TD COMPILATION — FEUILLE 1

Exercice 1. Soit la grammaire G suivante :

$$S \to SaSaS \qquad\qquad S \to bS \qquad\qquad S \to \epsilon$$

▷ **Question 1** *Caractériser le langage engendré par cette grammaire. Montrer qu'elle est ambiguë et proposer une grammaire non ambiguë pour ce langage.*

Exercice 2. Donner des grammaires engendrant les langages suivants :

1. $\{a^n b^p \mid n \geq p \geq 0\}$
2. $\{a^n b^p \mid n \neq p\}$
3. $\{a^n b^p \mid 2p \geq n \geq p\}$
4. $\{a^n b^p c^q \mid n + p = q\}$

▷ **Question 1** *Discuter du caractère ambigu ou non ambigu des grammaires proposées.*

Exercice 3. En Ada l'instruction d'affectation est de la forme :

inst → place **:=** exp

Rappel : en Ada, en partie gauche d'une affectation, on peut trouver les éléments suivants :
- un identificateur (ex : `X := 1`),
- une sélection d'enregistrement (ex : `C.Reel := 0`),
- une "déréférence" pour un type accès (ex : `P.all := 1` où `P` est de type accès vers un entier),
- un composant indexé (ex : `M(I,J+1) := 0` où `M` est un tableau d'entiers à deux dimensions).

▷ **Question 1** *Écrire une grammaire décrivant la catégorie syntaxique place. On utilisera sans les définir les catégories syntaxiques idf (identificateurs) et exp (expressions). Préciser le vocabulaire terminal utilisé.*

▷ **Question 2** *La sémantique de l'affectation est définie en Ada de la manière suivante : pour l'exécution d'une instruction d'affectation on évalue tout d'abord le nom de la variable et l'expression, dans un certain ordre, lequel n'est pas défini dans le langage. Donnez des exemples d'affectation qui produisent des résultats différents suivant l'ordre d'évaluation choisi.*

Exercice 4.

▷ **Question 1** *Les instructions suivantes sont-elles acceptables par un compilateur* Ada *conforme à la norme ? Si oui, quel est le résultat de leur exécution ?*

1. ```
   for i in 1 .. 5 loop
      i  := i-1; put(i);
   end loop;
   ```
2. ```
   declare n :natural  := 5;
   begin
     for i in 1 .. n loop
       n  := i+1; put(n);
     end loop;
   end;
   ```

3. ```
   declare n :natural  := 5;
   begin
     for i in n+1 .. n loop
       put(i);
     end loop;
   end;
   ```

▷ **Question 2** *Donner une traduction de l'instruction for à l'aide d'autres primitives de ce langage. Même question pour l'instruction for du langage C.*

Exercice 5. Le langage C++, en plus des aspects objet, se différencie du langage C en offrant une gestion plus fine des espaces de noms et une politique de typage plus sûre. En particulier les booléens, les types énumérés et les classes deviennent des types à part entière. En C++ le mot réservé typedef est inutile.

▷ **Question 1** *Discuter des différences en C et en C++ du programme suivant :*

```
typedef enum { Vert, Jaune, Rouge } Couleur ;
typedef enum  { Vert2, Jaune2, Rouge2 } Couleur2 ;

Couleur f(Couleur c) {
   switch (c) {
 Vert: return Jaune ;
 Jaune: return Rouge ;
 Rouge: return Vert ; } }

int main() {
  Couleur a = Vert ;
  Couleur b = f(a) ;
  Couleur2 c = Vert2 ;
  Couleur d = f(c) ;
  Couleur2 e = f(a) ; }
```

Une autre différence réside dans la représentation des éléments d'un type énuméré (annexe 3 de la norme C++). En particulier il est dit [Annexe 3, 3.1.5] : *en C++, le type d'un énumérateur est son nom. En C, le type d'un énumérateur est int.* Par exemple on a :

```
   enum e { A };
   sizeof(A) == sizeof(int)        //  in C
   sizeof(A) == sizeof(e)          //  in C++
  /* and sizeof(int) is not necessary equal to sizeof(e) */
```

Finalement en C++ les valeurs d'un type énuméré peuvent être implicitement converties en int. La réciproque est fausse. Un int peut être explicitement converti en une valeur d'un type énuméré (static_cast<nom-type-enum>(i)), la conversion pouvant provoquer un comportement indéfini.

▷ **Question 2** *Soit le programme C suivant :*

```
typedef enum {Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche} jour ;
int main() {
  int i = 0 ;
```
```

```
  Jour j = Lundi ;
  j = i ;
  i = 7 ;
  j = i ;  }
```

Corriger ce programme afin qu'il soit admis par un compilateur C++. Que donne le résultat à l'exécution de ce programme et de sa version corrigée en C++ ? Justifier les réponses.

Exercice 6. Que peut-on dire des exemples Java suivants : sont-ils admis par le compilateur ? Si oui quels sont les résultats de l'exécution ?

▷ **Question 1**

```
class O1 { int f1() {return 1 ; }}
class O2 { int f2() {return 2 ; }}

class Securite1 {
   public static void main (String args[])
   { int i = 1 ; O1 o1 ; O2 o2 ;
     if (i==1) o1=new O1() ; else o2 = new O2() ;
     if (i==1) System.out.println(o1.f1()) ; else System.out.println(o2.f2()) ; }}

class Securite2 {
   public static void main (String args[])
   { int i = 1 ; Object o ;
     if (i==1) o=new O1() ; else o = new O2() ;
     if (i==1) System.out.println(((O1)o).f1()) ; else System.out.println(((O2)o).f2()) ; }}
```

▷ **Question 2**

```
class A {
   public void m (B bb, A aa) {System.out.println (" 1 ") ; } }

class B extends A  {
   public void m (B bb, A aa) {System.out.println (" 2 ") ; }
   public void m (A aa, B bb) {System.out.println (" 3 ") ; } }

class C {
   public static void main  (String[] args) {
       A a = new A() ;
       B b = new B() ;
       a.m(b, a) ;
       a.m(b, b) ;
       b.m(b, a) ;
       b.m(a, b) ;
       b.m(b, b) ; }}
```

▷ **Question 3** *Même exemple en remplaçant la classe C par :*

```
class C {
   public static void main  (String[] args) {
       A a = new A() ;
       B b = new B() ;
       A aa = b ;
```

```
aa.m(b, a) ;
((B) aa).m(a, b) ;
((B) aa).m(aa, b) ;
aa.m(b, aa) ; } }
```