INSA - 1ère année
2008 – 2009

Durée : 50min

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.
- Aucune feuille supplémentaire ne sera prise en compte.
- Indiquez vos nom et prénom sur chaque feuille.

- Le corrigé sera mis en ligne à la fin du contrôle.

Barème sur 40 points

Exercice	Barème	Note	Total partiel
Exercice 1 – Compréhension générale	4 points		
Exercice 2 – Typage	8 points		
Exercice 3 – Typage	8 points		
Exercice 4 – Sémantique (I)	10 points		
Exercice 5 – Sémantique (II)	10 points		

☞ Ne restez jamais bloqués sur une question : avancez

Note finale : / 40

Exercice 1	4 points	Compréhension générale

Q1 – J'ai écrit un programme nommé Mission1 qui calcule et affiche des nombres premiers.

Où dois-je cliquer pour le **compiler** ?
- ☐ Sur le fichier mission1.ali
- ☐ Sur le fichier b~mission1.adb
- ☐ Sur le fichier upload-exe
- ☐ Sur le fichier mission1.adb
- ☐ Sur le fichier mission1-exe
- ☐ Sur le fichier mission1.o
- ☐ Sur le fichier gada-text_io.ads
- ☑ Sur la commande BUILD du menu ADA de emacs.

Où dois-je cliquer pour **l'exécuter** ?
- ☐ Sur le fichier mission1.ali
- ☐ Sur le fichier b~mission1.adb
- ☐ Sur le fichier upload-exe
- ☐ Sur le fichier mission1.adb
- ☑ Sur le fichier mission1-exe
- ☐ Sur le fichier mission1.o
- ☐ Sur le fichier gada-text_io.ads
- ☐ Sur la commande BUILD du menu ADA de emacs.

/1

Les différentes phases de l'écriture d'un sous-programme sont (l'ordre n'est pas strict) :

1. Écriture de l'algorithme sur papier
2. Test de l'algorithme sur papier
3. Codage du sous-programme en Ada, sur machine
4. Compilation du sous-programme
5. Codage de (nombreux) tests en Ada
6. Compilation des tests
7. Exécution des tests

Q2a – Si la compilation du sous-programme (phase 4) n'affiche pas d'erreur, que puis-je en déduire ? *Cochez une ou plusieurs cases de la **première** colonne.*

	Q2a	Q2b	Q2c
L'algorithme est correct	☐	☐	☑
Le sous-programme Ada correspond fidèlement à l'algorithme	☐	☐	☑
Le sous-programme Ada ne contient pas d'erreur de syntaxe ni de typage	☑	☑	☑
Le sous-programme Ada est correct : il donnera le bon résultat	☐	☐	☑
Les tests Ada ne contiennent pas d'erreur de syntaxe ni de typage	☐	☑	☑
Les tests afficheront les valeurs attendues	☐	☐	

/2

Q2b – Si la compilation du programme et des tests (phase 6) n'affiche pas d'erreur, que puis-je en déduire ?
Cochez maintenant une ou plusieurs cases de la seconde colonne.

Q2c – En supposant que les tests couvrent tous les cas possibles, qu'ils compilent sans erreur, et que leur exécution (phase 7) produit bien les résultats attendus, que puis-je raisonnablement en déduire ? *Cochez des cases de la troisième colonne.*

Q3 – Répondez par **V** (Vrai) ou par **F** (Faux) : **/1**

- ◦ Si un problème est bien spécifié, il y a au maximum un seul algorithme qui le résout **F**
- ◦ L'algorithmique, c'est l'art de décomposer un problème complexe en opérations simples **V**
- ◦ Les professionnels (par exemple chez Google), ne conçoivent pas d'algorithmes car ils utilisent des concepts plus puissants. **F**
- ◦ On suppose qu'une ligne de programme s'exécute en strictement moins d'1μs ($10^{-6}s$). Alors un programme qui tient sur une page de 50 lignes s'exécute en moins de 50 μs. **F**

Nom : Prénom : Groupe :

Exercice 2	8 points	Typage

Rappel : La notation $\vdash$ e : τ signifie « Le terme e est correctement typé et son type est τ ».

Complétez les jugements de typage suivants. Certains termes sont incorrects, dans ce cas vous devez les barrer $\vdash$ ~~e~~ :

On suppose que A et B sont deux variables de type *Float*, et que la procédure Fail et fonction abs sont bien définies :

$\vdash$ A	:	***Float***	$\vdash$ "B"	:	***String***
$\vdash$ A > 10.0	:	***Boolean***	$\vdash$ ~~abs (X < 10.0)~~	:	
$\vdash$ A < B	:	***Boolean***	$\vdash$ "A * B"	:	***String***
$\vdash$ "A^2 + X^2 < 0"	:	***String***	$\vdash$ "AB"	:	***String***
$\vdash$ abs (A * B)	:	***Float***	$\vdash$ abs (A) * B	:	***Float***
$\vdash$ ~~abs (AB) < 10.0~~	:		$\vdash$ ~~10.0 < A < 20.0~~	:	
$\vdash$ abs (A) > Float(10)	:	***Boolean***	$\vdash$ Float'Image(abs(A))	:	***String***
$\vdash$ Integer(A) = Integer(B)	:	***Boolean***	$\vdash$ A := abs(A)	:	bloc

Exercice 3	8 points	Typage

Q1 – Mêmes consignes qu'à l'exercice 2. **/4**

$\vdash$ ~~20.0 > A **and** B~~	:		$\vdash$ abs(abs(A * B) / 100.0)	:	***Float***
$\vdash$ Fail (abs(A) < 10.0)	:	bloc	$\vdash$ Fail (**not** (abs(A) > abs (B)))	:	bloc
$\vdash$ ~~abs(A) := 10.0~~	:		$\vdash$ **if** abs(A) > 10.0 **then** A := 10.0 ; **end if** ;	:	bloc
$\vdash$ ~~**not** (Fail (A > 20.0))~~	:		$\vdash$ ~~**if** Fail(True) **then** A := 10.0 ; **end if** ;~~	:	

Q2 – Un programme sans erreur contient les deux lignes suivantes : **/4**

`X : Float ;` (avant le **begin**)
`X := Mesure ;` (après le **begin**)

L'identifiant Mesure est défini avant le **begin**. La liste ci-dessous contient des définitions, parfois partielles. Cocher la case lorsque la définition correspondante pourrait se trouver avec le **begin**.

- ☑ `Mesure : constant Float := 100.0 ;`
- ☐ `procedure Mesure is ...`
- ☐ `procedure Mesure (X : Float) is ...`
- ☐ `procedure Mesure (Mesure : Float) is ...`
- ☑ `function Mesure return Float is ...`
- ☐ `function Mesure (X : Float) return Float is ...`
- ☑ `Mesure : Float ;`
- ☐ `with Mesure ;`
- ☐ `type Mesure is record ...`

Exercice 4 — 10 points — Sémantique (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 : si ⊢ `e` : *String*, alors ⊢ `Txt.Put (e)` : bloc.

```
J := 5 ; -- J est une variable
while J > 0 loop
   J := J - 1 ;
end loop ;
```

- Combien vaut J en sortie de boucle ? **0** **/1**

```
-- A est une variable
A := 10 ;
while A > 12 loop
   Txt.Put("Foo !") ;
   A := A + 1 ;
end loop ;
```

- Combien de fois est affiché "Foo" ? **0** **/1**
- Même question si on remplace 10 par 15 : $+\infty$ **/2**

On suppose que `Mesurer` est une fonction sans argument. Cocher les affirmations vraies.

```
X := Mesurer ;
N := 0 ;
while X < 35 and N < 100 loop
   X := Mesurer ;
   N := N + 1
end loop ;
```

☐ On peut remplacer cette boucle **while** par une boucle **for**. **/2**

En sortie du bloc **while**, il est possible d'avoir : **/2**

- ☑ N vaut 0
- ☑ N vaut 1
- ☑ N vaut 100
- ☐ N vaut 101
- ☑ X vaut 0
- ☑ X vaut -20
- ☑ X vaut 50
- ☑ X vaut 100

Complétez le programme suivant pour qu'il affiche tous les entiers de 1 à 8000. **/2**

```
for J in 1 .. 8000 loop

    Txt.Put ( Integer'Image(J) ) ;

end loop ;
```

Nom : Prénom : Groupe :

Exercice 5	10 points	Sémantique (II)

On suppose que la fonction `Carre` renvoie son argument (de type *Integer*) au carré.

```
for Z in 1 .. 15 loop
   if 100 = Carre(Z) then
      if abs(Z) > 10 then
         Txt.Put ("Moo !") ;
      end if;
   else
      Txt.Put ("Bar !") ;
   end if ;
end loop ;
```

- Combien de fois est invoquée la fonction Carre ? **15** **/1**
- Combien de fois est invoquée la fonction abs ? **1** **/1**
- Combien de fois est affiché “Moo” ? **0** **/1**
- Combien de fois est affiché “Bar” ? **14** **/1**

Les variables du programme suivant sont dûment déclarées.
La fonction `Mesurer` est sans argument. **/4**

```
Mes_A := Mesurer ;
X := Mes_A + 1 ;
for N in 1..40 loop
   Mes_B := Mes_A ;
end loop ;
Y := Mes_B + 1 ;
if X = Y then Txt.Put ("Yal !") ;
else Txt.Put ("Blop !") ;
end if ;
```

- Arrive-t-il que ce programme affiche “Yal” ? **Oui**
- Arrive-t-il que ce programme affiche “Blop” ? **Non**
- Combien de mesures sont effectuées par ce programme ? **1**

```
for Alpha in 1..X loop
   for Beta in 1..Alpha loop
      Txt.Put ("K") ;
   end loop ;
end loop ;
```

- Ce programme affiche des “K”. Combien exactement ? (Résultat à exprimer en fonction de X).

$1 + 2 + \ldots + X$, **c.-à-d.,** $X \times \frac{X+1}{2}$ **/2**