TD Vérification statique et vérification dynamique — FEUILLE 1

Exercice 1.

▷ **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. La traduction proposée ne devra pas provoquer plus d'exception que l'instruction for initiale. Expliquer quelles vérifications statiques sont nécessaires et comment les spécifier.*

▷ **Question 3** *Même question pour l'instruction for du langage C.*

Exercice 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.

▷ **Question 1** *Donner un exemple d'affectation qui produit des résultats différents suivant l'ordre d'évaluation.*

▷ **Question 2** *Que peut-on dire des langages C et Java ? Quels sont les avantages et les inconvénients des langages imposant (n'imposant pas) un ordre d'évaluation ?*

Exercice 3. On considère le langage suivant :

Var	→	**idf** | ***** Var
Exp	→	Var | **&** Var | **null**
Affect	→	Var **:=** Exp

▷ **Question 1** *Quelles sont les valeurs associées à x, p et $*p$ après la séquence :*

$$x := 2 \ ; \ p := \&x \ ; \ *p := 3 \ ;$$
   ```

▷ **Question 2** *On rappelle les définitions vues en cours pour la fonction $\mathcal{A}$ calculant l'adresse d'une variable et la fonction $\mathcal{V}$ calculant la valeur d'une expression :*

Definition
$\mathcal{A}_{e,m}(nom) = e(nom)$
$\mathcal{A}_{e,m}(*var) = \mathcal{V}_{e,m}(var)$

Definition	*Condition*
$\mathcal{V}_{e,m}(\texttt{null}) = \texttt{ad\_nulle}$	
$\mathcal{V}_{e,m}(var) = m(\mathcal{A}_{e,m}(var))$	$\mathcal{A}_{e,m}(var) \neq \texttt{ad\_nulle}$

En utilisant les fonctions $Env : Nom \rightarrow Adresse$ et $Mem : Adresse \rightarrow Value$ compléter les tableaux ci-dessus pour prendre en compte la nouvelle construction $Exp \rightarrow \&\ Var$.

▷ **Question 3** *Dérouler le calcul de la mémoire obtenue après la séquence :*

$$x := 2\ ;\ p := \&x\ ;\ *p := 3\ ;$$

pour les valeurs initiales $e = \{x \mapsto ad_1, p \mapsto ad_2\}$ et $m_0 = \{ad_1 \mapsto 0, ad_2 \mapsto ad_3, ad_3 \mapsto 0\}$

▷ **Question 4** *En utilisant les définitions proposées calculer les expressions $\mathcal{V}_{e,m}(\& * p)$ et $\mathcal{V}_{e,m}(*\&p)$, avec p une variable quelconque.*

▷ **Question 5** *La norme du langage C donne la définition suivante :*

- *the unary & operator yields the adress of its operand. If the operand is the result of a unary $*$ operator, neither that operator nor the & operator is evaluated and the result is as if both were omitted.*

La définition précédente est-elle en accord avec la norme C ?

Exercice 4. On s'intéresse dans cet exercice à définir la sémantique de différents modes de passage de paramètres.

Soit p(t x)$\hat{=}$ S la définition d'une procédure avec x le paramètre formel de type t et S le corps de la procédure, dans un langage L suivant la même syntaxe que le lange C. On considère les différentes sémantiques suivantes pour l'appel p(v), avec v une place.

1. p(v) est définie par la séquence suivante : {t x ; x=v ; S }
2. p(v) est définie par {t *x ; x :=&v ; $[x \leftarrow *x]S$ }

La notation $[x \leftarrow t]S$ désigne le texte dans lequel les occurrences libres de x sont remplacées par le terme t.

On considère le programme suivant :

```
#include<stdio.h>
int n=2;
int t[10];
int i;
int p (int x) {x=x+1; n=1; return(x);}
int main()
{  for (i=0; i<10; i++) {t[i]=i;}
    n=2;
    i = p(t[n]);
```

```
    printf("%i\n", i);
    printf("%i\n", t[2]);
}
```

▷ **Question 1** *A quelle forme de passage de paramètre correspondent ces 2 définitions ? Appliquer ces définitions à l'exemple ci-dessus. Que peut-on constater ?*

▷ **Question 2** *Certains langages, dits paresseux, n'évaluent les paramètres que lorsque c'est nécessaire. On définit alors p(v) par $[x \leftarrow v]S$, i.e. la suite d'instruction S dans laquelle les occurrences de x sont remplacées par le terme v. Comparer ce mode de passage avec les 2 premiers.*