Programmer en ADA

Traduction des arbres programmatiques en programmes ADA

1 Un programme ADA

Tout ce qui est mis après '--' n'est pas compilé : c'est un commentaire.

Un programme en ADA est composé comme suit :

```
--Programme ADA

with Ada.integer_Text_Io ; use Ada.integer_Text_Io; --pour lire/écrire des entiers
with Ada.float_Text_Io ; use Ada.float_Text_Io; --pour lire/écrire des flottants
with Text_Io; use Text_Io; --pour lire/écrire des caractères
with Ada.Numerics.Elementary_Functions; use Ada.Numerics.Elementary_Functions; --pour
utiliser les fonctions mathématiques (sqrt, sin, ...)

procedure nom_à_mettre is --Début du programme que vous avez nommé "nom_à_mettre "

--déclaration des assertions de déclaration de l'arbre programmatique

begin
--instructions séquentielles détaillées dans l'arbre programmatique
end nom_à_mettre;
```

2 Compilation et exécution

2.1 Sauvegarde du programme

Votre programme doit être sauvé dans un fichier de même nom que le programme lui-même avec l'extension .adb (exemple : *nom_à_mettre*.adb).

2.2 Compiler (sous Linux)

Sous emacs, il suffit de choisir dans la barre de menu "ADA ", et de cliquer ensuite sur "Compile file ".

Si vous n'êtes pas sous emacs, vous pouvez taper directement dans votre shell la ligne de commande suivante :

gnatmake -o *nom_à_mettre* *nom_à_mettre*.adb -g -cargs -gnatq -gnatQ -bargs -largs

2.3 Exécuter (sous Linux)

Sous emacs, il suffit de choisir dans la barre de menu "ADA ", et de cliquer ensuite sur "run ".

Si vous n'êtes pas sous emacs, vous avez généré (avec la ligne de commande précédente) un fichier exécutable du nom *nom_à_mettre* , il vous suffit alors dans votre shell de taper la commande :

./nom_à_mettre

3 Traduction de l'arbre programmatique en programme ADA

3.1 Déclaration des variables

Les assertions de déclaration regroupent toutes les variables qui sont utilisées dans le programme, il faudra les "déclarer " de la manière suivante :

nom_variable **:** *type_variable* **;**

	type_variable
Entiers	integer
réels	float
caractères	character
booléens	boolean (prendra 2 valeurs *True*, *False*)
Tableaux de *N* éléments	Array(1..*N*) of *type_variable*

Exemples :

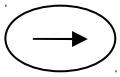
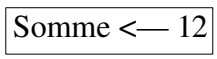
a entier => a **:** integer **;**

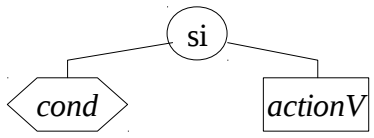
T tableau de 50 réels => T **:** array(1..50) of float **;**

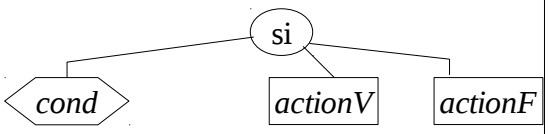
Attention !!! En ADA, l'écriture des noms de variable ou de programme doit respecter certaines règles : lettres, chiffres (0 à 9) et trait bas sont autorisés et le nom doit commencer par une lettre. (Exemple : « fonction-1 » est interdit, mais « fonction_1 » est autorisé) .

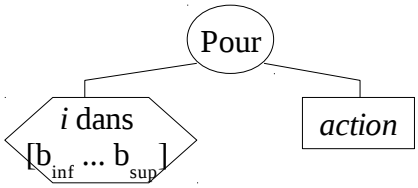
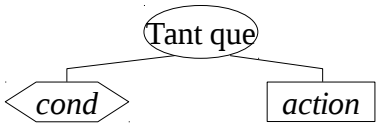
3.2 Structures de l'arbre programmatique

Tous les caractères ou mots en gras sont des mots réservés dans le langage ADA.

	Traduction en ADA	Remarques
	Pas de traduction directe : on traduira les expressions de gauche à droite séquentiellement	On terminera chaque instruction par un **;**
	Somme **:=** 12 **;**	Ici on dit qu'on affecte à la variable *Somme* l'expression[§] 12.
lire(x)	**get**(x) **;**	C'est une instruction à part entière (on met ;)
afficher("x ")	**put**("x ") **;**	Attention on affiche à l'écran le mot **X**
afficher(x)	**put**(x) **;**	Attention on affiche à l'écran la **valeur contenue** dans le mot **X**

	**if** *cond* **then** *actionV* **;** **end if;**	

	**if** *cond* **then**
	  actionV **;**
	else
	  actionF **;**
	end if;	

	**for** *i* **in** b_{inf} **..** b_{sup} **loop** *action* **;** **end loop;**	Attention en ADA l'indice de boucle (ici nommé i) n'a pas besoin d'êtte déclaré (cf exemple 4.)
	**while** *cond* **loop** *action* **;** **end loop;**	

[§]Expression : une expression peut être réduite à une seule constante (comme dans le tableau) , mais aussi être composée d'une ou plusieurs opérations. L'expression sera calculée et sa valeur pourra alors être affectée à la variable donnée.

Exemple : a **:=** 3***b** ; dans un premier temps l'ordinateur multipliera la valeur de b par 3, le résultat de l'expression dépendra donc de la valeur stockée dans la variable b ; puis dans un deuxième temps la valeur de l'expression sera affectée à (stockée dans) la variable a.

Une expression a un certain type et on ne pourra pas affecter à une variable le résultat d'une expression de type différent (exemple : si dans notre exemple précédent b est un entier, alors la multiplication sera la multiplication de 2 entiers, donc l'expression est entière et ne peut être affectée à une variable réelle par exemple.)

3.3 Exemples du cours

1. Moyenne de 3 nombres

```
--Programme Moyenne de 3 nombres

--with Ada.integer_Text_Io ; use Ada.integer_Text_Io; --pour lire/écrire des entiers
with Ada.float_Text_Io ; use Ada.float_Text_Io; --pour lire/écrire des flottants
with Text_Io; use Text_Io; --pour lire/écrire des caractères
--with Ada.Numerics.Elementary_Functions; use Ada.Numerics.Elementary_Functions; --pour utiliser les fonctions mathématiques (sqrt, sin, ...)

procedure Moyen3Nombres is --Début du programme que vous avez nommé "nom_à_mettre "

a,b,c : float ;
res, moyenne : float ;
begin
```

```
–   Saisie des 3 nombres (a,b,c)
put ("Entrer a : ") ;-- ajout par rapport à l'AP, l'utilisateur doit savoir ce que le programme
                        attend. Le texte entre guillemets est écrit tel quel à l'écran
get(a) ; --lire(a)
put ("Entrer b : ") ;-- ajout par rapport à l'AP
get(b) ; --lire(b)
 put ("Entrer c : ") ;-- ajout par rapport à l'AP
get(c) ; --lire(c)
–   Calcul moyenne
res := a+b+c ;
moyenne := res / 3.0 ; --il faut avoir une division réelle
–   affichage du résultat
put ("La moyenne est : ") ;-- ajout par rapport à l'AP. Le texte entre " " est écrit à l'écran
put (moyenne) ;-- la valeur stockée dans moyenne est affichée à l'écran
end Moyen3Nombres ;
```

2. Résolution de ax+b=0

```
--Programme Résolution de ax+b=0

--with Ada.integer_Text_Io ; use Ada.integer_Text_Io; --pour lire/écrire des entiers
with Ada.float_Text_Io ; use Ada.float_Text_Io; --pour lire/écrire des flottants
with Text_Io; use Text_Io; --pour lire/écrire des caractères
--with Ada.Numerics.Elementary_Functions; use Ada.Numerics.Elementary_Functions; --pour
utiliser les fonctions mathématiques (sqrt, sin, ...)

procedure ResEqDegre1 is --Début du programme que vous avez nommé "nom_à_mettre "

a,b,c : float ;
x : float ;
begin
–   Saisie des 2 nombres (a,b)
put ("Entrer a : ") ;-- ajout par rapport à l'AP, l'utilisateur doit savoir ce que le programme
                        attend. Le texte entre guillemets est écrit tel quel à l'écran
get(a) ; --lire(a)
put ("Entrer b : ") ;-- ajout par rapport à l'AP
get(b) ; --lire(b)
–   Résolution de ax+b=0 et affichage du résultat
if (a /= 0) then
   x := -b/a ;
   put ("La solution est : ") ;-- ajout par rapport à l'AP. Le texte entre " " est écrit à l'écran
   put (x) ;-- la valeur stockée dans x est affichée à l'écran
else
   if (b = 0) then
       put ("Il y a une infinité de solutions ") ;
   else
      put ("Il n'y a pas de solution ") ;
```

```
   end if ;
end if;
end ResEqDegre1 ;
```

3. Saisie d'un nombre positif

```
--Programme Saisie d'un nombre positif entier (précision supplémentaire par rapport au cours)

with Ada.integer_Text_Io ; use Ada.integer_Text_Io; --pour lire/écrire des entiers
--with Ada.float_Text_Io ; use Ada.float_Text_Io; --pour lire/écrire des flottants
with Text_Io; use Text_Io; --pour lire/écrire des caractères
--with Ada.Numerics.Elementary_Functions; use Ada.Numerics.Elementary_Functions; --pour
utiliser les fonctions mathématiques (sqrt, sin, ...)

procedure SaisieEntierPos is --Début du programme que vous avez nommé "nom_à_mettre "

n : integer ;

begin
–       Saisie du nombre n
put ("Entrer un nombre positif : ") ;-- ajout par rapport à l'AP, l'utilisateur doit savoir ce que le
                        programme attend. Le texte entre guillemets est écrit tel quel à l'écran
get(n) ; --lire(n)
–       n doit être positif
while (n < 0) loop
    put ("Attention le nombre que vous avez saisi n'est pas positif. Veuillez entrer un nombre
          positif : ") ; -- ajout par rapport à l'AP
    get(n) ; --lire(n)
end loop ;
–       Affichage de n
put ("Le nombre que vous avez saisi est :  ") ; -- ajout par rapport à l'AP
put (n) ; – afficher(n)
end ResEqDegre1 ;
```

4. Calcul de la somme des 50 premiers entiers

```
-Programme Somme des 50 premiers entiers

--with Ada.integer_Text_Io ; use Ada.integer_Text_Io; --pour lire/écrire des entiers
with Ada.float_Text_Io ; use Ada.float_Text_Io; --pour lire/écrire des flottants
with Text_Io; use Text_Io; --pour lire/ecrire des caracteres
--with Ada.Numerics.Elementary_Functions; use Ada.Numerics.Elementary_Functions; --pour
utiliser les fonctions mathématiques (sqrt, sin, ...)

procedure Somme50Entiers is --Début du programme que vous avez nommé "nom_à_mettre "

somme : integer ; – on l'utilise pour les calculs et le résultat final
```

```
begin
–       Calcul de la somme
somme := 0 ; – on affecte 0 à la variable somme
for  i in 1 .. 50 loop  – en ADA l'indice de boucle, ici « i »,  n'a pas à être déclaré
   somme := somme + i ;
end loop ;

put ("La somme des 50 premiers nombres est égale à : ") ;-- ajout par rapport à l'AP. Le texte
                                                          entre " " est écrit à l'écran
put (somme) ; – afficher(somme)
end Somme50Entiers ;
```

4 Opérateurs (de base) possibles en ADA

Les opérateurs sont donnés ci-après par ordre décroissant de priorité, au sein d'une même classe il n'existe pas de priorité :

Classes d'opérateurs	Opérateurs	Significations	Exemples	
			Expressions	Valeurs
prioritaires	** abs not	Exponentiation (exposant entier) valeur absolue fonction logique NON	2.0**3 abs(-4.2) not(x=2)	8.0 4.2 x≠2
multiplicatifs	* / rem mod	multiplication division reste de la division euclidienne modulo (même résultat que *rem* si les 2 opérandes sont de même signe)	4.5 * 2.0 5.0 / 2.0 5 / 2 5 rem 2 5 mod 2	9.0 2.5 2 1 1
additifs unaires	+ -	identité opposé	+2.0 -5	2.0 -5
additifs binaires	+ -	addition soustraction	3.0+5.4 2-3	8.4 -1
Comparatifs *!!! ne peuvent se faire qu'entre expressions de même type !!!*	= /= <= < >= >	égalité différence infériorité ou égalité infériorité supériorité ou égalité supériorité	4*2=3+5 4*2/=3+5 5/2<=2 5/2<2 5.5>=11.0/2.0 5.5>11.0/2.0	true false true false true false
Logiques *!!! même priorité !!!*	and or xor	Fonction logique ET Fonction logique OU Fonction logique OU exclusif	(5/2<=2) and (5.5>11.0/2.0) (5/2<=2) or (5.5>11.0/2.0) (5/2<=2) xor (5.5>11.0/2.0)	false true true