String (1 .. 20) := (others => ' ');
    Lnom : Natural := 0;
  end record;
```

L'objectif des deux questions suivantes est de construire deux types nous permettant de décrire une ligne.

Question 1 (2 points) :

Rédigez le source permettant de déclarer le type `Tab_Stations`, un tableau non contraint de `Une_station` indexé par des entiers strictement positifs.

Solution de la question 1 de l'exercice 1:

```
type Tab_Stations is array (Positive range <>) of Une_Station;
```

Fin de la solution de la question 1 de l'exercice 1.

On va s'intéresser au type `Ligne`. Ce type possède un discriminant (`Nb_Stations`) qui est un entier positif donnant le nombre total de stations prévu sur la ligne.

Le type `Ligne` possède également les composantes suivantes :

- `Num`, un entier *strictement positif* qui indique le numéro de la ligne,
- `Nom`, une chaîne de 20 caractères initialisée avec des espaces,
- `Lnom`, un entier permettant de spécifier le nombre de caractères exact du nom (quand celui-ci est inférieur à 20 caractères), ce champ est initialisé à 0,
- `Nb_Actuel`, un entier permettant de savoir combien de stations ont été créées dans une variable de type ligne (les stations seront ajoutées les unes après les autres par des primitives définies plus loin dans le sujet), ce champ est initialisé à 0,
- `Les_Stations`, un tableau de stations dont la taille est le nombre de stations prévu pour la ligne.

Question 2 (3 points) :

On souhaite déclarer le type `Ligne` comme un type privé d'un paquetage `Metro`.

Rédigez l'ensemble des déclarations du type `Ligne` en respectant scrupuleusement les instructions qui vous sont données.

Solution de la question 2 de l'exercice 1:

Les choses se déroulent en deux temps, la déclaration publique :

```

|| Les types (1ere partie)
type Ligne
  (Nb_Stations : Positive) is private;

```

La déclaration privée :

```

type Ligne
  (Nb_Stations : Positive) is
  record
    Num          : Positive;
    Nom          : String (1 .. 20)           := (others => ' ');
    Lnom         : Natural                   := 0;
    Nb_Actuel    : Natural                   := 0;
    Les_Stations : Tab_Stations (1 .. Nb_Stations);
  end record;

```

Fin de la solution de la question 2 de l'exercice 1.

À partir des éléments dont on dispose, il s'agit maintenant de compléter la construction du paquetage `Metro`. Nous disposons des éléments de spécification suivants :

```

|| Pour gerer les erreurs
Ligne_Error : exception;

|| Creation d'une ligne et affectation de son numero et de son nom
function Creer_Ligne (
  Nb_Stations : in    Positive;
  Nom         : in    String;
  Num        : in    Positive )
return Ligne;

|| Ajout d'une station sur la ligne. Si on ajoute une station alors que la ligne possede deja le nombre de stations prevu, alors on doit lever
l'exception Ligne_error
procedure Ajouter_Station (
  L       : in out Ligne;
  N_Station : in    String );

|| Recuperer le nombre actuel de stations d'une ligne
function Nombre_De_Stations (
  L : in    Ligne )
return Natural;

|| Recuperer le nombre theorique (maximum) de stations d'une ligne
function Nombre_Theorique_De_Stations (
  L : in    Ligne )
return Positive;

|| Recuperer le nom de la Nieme station sur la ligne. Si N est superieur au nombre de stations creees, il faut lever l'exception Ligne_error.
La fonction ne renvoie que les caracteres significatifs (pas les espaces supplementaires).
function Nom_Nieme_Station (
  L : in    Ligne;
  N : in    Positive )
return String;

|| donner le nombre de stations qui separent deux stations d'une ligne. Les stations sont donnees dans n'importe quel ordre. Il faut retourner
0 si les deux stations specifiees sont identiques. Il faut lever Ligne_error si l'une des deux stations n'existe pas.
function Distance (
  L : in    Ligne;
  S1,
  S2 : in    String )
return Natural;

```

Question 3 (3 points) :

Quelles sont les opérations associées au type `Ligne`. Justifiez brièvement (3 lignes max) votre réponse.

Solution de la question 3 de l'exercice 1:

- " := " et " = " qui sont définis automatiquement avec la notion de type `private`
- `Ajouter_Station`, `Nombre_De_Stations`, `Nombre_Theorique_De_Stations`, `Nom_Nieme_Station` qui sont définies explicitement.

Fin de la solution de la question 3 de l'exercice 1.

Dans la suite, on considère que le type `Ligne` est celui que vous avez défini dans la question 2.

Question 4 (2 points) :

Rédigez le source de la fonction `Creer_Ligne`.

Solution de la question 4 de l'exercice 1:

```
function Creer_Ligne (
    Nb_Stations : in    Positive;
    Nom          : in    String;
    Num         : in    Positive )
return Ligne is
    Ret : Ligne (Nb_Stations);
begin
    Ret.Nom (1..Nom'Length) := Nom;
    Ret.Lnom := Nom'Length;
    Ret.Num := Num;
    return Ret;
end Creer_Ligne;
```

Fin de la solution de la question 4 de l'exercice 1.**Question 5 (3 points) :**

Rédigez le source de la procédure `Ajouter_Station`.

Solution de la question 5 de l'exercice 1:

```
procedure Ajouter_Station (
    L          : in out Ligne;
    N_Station : in    String ) is
begin
    if L.Nb_Actuel = L.Nb_Stations then
        raise Ligne_Error;
    end if;
    L.Nb_Actuel := L.Nb_Actuel + 1;
    L.Les_Stations (L.Nb_Actuel).Nom (1..N_Station'Length) := N_Station;
    L.Les_Stations (L.Nb_Actuel).Lnom := N_Station'Length;
end Ajouter_Station;
```

Fin de la solution de la question 5 de l'exercice 1.**Question 6 (2 points) :**

Rédigez le source de la fonction `Nombre_De_Stations`.

Solution de la question 6 de l'exercice 1:

```
function Nombre_De_Stations (
    L : in Ligne )
return Natural is
begin
    return L.Nb_Actuel;
end Nombre_De_Stations;
```

Fin de la solution de la question 6 de l'exercice 1.**Question 7 (2 points) :**

Rédigez le source de la fonction `Nombre_Theorique_De_Stations`.

Solution de la question 7 de l'exercice 1:

```
function Nombre_Theorique_De_Stations (
  L : in Ligne )
  return Positive is
begin
  return L.Nb_Stations;
end Nombre_Theorique_De_Stations;
```

Fin de la solution de la question 7 de l'exercice 1.

Question 8 (2 points) :

Rédigez le source de la fonction `Nom_Nieme_Station`.

Solution de la question 8 de l'exercice 1:

```
function Nom_Nieme_Station (
  L : in Ligne;
  N : in Positive )
  return String is
begin
  if N > L.Nb_Actuel then
    || Remarque, Nb_Actuel ne dépasse JAMAIS Nb_Stations ;- )
    raise Ligne_Error;
  end if;
  return L.Les_Stations (N).Nom (1..L.Les_Stations (N).Lnom);
end Nom_Nieme_Station;
```

Fin de la solution de la question 8 de l'exercice 1.

Question 9 (3 points) :

Rédigez le source de la fonction `Distance`.

Solution de la question 9 de l'exercice 1:

La programmation est "claire et lisible" mais hélas, on fait deux parcours avec deux appels de la fonction `where_is`.

```
function Distance (
  L : in Ligne;
  S1,
  S2 : in String )
  return Natural is
  Pos1,
  Pos2 : Natural;

  function Where_Is (
    S : in String )
    return Natural is
  begin
    for I in 1 .. L.Nb_Actuel loop
      if Nom_Nieme_Station (L, I) = S then
        return I;
      end if;
    end loop;
    raise Ligne_Error;
  end Where_Is;

begin
  Pos1 := Where_Is (S1);
  Pos2 := Where_Is (S2);
  if Pos1 > Pos2 then
    return Pos1 - Pos2;
  else
    return Pos2 - Pos1;
  end if;
end Distance;
```

Fin de la solution de la question 9 de l'exercice 1.

Exercice 2 – Des pointeurs dans le métro (18 points)

On décide désormais d'élaborer un type `Le_Metro` qui décrira la liste des lignes du campus de Jussieu. La spécification du packaging concerné (déjà commencée dans la partie précédente) s'enrichit comme suit :

```
type Le_Metro is limited private;
```

|| Ajouter une Ligne dans une variable de type `Le_Metro` dans la structure, les ligne sont triées par leur numero et doivent donc etre ajoutées de maniere ordonnée. Si une ligne avec le meme numero existe deja, il faut lever `Ligne_Error`

```
procedure Ajouter_Ligne (
  M : in out Le_Metro;
  L : in Ligne );
```

|| Compter les lignes dans une variable de type `Le_Metro`

```
function Nombre_De_Lignes (
  M : in Le_Metro )
return Natural;
```

|| Recuperer la ligne numero N dans une variable de type `Le_Metro`. Si le numero de ligne correspondant n'existe pas, alors il faut lever l'exception `Ligne_Error`

```
function Recuperer_Ligne_n (
  M : in Le_Metro;
  N : in Positive )
return Ligne;
```

Pour l'implémenter, on dispose les types de données suivants :

```
type Cell_Ligne
  (Nb_Stations : Positive);

type A_Cell_Ligne is access Cell_Ligne;

type Cell_Ligne
  (Nb_Stations : Positive) is
  record
    Contenu : Ligne (Nb_Stations);
    Succ : A_Cell_Ligne;
  end record;

type Le_Metro is
  record
    Premiere_Ligne : A_Cell_Ligne;
  end record;
```

Question 1 (2 points) :

Indiquez la liste des opérations possibles sur le type `Le_metro`. Justifiez votre réponse (4 lignes max).

Solution de la question 1 de l'exercice 2:

Comme le type est `limited private`, on ne dispose pas des opérateurs d'affectation et de comparaison d'égalité.

On dispose par contre des procédures suivantes qui sont définies explicitement : `Ajouter_Ligne`, `Nombre_De_Lignes` et `Recuperer_Nieme_Ligne` qui sont définies explicitement.

Fin de la solution de la question 1 de l'exercice 2.

Question 2 (6 points) :

Écrivez le source de la procédure `Ajouter_Ligne`.

Solution de la question 2 de l'exercice 2:

```

procedure Ajouter_Ligne (
  M : in out Le_Metro;
  L : in Ligne ) is

  Tmp : A_Cell_Ligne :=
  new Cell_Ligne (L.Nb_Stations);
  Parcours : A_Cell_Ligne;

begin
  Tmp.all.Contenu := L;
  if M.Premiere_Ligne = null then
    || cas particulier, c'est la premiere ligne qu'on ajoute, on ne se pose donc pas de questions.
    M.Premiere_Ligne := Tmp;
  elsif M.Premiere_Ligne.all.Contenu.Num <=L.Num then
    || On doit inserer en debut de liste sauf si le numero est le meme que celui de la ligne qu'on insere
    if M.Premiere_Ligne.all.Contenu.Num =L.Num then
      raise Ligne_Error;
    end if;
    Tmp.all.Succ := M.Premiere_Ligne;
    M.Premiere_Ligne := Tmp;
  else
    || Il faut desormais se positionner juste avant la position a inserer
    Parcours := M.Premiere_Ligne;
    while Parcours.Succ /= null and then
      Parcours.Succ.all.Contenu.Num <= L.Num loop
      Parcours := Parcours.Succ;
    end loop;
    if Parcours.Succ = null then
      -- on insere apres le dernier element de la liste
      Parcours.Succ := Tmp;
    else
      || On insere juste apres l'element courant (le successeur peut-etre vide, ca ne pose pas de probleme) sauf si le
      numero de la ligne existe deja.
      if Parcours.Succ.all.Contenu.Num =L.Num then
        raise Ligne_Error;
      end if;
      Tmp.Succ := Parcours.Succ;
      Parcours.Succ := Tmp;
    end if;
  end if;
end Ajouter_Ligne;

```

Fin de la solution de la question 2 de l'exercice 2.**Question 3** (5 points) :

Écrivez le source de la fonction Nombre_De_Lignes que vous devrez programmer *de manière récursive*.

Solution de la question 3 de l'exercice 2:

```

function Nombre_De_Lignes (
  M : in Le_Metro )
return Natural is

  function Int_Nombre_De_Lignes (
    A : A_Cell_Ligne )
  return Natural is

  begin
    if A = null then
      return 0;
    else
      return Int_Nombre_De_Lignes (A.Succ) + 1;
    end if;
  end Int_Nombre_De_Lignes;

begin
  return Int_Nombre_De_Lignes (M.Premiere_Ligne);
end Nombre_De_Lignes;

```

Fin de la solution de la question 3 de l'exercice 2.

Question 4 (5 points) :

Écrivez le source de la fonction `Recuperer_Ligne_N`.

Solution de la question 4 de l'exercice 2:

```
function Recuperer_Ligne_n (  
    M : in Le_Metro;  
    N : in Positive )  
    return Ligne is  
    Tmp : A_Cell_Ligne := M.Premiere_Ligne;  
  
begin  
    || si N = 1, on ne passe pas dans la boucle  
  
    while Tmp /= null loop  
        if Tmp.all.Contenu.Num = N then  
            return Tmp.all.Contenu;  
        end if;  
        Tmp := Tmp.Succ;  
    end loop;  
    raise Ligne_Error;  
end Recuperer_Ligne_n;
```

Fin de la solution de la question 4 de l'exercice 2.