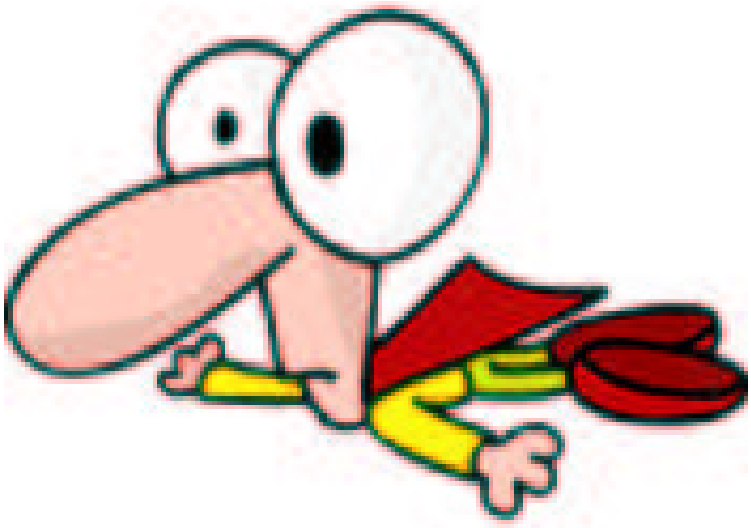
INSA TOULOUSE *Ministère de l'Enseignement Supérieur et de la Recherche*

SURVOL



DE LA SYNTAXE DU LANGAGE ADA

Pascal ACCO Patrick ESQUIROL Colette MERCE Gilles MOTET

Table des matières

Remarques :

- le but de ce document est de décrire, sous une forme simplifiée, les éléments syntaxiques les plus utilisés dans le cours de 1ère année. On n'y trouvera donc pas une description complète de la syntaxe du langage Ada.
- pour chaque construction syntaxique, on fournit une forme générale (éventuellement simplifiée) suivie de un ou plusieurs exemples.
- les mots clés du langage apparaissent en caractères gras.
- les éléments entre crochets [ ] sont facultatifs.

I\ FORME D'UN PROGRAMME ADA

```
with .....;
procedure   Nom_Du_Programme  is
   -- partie déclarative
begin
   -- corps de la procédure
end   Nom_Du_Programme;
```

Remarque : toute instruction se termine par **;**

II\ DÉCLARATIONS DE CONSTANTES ET DE VARIABLES

1- Déclaration de constante

```
Identificateur_De_Constante  :  constant Identificateur_De_Type  :=  valeur ;
Pi: constant Float        := 3.141516;
Nombre_D_Elements: constant Integer := 10;
Blanc: constant Character     := ' ';
```

2- Déclaration de variable

```
Identificateur_De_Variable  :  Identificateur_De_Type  ( :=  valeur_initiale ) ;
Risque_D_Avalanche: Integer;                -- pas d'initialisation
Epaisseur_Du_Manteau, Poids_De_Neige : Float := 0.0;
Alerte_D_Avalanche: Boolean  := False;
```

III\ LES INSTRUCTIONS

1- Affectation

```
Identificateur_De_Variable  :=  expression ;
Epaisseur_Du_Manteau := 3.55;
Poids_De_Neige := Epaisseur_Du_Manteau*0.3;
Risque_D_Avalanche := integer(Epaisseur_Du_Manteau * Poids_De_Neige) + 1;
Alerte_D_Avalanche :=  Risque_D_Avalanche > 10;
```

2- Instructions de choix

a) Alternative

```
if Condition_1 then
   Action_1;
else
    Action_2;
end  if;
```

Si *Condition_1* a la valeur **True**, on exécute *Action_1* et on quitte la structure **if.**

Si ce n'est pas le cas, on exécute *Action_2*.

```
subytpe Les_Notes is natural
                        range 0..20;
Note: Les_Notes;
.....
if  Note < 10  then
    Redoublement;
else  -- Note sous la moyenne

    Passage;
end  if;
```

```
Lettre_A_Tester: character;
Nombre_De_Majuscules: natural:= 0;
subtype Les_Majuscules is integer range 'A'..'Z';
...
if  Lettre_A_Tester  in  Les_Majuscules  then
      -- C'est une majuscule
      Nombre_De_Majuscules :=
                    Nombre_De_Majuscules + 1;
end if;
```

b) Choix multiple

```
if  Condition_1     then
    Action_1;
elsif  Condition_2  then
    Action_2;
      . . .
else
    Action_4;
end if;
```

Si *Condition_1* est **True**, on exécute *Action_1* et on quitte la structure **if**.

Sinon, on teste *Condition_2*; si elle est **True**, on exécute *Action_2* et on quitte la structure **if**.

Sinon, on teste *Condition_3*; etc.

Si aucune des conditions n'est **True**, on exécute *Action_4*.

Remarques :

- le nombre de branches **elsif** est quelconque.

```
if    Note < 10   then
    Redoublement;      --  ici 0 ≤ Note < 10
elsif  Note < 12  then
    Mention_Passable;  --  ici  10 ≤ Note < 12
elsif  Note <  14  then
    Mention_AB    ;       --  ici 12 ≤ Note < 14
elsif  Note < 16  then
    Mention_B;         --  ici 14 ≤ Note < 16
else
    Mention_TB;        --  ici 16 ≤ Note < 20
end if;
```

c) Instruction case

```
case   Expression  is
   when   liste_de_valeurs_1  =>   Action_1;
   when   liste_de_valeurs_2  =>  Action_2;
   when   liste_de_valeurs_3   =>  Action_3;
   when  others                  => Autre_Action;
end case;
```

La valeur de *Expression* est calculée. Si cette valeur appartient à *liste_de_valeurs_1*, *Action_1* est exécutée ; si elle appartient à *liste_de_valeurs_2*, *Action_2* est exécutée ; etc. Si elle n'appartient à aucune liste de valeurs, *Autre_Action* est exécutée.

Remarques :

- *Expression* est de type discret;
- une liste de valeurs peut être une valeur simple, un intervalle de valeurs, une énumération (cf. exemple ci-après). Les **listes de valeurs doivent être disjointes** : sans aucun élément en commun.
- la clause **when others** peut être omise lorsque les *listes_de_valeurs* recouvrent toutes les valeurs possibles de *Expression*.
- lorsqu'elle apparaît, la clause **when others** est forcément la dernière.

```
subtype Ensemble_Des_Heures is natural range 0..23;
Heure: Ensemble_Des_Heures;
.....
case  Heure  is
   when 7                 =>  Se_Lever;        -- valeur simple
   when 9..12             =>  Travailler;      -- intervalle de valeurs
   when 8 | 13 | 17 | 20     =>  Manger ;        -- énumération de valeurs
   when   0..6 | 22..23   =>  Dormir;          -- énumération d'intervalles
   when others            =>  Se_Reposer;
end case;
```

3- Instruction nulle

```
...
else
   null;
end if;
```

Remarque : Utilisée le plus souvent dans les instructions **if** ou les instructions **case,** elle permet de signifier de façon explicite l'absence d'action dans une des deux branches d'un **if** ou dans un des cas énumérés dans un **case**.

4- Instructions de boucle

a) Boucles for

```
for   Indice_De_Boucle  in ( reverse ) Intervalle  loop
   Action ;
end  loop  ;
```

Indice_De_Boucle prend successivement toutes les valeurs de *Intervalle* (de la plus petite à la plus grande avec **in** ; de la plus grande à la plus petite avec **in reverse**); *Action* est répétée pour chacune de ces valeurs.

Remarques :

- *Indice_De_Boucle* est une variable locale à la boucle **for**; elle ne doit pas être déclarée et n'existe pas hors de la boucle **for;**
- *Indice_De_Boucle* ne peut pas être modifiée à l'intérieur de la boucle.

```
subtype Les_Lettres_Majuscules is character range `A' ..'Z';
.....
for Lettre_Majuscule in Les_Lettres_Majuscules  loop
   Ada.Text_Io.Put(Lettre_Majuscule);
    Ada.Text_Io.Put(' ');
end loop;
```

```
Nombre_De_Codes: constant integer:= 10;
subtype Les_Indices_De_Codes is integer range 1..Nombre_De_Codes;
type Des_Ensembles_De_Codes is array (Les_Indices_De_Codes) of Integer;
Codes_De_Portes: Des_Ensembles_De_Codes;
.....
for Numero_De_Porte in reverse 1 .. (Nombre_De_Codes-1) loop
        Codes_De_Portes(Numero_De_Porte):=
                                        Codes_De_Porte(Numero_De_Porte+1);
end loop;
```

b) Boucle *«tant que»* :

```
while  Condition   loop
     Action ;
end  loop ;
-- ici Condition= False
```

Tant que *Condition* a la valeur **True**, *Action* est répétée. On quitte la boucle lorsque *Condition* a la valeur **False**.

Remarque : *Action* peut ne jamais être exécutée si, dès le départ, *Condition* est **False**.

```
Nombre_De_Joueurs: Integer;
.....
while  (Nombre_De_ Joueurs /= 0)
         and (Nombre_De_ Joueurs  rem 2 = 0)  loop
   Nombre_De_Joueurs:= Nombre_De_Joueurs  / 2;
end  loop;
-- Ici Nombre_De_Joueurs est soit nul, soit impair.
```

c) Boucle «*répéter jusqu'à»* :

```
loop
   Action;
exit  when  Condition ;
end  loop  ;
-- ici Condition = True
```

Action est répétée jusqu'à ce que *Condition* ait la valeur **True**. On quitte la boucle lorsque *Condition* est à **True**.

Il ne doit y voir qu'une seule condition **exit when** dans la boucle.

Remarque : *Action* est toujours exécutée au moins une fois.

```
Nombre_De_Joueurs: integer;
.....
loop
   Nombre_De_Joueurs:= Nombre_De_Joueurs* 2;
exit when (Nombre_De_Joueurs >= 50 );
end  loop  ;
-- ici Nombre_De_Joueurs  est supérieur à 50
```

IV\ CARACTÉRISTIQUES DES TYPES SIMPLES PRÉDÉFINIS

Un type est caractérisé par :
- un ensemble de valeurs
- l'ensemble des opérations qui peuvent être effectuées sur ces valeurs.

De plus, en Ada, des attributs sont associés à chaque type : ils renseignent sur les caractéristiques du type (plus petite et plus grande valeurs représentables, etc.) ou complètent l'ensemble des opérations (valeur suivante dans le type, etc.). Se reporter au §6 pour la signification des attributs.

1- Le type INTEGER

ENSEMBLE DE VALEURS	un sous-ensemble des entiers relatifs
FORME DES CONSTANTES	`18  -123  +4235`
Opérations sur Integer à résultat de type Integer : Opérations sur Integer à résultat de type Boolean :	`+  -  *  /  ** abs mod rem` `= /= < <= > >= in (not in)`
PRINCIPAUX ATTRIBUTS	`First  Last`

Remarque : **abs** fournit la valeur absolue d'un nombre, **mod** le modulo de 2 nombres et **rem** le reste de la division entière de 2 nombres.

Exemples d'expressions :

Expression	Type du résultat	Valeur du résultat
`7 / 2`	`Integer`	`3`
`7 rem 2`	`Integer`	`1`
`(-7) mod 2`	`Integer`	`1`
`18 <= 10`	`Boolean`	`False`
`5 in -10..10`	`Boolean`	`True`

2- Le type FLOAT

ENSEMBLE DE VALEURS	approximations de valeurs réelles
FORME DES CONSTANTES	`4.827 -0.56  +12.235 E-4  -5.12 E+5`
Opérations sur Float à résultat de type Float : Opérations sur Float à résultat de type Boolean :	`-  *  /  **  abs` `= /= < <= > >=`
PRINCIPAUX ATTRIBUTS	`First  Last  Digits  Epsilon`

Pour les PC des salles de TP, on a :
Float'**first** = -3.40282 E+38 Float'**last** = 3.40282 E+38
Float'**digits** = 6 Float'**epsilon** = 9.53674 E-07

3- Le type BOOLEAN

ENSEMBLE DE VALEURS	{True, False}
FORME DES CONSTANTES	True False
Opérations sur Booléan à résultat Boolean :	not and or xor (and then)(or else) = /=

4- Le type CHARACTER

ENSEMBLE DE VALEURS	les 128 caractères de la norme ASCII
FORME DES CONSTANTES	'A' 'v' '?' '3' ' '
Opérations sur Character à résultat de type boolean :	= /= < <= > >= in (not in)
PRINCIPAUX ATTRIBUTS	Pred Succ Pos Val

Remarques :

- le jeu des caractères est ordonné. Ainsi par exemple, l'expression 'A'<'G' a la valeur **True**.

Integer, Boolean et Character sont des types **discrets** (chaque valeur a une valeur précédente et une valeur suivante).

5- Conversions de types

En Ada, il n'est pas possible de mélanger des types numériques différents dans une même expression. Il faut faire une **conversion explicite**. Les conversions doivent être seulement utilisées dans des cas très particuliers; vérifiez au préalable si son usage n'est pas dû à une mauvaise programmation qu'il faut alors corriger.

a) Conversion d'une expression I de type Integer, en une valeur de type Float : Float(I)

Ainsi : Float(12) = 12.0

```
Somme_Des_Valeurs , Moyenne: Float;
Nombre_De_Valeurs: Integer;
...
Moyenne := Somme_Des_Valeurs / Float(Nombre_De_Valeurs);
```

b) Conversion d'une expression F de type Float en une valeur de type Integer : Integer(F)

Ainsi :

```
Integer(2.0 + 3.3)  =  5
Integer( 17.9 ) = 18
```

6- Les attributs usuels

a) Sur les types discrets

T'**First**	plus petite valeur de T	
T'**Last**	plus grande valeur de T	
T'**Pred** (X)	valeur précédant X dans T	(ex : Character'**Pred**('B') = 'A')
T'**Succ**(X)	valeur suivant X dans T	(ex : Character'**Succ**('B') = 'C')
T'**Pos**(X)	rang de X dans le T	(ex : Character'**Pos**('A') = 64)
T'**Val**(R)	valeur de rang R dans T	(ex : Character'**Val**(64) = 'A')
T'**Image**(X)	chaîne de caractères représentant la valeur de X	
T'**Value**(S)	valeur (de type discret T) correspondant à la chaîne de caractères S	

b) Sur les types réels flottants

R'**Digits**	nombre de chiffres décimaux de la mantisse de valeurs du type réel R
R'**Epsilon**	plus petite valeur du type réel R telle que 1.0 + R'**Epsilon** > 1.0

V\ PROCÉDURES D'ENTRÉES & SORTIES

Les Entrées-Sorties (E/S en abrégé) (opérations de lecture ou d'écriture) se font par appel à des sous-programmes. Ces sous-programmes sont regroupés dans des paquetages :

- paquetage **Ada.Text_Io** pour les E/S portant sur des Character ou des String (chaînes de caractères),
- paquetage **Ada.Integer_Text_Io** pour les E/S portant sur des Integer,
- paquetage **Ada.Float_Text_Io** pour des E/S portant sur des Float.

```
with Ada.Text_Io, Ada.Integer_Text_Io;

procedure Demander_Un_Entier_A_L_Utilisateur is
   Valeur_Saisie: integer;
begin
   Ada.Text_Io.Put( " Entrez une valeur entière: ?" );
   Ada.Integer_Text_Io.Get( Valeur_Saisie );  -- lecture d'un entier au clavier

   Ada.Text_Io.Put( " La valeur lue est : " );    -- écriture d'une chaîne de caractères
   Ada.Integer_Text_Io.Put(Valeur_Saisie );  -- écriture d'un Integer à l'écran
   Ada.Text_Io.New_Line;
end  Demander_Un_Entier_A_L_Utilisateur;
```

1- Lecture au clavier d'une valeur de type Integer, Float ou Character

- **Get**(*Variable*) ;

  Lit une valeur tapée au clavier et validée par un retour chariot (↵) et l'affecte à *Variable.*

  Remarque : le type de la valeur fournie doit correspondre au type de la variable.

Le tableau ci-dessous fournit des exemples de valeurs valides et non valides tapées par l'utilisateur en fonction du type de *Variable*. Une exception nommée **Constraint_Error** est levée si la valeur a un type non valide.

Type de *Variable*	Exemples de valeurs valides	Exemples de valeurs non valides
Integer	12 0 -5	0.65 1.2 E+4 A
Float	12.67 -4.5E2 0.0	0 12 -3 A
Character	A x ? 4	

- **Ada.Text_Io.Skip_Line;** --provoque un changement de ligne en lecture (vide le buffer d'entrée)

Remarque : Il est parfois nécessaire de faire suivre une opération de lecture par un **Skip_Line** ; c'est le cas par exemple lorsqu'un premier caractère d'un texte frappé est lu et qu'on ne désire pas conserver les autres, par exemple pour lire le 1er caractère de chaque ligne. **Skip_Line** est fournie uniquement par Ada.Text_Io.

2- Lecture au clavier d'une chaîne de caractères

- **Get(*Chaîne*);**
  où *Chaîne* est une variable de type String(1..n)
  n représente une valeur entière quelconque.
  Lit au clavier une chaîne de n caractères **exactement** et l'affecte à ***Chaîne.*** **Sinon une erreur de type constraint_error est déclenchée.**

```
subtype Des_Noms_De_Comptes is string(1..8);
Login_Utilisateur: Des_Noms_De_Comptes;
.....
   Ada.Text_Io.Get(Login_Utilisateur);  -- il faut taper exactement 8 caractères!!!
                                        --  Par exemple   jhendrix
```

- **Get_Line(*Chaîne, Longueur*);**
  Chaîne représente une variable de type String(1..n) avec *n* un entier donné (lors de la déclaration) : n = Chaine'length .
  Longueur représente une variable de type **Integer**.
  Lit au clavier une chaîne d'**au maximum** n caractères. La lecture s'arrête
  - soit lorsque n caractères ont été lus
  - soit lorsque une fin de ligne (**return**) est trouvée.

  En fin de lecture, *Longueur* indique combien de caractères ont **effectivement** été lus.

```
subtype Des_Noms_De_Comptes is string(1..8);
Login_Utilisateur: Des_Noms_De_Comptes := (others => ' ');
Nombre_De_Caracteres_Lus: integer;
.....
Ada.Text_Io.Get_Line(Login_Utilisateur, Nombre_De_Caracteres_Lus);
```

si on tape : jhendrix↵ on aura : Login_Utilisateur ="jhendrix" et Nombre_De_Caracteres_Lus =8

si on tape : jpp↵ on aura : Login_Utilisateur ="jpp" et Nombre_De_Caracteres_Lus =3

si on tape : jjp ↵ on aura : Login_Utilisateur ="jpp " et Nombre_De_Caracteres_Lus =6

3- Écriture à l'écran

- **Put (*Variable*) ;** --affiche à l'écran la valeur contenue dans *Variable* (format standard)
- **Ada.Text_Io.New_Line ;** fait passer le curseur (pavé ou caractère souligné clignotant) à la ligne suivante. New_Line est fourni uniquement par Ada.Text_Io.

a) Formats de sortie

- si *Variable* est un Integer
  Put(*Variable*, *v*); --la valeur de *Variable* s'affiche sur *v* chiffres(ou digits) à l'écran (*v* est une valeur entière)

- si *Variable* est un Float
  Put (*Variable* , *m* , *f* , *e*); --la valeur s'affiche avec : *m* chiffres pour la partie entière de *Variable, f* chiffres pour sa partie fractionnaire et *e* chiffres pour l'exposant de *Variable.*

Exemples :

```
X: integer   := -12;                          impressions obtenues
Y: float     := 0.12345 E+2;          (le caractère ◦ représente le caractère blanc).
...                                                          Bord de
Ada.Integer_Text_Io.Put(X);              ◦◦◦◦◦◦◦◦-12          l'écran
Ada.Integer_Text_Io.Put(X,5);            ◦◦-12
Ada.Integer_Text_Io.Put(X, 7);           ◦◦◦◦-12
Ada.Float_Text_Io.Put(Y);                ◦1.23450E+01
Ada.Float_Text_Io.Put(Y,2,3,2);          ◦1.235E+1
Ada.Float_Text_Io.Put(Y,4,2,0);          ◦◦12.35
```

4- Lecture/Écriture dans un fichier

Comme on peut lire une variable sur un clavier ou l'écrire sur l'écran, on peut lire ou écrire dans un fichier des données. Les Entrées-Sorties se font par appel à des sous-programmes. Ces sous-programmes sont regroupés dans les mêmes paquetages :

- paquetage **Ada.Text_Io** pour les E/S portant sur des Character ou des String (chaînes de caractères),
- paquetage **Ada.Integer_Text_Io** pour les E/S portant sur des Integer,
- paquetage **Ada.Float_Text_Io** pour des E/S portant sur des Float.

a) Déclaration d'un fichier

Les fichiers textes « manipulés » en Ada sont des objets du type limité privé File_Type exporté du paquetage Ada.Text_Io. Il est indispensable d'évoquer ce paquetage avec with évidemment.

```
Identificateur_du_fichier :  Ada.Text_Io.File_Type ;
Fichier_Des_Eleves : Ada.Text_Io.File_Type ;
```

b) Création, ouverture et fermeture d'un fichier

Il faut ouvrir un fichier avant de l'utiliser. Pour cela on dispose dans le paquetage Ada.Text_Io des deux procédures :

(i) Create

La procédure Create ouvre un fichier en vue de sa création. Si le fichier existe déjà l'ouverture avec Create le supprimera !

```
Create(Identificateur_Du_Fichier, Mode_d_acces, Nom_du_Fichier, Forme_Du_Fichier);
Create(Fichier_Des_Eleves, Ada.Text_Io.Out_File, "fichier_des_eleves.txt","");
```

- *Identificateur_Du_Fichier :* est la variable du type File_Type identifiant le fichier.
- *Mode_d_acces :* Ada.Text_Io.Out_File lorsque l'on veut écrire dans le fichier, et Ada.Text_Io.In_File lorsque l'on veut lire dans le fichier. Dans le cas de *create*, on veut forcément écrire dans le fichier que l'on vient de créer, ce sera donc ici toujours Out_File.
- *Nom_du_Fichier* : est un String contenant le nom du fichier que l'on veut créer.
- *Forme_Du_Fichier :* N'est utilisé que pour des cas complexes! Ce sera toujours un string vide dans notre cas "".

En TP : On peut voir ce que le fichier contient avec la commande « more fichier_des_eleves.txt » dans une xterm ou avec emacs par exemple.

(ii) Open

La procédure **Open** ouvre un fichier qui existe déjà sur le disque soit en lecture (**In_File**), soit en écriture (**Out_File**). Dans le cas de l'écriture, le contenu initial du fichier est perdu. Si le fichier n'existe pas, l'ouverture avec **Open** lève une exception. Le fichier ne doit pas être déjà ouvert !

```
Open(Identificateur_Du_Fichier, Mode_d_acces, Nom_du_Fichier, Forme_Du_Fichier );
Open(Fichier_Des_Eleves, Ada.Text_Io.In_File, "fichier_des_eleves.txt","");
```

Les paramètres sont les mêmes que pour **Create**.

(iii) Close

La procédure **Close** ferme le fichier, le fichier doit être ouvert!

```
Close( Identificateur_Du_Fichier );
Close(Fichier_Des_Eleves);
```

c) Lecture et écriture dans le fichier

(i) Pour les **character**, **integer** et **float**

On retrouve, toujours dans le paquetage Ada.Text_Io, Integer_Text_Io, Float_Text_Io nos « vieilles connaissances » **Get, Get_Line, Put, Put_Line et New_Line** déjà vues. Mais cette fois il y a un paramètre supplémentaire permettant d'identifier le fichier.

```
Get(Identificateur_Du_Fichier, Variable du type character integer ou float);
Ada.Text_Io.Get(Fichier_des_Eleves, Initiale_Du_Nom); -- Initiale_Du_Nom est  character
Ada.Integer_Text_Io.Get(Fichier_des_Eleves, Age); -- Age est un integer
Ada.Float_Text_Io.Get(Fichier_des_Eleves, Moyenne); -- Moyenne est un float
```

Dans le cas d'un *Get* il faut que le fichier soit ouvert et que son mode d'accès (spécifié dans Open) soit en lecture (In_File).

```
Put( Identificateur_Du_Fichier , Variable du type character inetger ou float );
```

Dans le cas d'un **Put** il faut que le fichier soit ouvert et que son mode d'accès (spécifié dans **Open** ou **Create**) soit en écriture (**Out_File**).

```
New_Line( Identificateur_Du_Fichier );
Ada.Text_Io.New_Line(Fichier_Des_Eleves);
```

New_Line n'est utilisable que sur un fichier en mode écriture **Out_File**.

Si l'on veut passer à la ligne suivante d'un fichier en mode lecture (**In_File**) il faut utiliser **Skip_Line**.

```
Skip_Line( Identificateur_Du_Fichier );
Ada.Text_Io.Skip_Line(Fichier_Des_Eleves);
```

(ii) Pour les chaînes de caractères

```
Get_Line( Identificateur_Du_Fichier , Variable du type string, Nombre de caractères lus);
Ada.Text_Io.Get_Line(Fichier_Des_Eleves, Nom, Longueur); -- Nom est un string
```

```
Put_Line( Identificateur_Du_Fichier , Variable du type string );
Ada.Text_Io.Put_Line(Fichier_Des_Eleves, "Janis Joplin");
```

d) Fin de ligne et fin de page lors de la lecture d'un fichier

Lorsque l'on effectue une lecture d'un fichier (mode **In_File**) il est nécessaire de détecter une fin de ligne ou une fin de fichier. Ada.Text_Io fournit pour cela deux fonctions :

(i) End_Of_Line

Retourne **True** si l'on se trouve à la fin d'une ligne du fichier.

```
Booléen := End_Of_Line( Identificateur_Du_Fichier );
if Ada.Text_Io.End_Of_Line(Fichier_Des_Eleves) then
   Ada.Text_Io.Skip_Line(Fichier_Des_Eleves);
      ...
```

(ii) End_Of_File

```
Booléen := End_Of_File( Identificateur_Du_Fichier );
While not  Ada.Text_Io.End_Of_File(Fichier_Des_Eleves) loop
   Ada.Text_Io.Get_Line(Fichier_Des_Eleves,Une_Ligne);
        ....
```

e) Exemple

```
-- Ce programme ouvre le fichier "lisez_moi.txt"
-- Affiche son contenu à l'écran
-- Et le copie dans le fichier "lisez_moi_copie.txt"
--   en y ajoutant à la fin "Ce fichier a ete copie par Ada"

with Ada.Text_Io;

procedure Lire_Afficher_Copier_Lisez_Moi is

  Fichier_A_Lire: Ada.Text_Io.File_Type;
  Fichier_A_Ecrire: Ada.Text_Io.File_Type;
  Caractere: character;

begin

  Ada.Text_Io.Open(Fichier_A_Lire,
                   Ada.Text_Io.In_File,
                   "lisez_moi.txt","");
       -- on ouvre le fichier de nom lisez_moi.txt
       -- dans le programme on l'appelle Fichier_A_Lire

  Ada.Text_Io.Create(Fichier_A_Ecrire,
                     Ada.Text_Io.Out_File,
                     "lisez_moi_copie.txt","");
       -- on créé le fichier de nom lisez_moi_copie.txt
       -- dans le programme on l'appelle Fichier_A_Ecrire
```

```
  while not Ada.Text_Io.End_Of_File(Fichier_A_Lire) loop

    if Ada.Text_Io.End_Of_Line(Fichier_A_Lire) then
      --on est en fin de ligne
      Ada.Text_Io.Skip_Line(Fichier_A_Lire);  -- on passe ainsi à la ligne
      Ada.Text_Io.New_Line; -- Nouvelle ligne à l'écran
      Ada.Text_Io.New_Line(Fichier_A_Ecrire); -- Nouvelle ligne
                                    --  dans le fichier produit
    else
      --on est dans une ligne
      Ada.Text_Io.Get(Fichier_A_Lire,Caractere);
      Ada.Text_Io.Put(Caractere);   -- on affiche le caractère à l'ecran
      Ada.Text_Io.Put(Fichier_A_Ecrire,Caractere);-- on l'écrit dans
                                    --  le fichier produit
    end if;
  end loop;-- tant qu'on est pas à la fin du fichier

  Ada.Text_Io.Close(Fichier_A_Lire);--on en a plus besoin

  Ada.Text_Io.New_Line(Fichier_A_Ecrire);
  Ada.Text_Io.Put_Line(Fichier_A_Ecrire,"Ce fichier a ete copie par Ada");
  Ada.Text_Io.Close(Fichier_A_Ecrire);

end Lire_Afficher_Copier_Lisez_Moi;
```

VI\ DÉCLARATIONS DE TYPES ET SOUS-TYPES

ATTENTION : **ne pas confondre la déclaration d'un type (ou sous-type) et la déclaration d'une variable** (les instructions du corps d'un programme portent sur des variables et non sur des types) **!**

Déclarations d'un type et d'un sous-type :

type *Identificateur_Du_Type* **is ...***à compléter selon la nature du type* **;**
subtype *Identificateur_Du_Sous-Type* **is** *Identificateur_Du_Type* **range** *Vmin..Vmax***;**

Vmin et Vmax sont des valeurs du type *Identificateur_Du_Type.* Le sous-type *Identificateur_Du_Sous-Type* définit l'ensemble des valeurs de l'intervalle [*Vmin*, *Vmax*] et hérite des opérations sur le type *Identificateur_Du_Type*.

Déclaration d'une variable :

*Identificateur_De_Variable***:** *Identificateur_De_Type* **(:=** *valeur_initiale* **) ;**

1- Déclaration d'un type énuméré

type *Identificateur_Du_Type* **is (** *énumération des valeurs du type* **);**

```
type   Jours_De_La_Semaine is
(lundi,mardi,mercredi,jeudi,vendredi,samedi,dimanche);
type  Couleurs_De_Base     is (jaune,vert,bleu,rouge,noir);
```

2- Déclaration d'un sous-type

```
subtype  Identificateur  is Identificateur_Type_De_Base range Intervalle_De_Valeurs;
subtype Ensemble_Des_Ages        is  integer                  range  0..120;
subtype Les_Lettres_Majuscules   is  character                range 'A'..'Z';
subtype Les_Jours_De_Travail     is  Jours_De_La_Semaine      range lundi..vendredi;
subtype Les_Numeros_D_Eleves     is  integer                  range 1..24;
```

3- Déclaration d'un type Tableau Contraint

Un type tableau est dit contraint lorsque le nombre de composantes du tableau est fixé dans sa déclaration.

a) type tableau à 1 dimension :

```
type Identificateur_Type is array(sous-type de l'indice) of Type_Des_Composantes;
type Des_Notes_D_Une_Classe is array(Les_Numeros_D_Eleves) of Float;
type  Des_Valeurs_De_Lettres is array(Les_Lettres_Majuscules) of Integer;
type Des_Heures_De_Travail_Hebdomadaires  is array(Les_Jours_De_Travail) of Float;
```

b) type tableau à 2 dimensions (matrices) :

```
type  Identificateur_Du_Type  is  array(type ou sous-type du 1er indice ,
                                        type ou sous-type du 2ème indice )
                              of  Type_Des_Composantes;
type Des_Matrices is array (Numeros_De_Ligne, Numeros_De_Colonnes) of Float;
```

4- Déclaration d'un type Tableau Non Contraint

Un type tableau est dit **non contraint** lorsque **le nombre de composantes** du tableau **n'est pas fixé dans la déclaration** du type. Quand on déclare une variable de type tableau non contraint, il est obligatoire de préciser le domaine de variation de l'indice.

- Déclaration d'un type tableau non contraint

```
type  Identificateur_Du_Type   is array (Identificateur_De_Type  range <>)
                               of Type_Des Composantes;
type Notes_Des_Eleves_Presents is array (Integer range <>) of Float ;
```

- déclaration d'une variable :

```
Notes_Des_Asinsa: Notes_Des_Eleves_Presents( 1..24 ) ; -- Il y a 24 étudiants Asinsa
                                                       --il n'y aura pas plus de 24 étudiants présents
```

5- Déclaration d'un type Record

```
type  Identificateur_Du_Type  is  record
   Champ_1 : Type_Du_Champ_1;
   Champ_2 : Type_Du_Champ_2;
        ...
end record;
type Des_Employes      is  record
    Nom:  Les_Noms_De_Comptes;
    Age: Ensemble_Des_Ages;
    Charge_Horaire: Des_Heures_De_Travail_Hebdomadaires;
end record;
```

Remarque : pour accéder à un champ d'une variable de type record, on utilise la notation pointée.

Exemple :

```
Responsable_Qualite_Logicielle : Des_Employes ;
...
Responsable_Qualite_Logicielle.Nom:= "Motet";
Responsable_Qualite_Logicielle.Age:= 20;
Responsable_Qualite_Logicielle.Age.Charge_Horaire:= 15000.0;
```

VII\ SOUS-PROGRAMMES

1- Procédures

a) Déclaration d'une procédure :

(i) Spécification :

procedure *Nom_de_la_Procedure* **(** *liste des paramètres formels* **);**

Les *paramètres formels* matérialisent les informations échangées entre le programme d'appel et la procédure. On précise la direction (le mode) de l'échange (**in**, **out** ou **in out**) et le type de l'information échangée. On donne également un nom à chaque paramètre formel.

La *liste des paramètres formels* a donc la forme suivante :

(*paramètre***:** *mode type* **;** *paramètre***:** *mode type* **;** .....**;** *paramètre* **:** *mode type***)**

Si plusieurs paramètres ont le même type et le même mode, ils peuvent être regroupés dans la liste :

(*paramètre_1 , paramètre_2* **:** *mode type***;** ..... **)**

```
procedure Permuter( X, Y: in out integer );
procedure Trier( Vecteur_Initial: in  Vecteur;  Vecteur_Trie: out Vecteur );
```

(ii) corps de la procédure :

```
procedure Nom_De_La_Procedure ( liste des paramètres formels) is
   -- partie déclarative
begin
    -- corps de la procédure
end Nom_De_La_Procedure;

procedure Permuter( X, Y: in out Integer ) is
   Auxiliaire: integer;
begin
   Auxiliaire := X;
   X:= Y;
   Y:= Auxiliaire;
end Permuter;
```

Remarque : si dans un programme, la spécification et le corps d'une procédure se suivent, on peut omettre la spécification (cf. exemple §3).

b) Appel d'une procédure

Nom_De_La_Procedure **(** *liste des paramètres d'appel* **);**

Les paramètres d'appel sont des variables (ou des constantes) du programme d'appel; ce sont sur ces grandeurs que s'applique le sous-programme

Exemple :

```
with Ada.Text_Io, Ada.Integer_Text_Io;

procedure Tester_La_Permutation is

   Valeur_A_Gauche, Valeur_A_Droite: integer;

   procedure Permuter( X, Y: in out integer ) is
      -- permute les valeurs contenues dans deux variables entières
      -- quelconques symbolisées par les paramètres X et Y
      Auxiliaire: integer;
   begin
      Auxiliaire:= X;
      X:= Y;
      Y:= Auxiliaire;
   end Permuter;

begin  -- début du test

   -- saisie des valeurs
   Ada.Text_Io.Put( " Entrer 2 valeurs entières :");
   Ada.Text_Io.New_Line;
   Ada.Text_Io.Put(" Première valeur =? ") ;
   Ada.Integer_Text_Io.Get(Valeur_A_Gauche);
   Ada.Text_Io.Put("Deuxième valeur = ? ") ;
   Ada.Integer_Text_Io.Get(Valeur_A_Droite);

   -- appel de la procédure pour permutter les deux valeurs
   Permuter(Valeur_A_Gauche, Valeur_A_Droite);

   -- affichage des valeurs
   Ada.Text_Io.Put(" Après permutation la première valeur doit se trouver à droite ");
   Ada.Text_Io.New_Line;
   Ada.Integer_Text_Io.Put(Valeur_A_Gauche, 3);
   Ada.Text_Io.Put("               ");
   Ada.Integer_Text_Io.Put(Valeur_A_Droite, 3);

end  Tester_La_Permutation ;
```

A l'exécution, on aura (les valeurs soulignées sont fournies par l'utilisateur du programme):

Entrer 2 valeurs entières :
Première Valeur = ? 10
Deuxième Valeur = ? 15
Après permutation la première valeur doit se trouver à droite
15 10

2- Fonctions

a) Déclaration d'une fonction

(i) Spécification

```
function Nom_De_La_Fonction ( liste des paramètres formels ) return
Type_Du_Résultat ;
```

Remarque : Les *paramètres formels* d'une fonction sont tous de mode **in**.

```
function Valeur_Minimale( Du_Vecteur: in Vecteur) return integer;
function Trier( Le_Vecteur: in Vecteur)  return Vecteur;
```

(ii) corps d'une fonction

```
function Nom_De_La_Fonction( liste des paramètres formels ) return
Type_Du_Résultat   is
   -- partie déclarative
   .....
begin
   -- corps de la fonction
   .............
  return  Expression;
end  Nom_De_La_Fonction ;
```

Remarque : le corps de la fonction doit obligatoirement contenir une seule instruction **return**. Placée à la fin.

Expression représente la valeur retournée par la fonction; elle est du type *Type_Du_Résultat*.

b) Appel d'une fonction

L'appel d'une fonction retourne un résultat. Ce résultat doit être intégré dans une expression, être imprimé, etc.

```
Variable := Nom_De_La_Fonction ( liste des paramètres d'appel );
```

ou

```
put( Nom_De_La_Fonction ( liste des paramètres d'appel ));
```

Exemple :

```
with Ada.Text_Io, Ada.Integer_Text_Io;

procedure Tester_La_Fonction_Valeur_Minimale is

   subtype Les_Elements_Du_Vecteur is integer range 1..4;
   type Vecteur is array(Les_Elements_Du_Vecteur) of integer;
   Mon_Vecteur: Vecteur;

   function Valeur_Minimale ( Du_Vecteur: in Vecteur ) return Integer is
      -- retourne la valeur min contenue dans un Vecteur quelconque
      Le_Min: Integer:= Du_Vecteur(1);
   begin
      for Case_A_Tester in 2..Du_Vecteur'Last loop
         if Du_Vecteur(Case_A_Tester) < Le_Min then
            Le_Min:= Du_Vecteur(Case_A_Tester);
         else
            null;
         end if;
      end loop;
      return Le_Min;
   end Valeur_Minimale;

   procedure Saisir(Un_Vecteur : out Vecteur ) is
      --permet de saisir au clavier les valeurs d'un vecteur
   begin
      for Case_A_Saisir in Un_Vecteur'Range loop
         Ada.Text_Io.Put(" Valeur? ");
         Ada.Integer_Text_Io.Get (Un_Vecteur(Case_A_Saisir));
      end loop;
   end Saisir ;

begin  -- début du test de Valeur_Minimale

   Saisir( Mon_Vecteur );-- saisie des valeurs de Mon_Vecteur

   -- affichage de la valeur minimum contenue dans Mon_Vecteur
   Ada.Text_Io.New_Line;
   Ada.Text_Io.Put(" La valeur minimale est:  ");
   Ada.Integer_Text_Io.Put( Valeur_Minimale(Mon_Vecteur), 2);

end Exemple;
```

Exécution :

```
Valeur ? 15
Valeur ? 22
Valeur ?  4
Valeur ? 18
La valeur minimale est : 4
```

Alphabetical Index