

# Analyse et programmation langage ADA



Informatique 1<sup>ère</sup> année



# Tableau de caractères, Type String

- Présentation du type prédéfinie String
- Attributs Image et Value
- Exemple de chaînes de caractères
- Manipulation de chaînes de caractères

## Tableau de caractères, Type String

Le type `STRING` est un type pré-défini du paquetage `STANDARD`.

Il est défini comme un tableau non contraint de caractères, indexé d'entiers strictement positifs.

```
type String is array (Positive range <> ) of Character;
```

Les chaînes de caractères se manipulent donc exactement comme des tableaux, ainsi :

```
ligne : String(1.. 80);
```

Quand on ne connaît pas la longueur d'une chaîne de caractères, on est obligé de fixer lors de la déclaration une taille maximum et de gérer ensuite explicitement la vraie longueur.

# Présentation du type prédéfinie String

```
procedure chaine_caractere is
```

```
  subtype Mon_String is String(1..20);
```

```
  Mon_Bonjour   : constant String := "Bonjour" ;
```

```
  Au_Revoir     : constant String := ('a', 'u', ' ', 'r', 'e', 'v', 'o', 'i', 'r') ; -- agreg. Pos.
```

```
  Chaine_Vide   : constant String := "";
```

```
  Taille_Message : constant       := 20 ;
```

```
  Message1      : String(1..Taille_Message) ;
```

```
  Message2      : Mon_String ;
```

```
begin
```

```
  Message1(9) := 'e';           -- Acces aux éléments de la chaine
```

```
  Message1(Mon_Bonjour'Range) := Mon_Bonjour; --tranche de tableau
```

```
  Message2(1.. Mon_Bonjour 'Length + Au_Revoir'Length) :=
```

```
    Message1(Mon_Bonjour'Range) & Au_Revoir ; -- concaténation
```

```
end chaine_caractere ;
```



# Attributs Image et Value

- **Attributs Image et Value** (valeur scalaire => image de caractères)

Integer'Image(12)	donne la chaîne " 12"
Integer'Image(-12)	donne la chaîne "-12"
Float'Image(12.34)	donne la chaîne " 1.23400E+01"
Character'Image('a')	donne la chaîne "'a'"
T_Mois_D_L_Anee'Image(Mars)	donne la chaîne "MARS"
Integer'Value(" 12")	donne le nombre entier 12
Integer'Value("-12")	donne le nombre entier -12
Float'Value(" 1.234E+01")	donne le nombre réel 1.234E+01
Character'Value("'a'")	donne le caractère 'a'
Natural'Value("-12")	lèvera la contrainte <i>Constraint_Error</i>

- **Entrées–sorties telles qu'elles sont définies dans Ada.Text\_IO**

```
procedure Put( Item : in String);  
procedure Put_Line(Item : in String);  
procedure Get(Item : out String);  
procedure Get_Line(Item : out String;  
                    Last  : out Natural);
```



# Exemple de chaines de caractères

```
with Ada.Text_IO; use Ada.Text_IO;

procedure Exemple4 is
  Bienvenue : constant String := "Bienvenue dans l'exemple!";
  Taille    : constant := 8;
  Chaine    : String(1..Taille); -- Chaine de 8 caracteres au maximum
  Nombre_Car_Lus : Natural ;
begin -- Exemple4
  Put_Line(Bienvenue);
  Put("Lecture d'une chaine avec Get(tapez" &
      Integer'Image(Taille) & " caractères aux moins) : ");           --1
  Get(Chaine);                                                         --2
  Skip_Line;                                                            --3
  Put_Line("Voici la chaine que vous avez tapee : " & Chaine);
  Put("Lecture d'une chaine avec Get_Line(terminez par une ");
  Put("fin de ligne avant" & Integer'Image(Taille)& " caracteres) : ");
  Get_Line(Chaine, Nombre_Car_Lus);                                     --4
  Put( "Voici la chaine que vous avez tapee : ");
  Skip_Line;                                                            --5
  Put_Line(Chaine(1..Nombre_Car_Lus));
end Exemple4;
```

# Exemple de chaînes de caractères

Écran de votre  
machine

Bienvenue dans l'exemple!

Lecture d'une chaîne avec Get(tapez 8 caractères aux moins) : **chevalier**

Voici la chaîne que vous avez tapée : chevalie

Lecture d'une chaîne avec Get\_Line(terminez par une fin de ligne avant 8 caractères) : **chevalier**

Voici la chaîne que vous avez tapée : chevalie

# Lecture d'une chaîne de caractères

```
with Ada.Text_io;      use Ada.Text_io;
procedure Chaine is
  Ligne : String(1..80);      -- on suppose une longueur maximum de la chaîne de 80
                              -- caractères

  lg : Natural range 0..80;

begin
  Put( "Tapez votre nom : ");
  Get_Line(ligne, lg);      --lg contient le nombre de caractères effectivement saisis
  Put( "ligne = ");
  Put(Ligne);
  New_Line;
  declare
    Nom : String(1..lg) := Ligne(1..lg); --le type String est contraint
  begin
    Put( "nom = ");
    Put(Nom);
  end;
end Chaine;
```



# Lecture d'une chaîne de caractères

Écran de votre machine

## Résultat

```
Tapez votre nom : Daniel
Ligne = DanielB j½?x~8 xÿÿÿÿtÿB c' @?Đ| xä| x ???✗???
ÿB È@?i✗A?0,A?i✗A?0,A?X✗A?~ÿB v @?
Nom = Daniel
```

- ➔ Une variable déclarée n'est pas vide
- ➔ La variable *Ligne* contient le nom Daniel + le reste des caractères non initialisées
- ➔ Mais la procédure *Put* affiche tout le contenu du tableau de caractères *Ligne*

## Remarque :

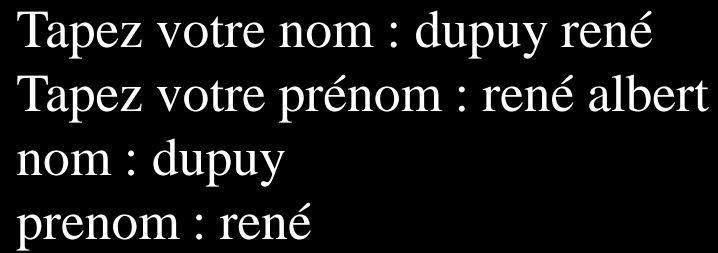
si une chaîne vide est saisie (en appuyant seulement sur la touche entrée),

- le nom contiendra une chaîne vide ( "" )
- la valeur de lg sera 0.

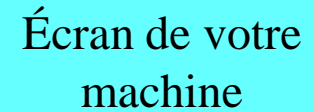
# Passage à la ligne

Après une saisie, la procédure `skip_line` permet de sauter tout ce que l'utilisateur a tapé sur le clavier jusqu'à la fin de la ligne.

```
put( "Tapez votre nom : ");  
get(nom);  
skip_line;  
put( "Tapez votre prénom : ");  
get(prenom);  
put("nom : ");  
put_line(nom);  
put("prenom: ");  
put_line(prenom);
```



```
Tapez votre nom : dupuy rené  
Tapez votre prénom : rené albert  
nom : dupuy  
prenom : rené
```



Écran de votre  
machine

# Affichage d'une chaîne de caractères II

```

with Ada.Text_io;      use Ada.Text_io;
procedure Chaine is
  Nom : String(1..40) := others=>' '; -- on initialise la chaine avec des blancs
  lg : Natural range 0..40;
begin
  Put( "Tapez votre nom : ");
  Get_Line(Nom, lg);      --lg contient le nombre de caractères effectivement saisis
  Put(" Nom = ");
  Put(Nom);
end Chaine;
  
```

Écran de votre machine

Tapez votre nom : dupuy rené  
Nom = dupuy rené

# Manipulation de chaînes de caractères 1

**Ada.Strings.Fixed** ( paquetage sur les traitements de chaînes de caractères)  
( opérations disponibles dans ce paquetage)

- **Index**, fonction qui recherche un motif dans une chaîne et qui retourne la position du premier caractère du motif dans la chaîne
- **Count**, fonction qui compte le nombre d'occurrences d'un motif dans la chaîne
- **Insert**, fonction ou procédure qui insert un motif dans une chaîne à partir d'une certaine position
- **Overwrite**, fonction ou procédure qui substitue une partie d'une chaîne par un motif à partir d'une certaine position
- **Delete**, fonction ou procédure qui supprime une partie d'une chaîne

# Manipulation de chaînes de caractères 2

Dans le fichier **a-strfix.ads** contenant les spécifications des fonctions et procédures incluses dans le paquetage **Ada.Strings.Fixed**, on trouve les deux prototypes suivants :

```
procedure Insert (Source : in out String;  
                 Before : in Positive;  
                 New_Item : in String;  
                 Drop : in Truncation := Error);
```

```
function Insert (Source : in String;  
                Before : in Positive;  
                New_Item : in String) return String;
```

Dans la procédure, la source est en mode **in out**, donc s'il y a débordement de la taille de source donnée en paramètre d'entrée, une exception est levée.

Dans la fonction, on construit une nouvelle chaîne à l'intérieur de la fonction et on l'envoie à l'extérieur par la clause **Return**,

 il n'y aura pas de débordement de la taille, donc pas de levée d'exception



# Manipulation de chaînes de caractères 3

```
with Ada.Text_IO;
with Ada.Strings.Fixed;
with Ada.Strings;
use Ada.Text_IO;
use Ada.Strings.Fixed;
use Ada.Strings;
```

```
procedure P3 is
  S : String:="linux";
  Ajout : String:="ni";
begin Put_Line(S);
  Insert(S,S'Last,Ajout,Right);
  Put_Line(S);
end P3;
```

```
Linux
linun
```

```
with Ada.Text_IO;
with Ada.Strings.Fixed;
with Ada.Strings;
use Ada.Text_IO;
use Ada.Strings.Fixed;
use Ada.Strings;
procedure P2 is
  S : String:="linux";
  Ajout : String:="ni";
begin
  Put_Line(S);
  Insert(S,S'Last,Ajout,Left);
  Put_Line(S);
end P2;
```

```
Linux
nunix
```

```
with Ada.Text_IO;
with Ada.Strings.Fixed;
use Ada.Text_IO;
use Ada.Strings.Fixed;
procedure P1 is
  S : String:="linux";
  Ajout : String:="ni";
begin
  Put_Line(S);
  Insert(S,S'Last,Ajout);
  PutLine(S);
end P1;
```

```
Linux
raised ADA.STRINGS.LENGTH_ERROR : a-strfix.adb:344
```



# Manipulation de chaînes de caractères 4

```
with Ada.Text_IO;
with Ada.Strings.Fixed;
use Ada.Text_IO;
use Ada.Strings.Fixed;
procedure P6 is
  S : String:="linux";
  Ajout : String:="ni";
  Resu : String(1..S'Length +
               Ajout'Length+1);
begin
  Put_Line(S);
  Resu(1..Resu'Length-1) :=
    Insert(S,S'Last,Ajout);
  Put_Line(Resu);
  Resu:=Insert(S,S'Last,Ajout);
  Put_Line(Resu);
end P6;
```

```
Linux
linunix@
raised CONSTRAINT_ERROR : p6.adb:12
```

```
with Ada.Text_IO;
with Ada.Strings.Fixed;
use Ada.Text_IO;
use Ada.Strings.Fixed;
procedure P5 is
  S : String:="linux";
  Ajout : String:="ni";
  Resu : String(1..S'Length +
               Ajout'Length);
begin
  Put_Line(S);
  Resu:=Insert(S,S'Last,Ajout);
  Put_Line(Resu);
end P5;
```

```
Linux
linunix
```

```
with Ada.Text_IO;
with Ada.Strings.Fixed;
use Ada.Text_IO;
use Ada.Strings.Fixed;
procedure P4 is
  S : String:="linux";
  Ajout : String:="ni";
begin
  Put_Line(S);
  S:=Insert(S,S'Last,Ajout);
  Put_Line(S);
end P4;
```

```
Linux
raised CONSTRAINT_ERROR : p4.adb:28 length check failed
```



# Manipulation de chaînes de caractères 5

```
with Ada.Text_IO;           use Ada.Text_IO;
with Ada.Integer_Text_IO;  use Ada.Integer_Text_IO;
with Ada.Strings.Fixed;    use Ada.Strings.Fixed;
```

procedure Exemple is

```
Texte : String := "Texte extra dans ce contexte"; -- une chaîne de 28 caractères
```

```
Ext   : constant String := "ext";           -- 2 motifs
```

```
Tex   : constant String := "tex";
```

begin -- Exemple

```
Put_Line(Integer'Image(Texte'length)) ;
```

-- Affiche la longueur 28

```
Put_Line(Integer'Image(Index(Texte, Ext))) ;
```

-- Affiche l'indexe du 1er motif "ext" 2

```
Put_Line(Integer'Image(Index(Texte, Tex))) ;
```

-- Affiche l'indexe du 1er motif "tex" 24

```
Put_Line(Integer'Image(Index(Texte, "Tex"))) ;
```

-- Affiche l'indexe du 1er motif "Tex" 1

```
Put_Line(Integer'Image(Ada.Strings.Fixed.Count(Texte, "ext"))) ; -- Affiche le nombre
```

-- d'occurrences "ext" 3

```
Put_Line(Texte);
```

```
Put_Line(Insert(Texte,12,"bleu ciel"));
```

```
Texte := "Texte extra dans ce contexte";
```

```
Put_Line(Overwrite (Texte,12,"-ordinaire"));
```

```
Texte := "Texte extra dans ce contexte";
```

```
Put_Line>Delete(Texte,12,Texte'Last));
```

end Exemple;



Écran de votre  
machine

28

2

24

1

3

Texte extra dans ce contexte

Texte extrableu ciel dans ce contexte

Texte extra-ordinaireontexte

Texte extra