INSA - 1ère année
2007 – 2008

Durée : 1H25min

Interrogation écrite d'informatique
UV 1 – CORRECTION

Nom :

Prénom :

Groupe :

Objectifs Ce contrôle cherche à évaluer la capacité à *lire* et *comprendre* du code Ada, c.-à-d., à maîtriser la *syntaxe* et la *sémantique* du langage.

Consignes

- Aucun document n'est autorisé, sauf le polycopié de cours (livret jaune).
- Toutes vos réponses doivent figurer dans ce document (dans les parties grises).
- Aucune feuille supplémentaire ne sera prise en compte.
- Indiquez vos nom et prénom sur chaque feuille.

- Le corrigé sera mis en ligne dans la journée.

Barème sur 86 points

Exercice	Barème	Note	Total partiel
Exercice 1 – Corriger les erreurs	20 points		
Exercice 2 – Que font ces blocs (I) ?	22 points		
Exercice 3 – Que font ces blocs (II) ?	24 points		
Exercice 4 – Écrire des sous-programmes	20 points		

☞ Chaque exercice contient au moins une partie facile.

☞ Il y a beaucoup de questions, ne restez jamais bloqués sur une question : passez à la suite.

Note finale : / 20

(La note finale est obtenue en divisant la note sur 86 par 4)

Exercice 1	20 points	Corriger les erreurs

- Pour chaque extrait de programme ci-dessous, le compilateur détecte une erreur.
- Le message d'erreur est donné et une flèche pointe vers l'endroit **où le compilateur s'aperçoit d'une erreur**.

☞ Écrivez dans le cadre la ou les lignes corrigées (en mettant leur numéro devant)
☞ Par exemple, s'il faut modifier la ligne 8, écrivez la nouvelle ligne ainsi : `8 X := Y + 20;`

Chaque question est sur 4 points.

```
1  with Centrale_Inertie ;
2  with Gouvernes ;
4  procedure Pilote is
6      procedure Tourner_Vers (Cap : Float) is
7      begin
9          Incliner_Ailerons (15.0) ;
10         while abs(Centrale_Inertie.Cap - Cap) > 10.0 loop
11             null ;
12         end loop ;
13         Incliner_Ailerons (0.0) ;
15     end Tourner_Vers ;
⋮      ⋮
```

"Incliner_Ailerons" is not visible
non-visible declaration at gouvernes.ads:14

```
9  Gouvernes.Incliner_Ailerons (15.0);
```

```
1  with INSA_Air ;
3  procedure Pilote is
5      Volume_A_Ajouter : Float ;
7  begin
9      Volume_A_Ajouter := 2000.0 - INSA_Air.Volume_Carburant ;
10     INSA_Air.Ajouter_Carburant (Volume_A_Ajouter) ;
12 end Pilote ;
```

expected type "Standard.Integer"
found type "Standard.Float"

```
10 INSA_Air.Ajouter_Carburant (Integer(Volume_A_Ajouter));
```

```
1  with INSA_Air ;
3  procedure Pilote is
5      procedure Decoller  is
6      begin
7          INSA_Air.Rouler_Vers_Piste ;
8          INSA_Air.Attendre_Autorisation ;
9          INSA_Air.Decollage ;
10     end Decoller ;
12 begin
14         INSA_Air.Decoller ;
16 end Pilote ;
```

"Decoller" not declared in "Insa_Air"

```
14 Decoller;
```

```
1  with INSA_Air ;
2  with Securite ;
4  procedure Exo is
5      package S renames Securite ;
6  begin
8      S.Assert(S.Moteur_OK(1), " Moteur 1 casse ") ;
9      S.Assert(S.Moteur_OK(2), " Moteur 2 casse ") ;
10     S.Assert((INSA_Air.Volume_Carburant > 6000.0)), " Pas assez de carburant ") ;
12 end Exo ;
```

"," should be ";"

```
10 Securite.Assert((INSA_Air.Volume_Carburant > 6000.0), "Pas assez de carburant");
```

```
1  with INSA_Air ;
2  with Securite ;
4  procedure Exo is
5      package S renames Securite ;
6  begin
8      S.Assert(S.Moteur_OK(1) and S.Moteur_OK(2) and
9               S.Moteur_OK(3) and S.Moteur_OK(4)
10              " Moteur casse ") ;
12 end Exo ;
```

missing argument for parameter "Message" in call to "Assert" declared at securite.ads:10

```
10 , " Moteur casse ");
```

Nom : Prénom : Groupe :

Exercice 2	22 points	Que font ces blocs (I) ?

- Les blocs ci-dessous sont des **extraits** de programmes. On suppose que le compilateur ne détecte pas d'erreur.
- Txt.Put permet d'afficher des messages à l'écran.
- ITxt renomme l'acteur Ada.Integer_Text_IO

```
for J in 1..12 loop
   Txt.Put ("Bar !") ;
   Txt.Put ("Foo !") ;
   Txt.Put ("Bar !") ;
end loop ;
```

- Combien de fois est affiché "Foo" ? **12** **/1**
- Combien de fois est affiché "Bar" ? **24** **/1**
- Doit-on déclarer J avant le **begin** ? **Non** **/1**

```
-- A et B sont deux variables entières
for H in A..B loop
   Txt.Put ("Moo !") ;
end loop ;
```

- Combien de fois est affiché "Moo" lorsque…
- A vaut 1 et B vaut -4 ? **0** **/1**
- A vaut 1 et B vaut 1 ? **1** **/1**
- A vaut 11 et B vaut 14 ? **4** **/1**

```
-- A, B, C sont des variables déclarées
   A := 6 ;
   B := A * A ;
   A := A + 1 ;
   Txt.Put ("Bla") ;
   B := B + 1 ;
   C := A = B ;
```

- Quel est le type de A ? **Integer** **/1**
- Que vaut B lorsque le programme affiche "Bla" ? **36** **/1**
- Que vaut B à la fin du bloc ? **37** **/1**
- Quel est le type de C ? **Boolean** **/1**

```
Txt.Put ("Zou !") ;
Vitesse := 280 ;
if Vitesse > 360 then Txt.Put ("Trop Vite !") ;
else Txt.Put ("Trop lent !") ;
end if ;
Vitesse := 560 ;
Vitesse := Vitesse - 200 ;
```

- Quels sont les messages affichés par cet extrait de programme ? **/4**

Zou ! Trop lent !

```
X := 1 ;
while X <= 4 loop
   Txt.Put ("Ouf") ;
   X := X * 2 ;
end loop ;
```

- Combien de fois est affiché "Ouf" ? **3 fois** **/2**
- Que vaut X à la fin du bloc ? **8** **/1**
- Combien de fois est affiché "Ouf" si l'on remplace la première ligne par `X := 6` ? **0 fois** **/1**

Nom : Prénom : Groupe :

```
-- Rappel : I mod 2 = 0 teste si I est pair
X := 0 ;
for I in 1..10 loop
   if I mod 2 = 0 then X := X + 1 ;
   end if ;
end loop ;
```

- Pendant l'exécution du bloc, combien de fois est appelée la fonction modulo ? **10** **/2**
- Que vaut X à la fin du bloc ? **5** **/2**

Exercice 3 — 24 points — Que font ces blocs (II) ?

Complétez les déclarations de constantes suivantes (la valeur manquante est toujours mille).

Mille_Entier : **constant** **Integer** := **1000** ; **/1**

Mille_Reel : **constant** **Float** := **1000.0** ; **/1**

Mille_En_Toutes_Lettres : **constant** **String** := **"Mille"** ; **/1**

```
-- Moyenne est une variable réelle
Txt.Put (" Combien de notes a moyenner ? ") ;
ITxt.Get (Nombre_Notes) ;
Moyenne := 0.0 ;
for No in 1..Nombre_Notes loop
   Txt.Put (" Note ? ") ;
   ITxt.Get (Note) ;
   Moyenne := Moyenne + Float(Note / Nombre_Notes) ;
end loop ;
```

Quelles sont les séquences que l'utilisateur peut taper lorsqu'il exécute ce programme ? (cocher)

☐ 3 8 10.5 14

☑ 4 10 15 11 6

☐ 1.25 11 **/2**

La moyenne calculée par ce programme est (cocher)

☐ Rigoureuse

☐ Légèrement imprécise (expliquer ci-dessous)

☑ Largement fausse (expliquer ci-dessous)

/2

La division Note / Nombre_Notes est une division entière. Si Nombre_Notes > 20 tout le monde a 0 de moyenne !

F est une variable de type Float et I une variable de type Integer, cocher les lignes correctes :

☑ Ada.Integer_Text_IO.Get (I) ;
☐ Ada.Integer_Text_IO.Get(F) ;
☐ Ada.Integer_Text_IO.Get (Integer(F)) ;
☐ Ada.Text_IO.Get(F) ;
☑ Ada.Float_Text_IO.Get(F) ; **/3**

☑ Ada.Integer_Text_IO.Put (I) ;
☐ Ada.Integer_Text_IO.Put(F) ;
☑ Ada.Integer_Text_IO.Put (Integer(F)) ;
☐ Ada.Text_IO.Put(F) ;
☑ Ada.Float_Text_IO.Put(F) ; **/2**

```
X := 120 ;
Y := 30 ;
A := True ;
B := (X > Y) and (X <= 140) ;
C := not (5 * Y > X) ;
if B and C then
   D := not (B and C) ;
else
   D := False ;
end if ;
Txt.Put ("Poz") ;
C := B or C ;
B := not B ;
D := not (D=D) ;
```

- Que valent B, C, et D lorsque le programme affiche “Poz” ?

  B = **True** C = **False**

  D = **False** **/2**

- Que valent B, C, et D à la fin du bloc ?

  B = **False** C = **True**

  D = **False** **/2**

```
X := 55 ;
Y := -25 ;
if (X > 100) or (X < 60) then
   Txt.Put ("Bar") ;
   Y := X ;
end if ;
if (Y < X) then
   Txt.Put ("Foo") ;
else
   Txt.Put ("Kak") ;
end if ;
```

- Qu’affiche ce programme ? **/2**

  BarKak

Le bloc suivant fait appel à un outil de mesure accessible via l’acteur Outil.
Le but est de déterminer si la pièce mesurée est un carré. La réponse est affichée directement à l’écran.

```
-- Les variables réelles X et Y sont déclarées avant le begin
X := Outil.Mesurer_Largeur ;
Y := Outil.Mesurer_Hauteur ;
if X - Y > 0.0 then Txt.Put (" Trop large ") ;
elsif X - Y < 0.0 then Txt.Put (" Trop haut ") ;
else Txt.Put (" C'est un carre ! ") ;
end if ;
```

- Ce programme remplit-il sa tâche ? (Si c’est non, justifier) **/3**

  Ce programme n’est pas acceptable car il oublie de prendre en compte l’erreur de mesure de l’outil.

```
-- N est une constante déclarée
for X in 1..N loop
   if X * X mod 12 = N mod (1 + X / 13)
   then Txt.Put ("Foo") ;
   else Txt.Put ("Bar") ;
   end if ;
end loop ;
```

- Information inutile : X mod Y renvoie le reste de la division euclidienne de X par Y

- Ce programme affiche 19 fois “Foo” et 101 fois “Bar”. Combien vaut N ? **120**

  On ne se sert pas de la formule pour répondre. **/3**

Nom : Prénom : Groupe :

Exercice 4	20 points	Écrire des sous-programmes

En supposant que la fonction valeur absolue ne soit pas déjà définie en Ada, écrivez une fonction qui renvoie la valeur absolue d'un nombre réel :

```
function abs (X : Float) return Float is
begin
    if X < 0.0 then return -X ;
    else return X ;
    end if ;
end abs ;
```

L'acteur Outil contient la fonction suivante :

```
function Mesurer (Dim : Integer) return Float ;
```

L'argument Dim permet d'indiquer la dimension que l'on souhaite mesurer :

Dim	Dimension mesurée
1	Largeur
2	Longueur
3	Hauteur

Écrire une procédure Tout_Mesurer qui permet de récupérer d'un coup les trois dimensions. Cette procédure peut utiliser l'acteur Outil si nécessaire (on suppose que **with** Outil ; est écrit au début du programme courant).

```
procedure Tout_Mesurer (Larg, Long, Haut : out Float) is
begin
    Larg := Outil.Mesurer (1) ;
    Long := Outil.Mesurer (2) ;
    Haut := Outil.Mesurer (3) ;
end Tout_Mesurer ;
```