Programmation

Le langage Ada – 16 Les caractères des chaînes

17/12/2008

Posté par La rédaction | Signature : Yves Bailly
Tags : GLMF
0 Commentaire | Ajouter un commentaire

Retrouvez cet article dans : Linux Magazine 90

Tout au long des articles précédents, nous n'avons cessé d'utiliser des chaînes de caractères. Celles-ci n'étaient alors connues que comme de simples tableaux. Mais Ada propose bien plus que cela : c'est un ensemble très complet d'outils, qui est disponible et qui n'a que peu de choses à envier à d'autres langages.

Modérons dès maintenant cette introduction : un reproche fréquemment fait à Ada est l'absence notable d'expressions régulières. Il existe toutefois des paquetages comblant ce manque, notamment les paquetages GNAT.Regexp et GNAT.Regpat fournis avec le compilateur GNAT. Ceux-ci ne faisant pas partie intégrante du langage, nous ne les aborderons pas ici.

Les caractères

Le type caractère de base, correspondant au type char du langage C, est Character. Il y a toutefois une différence fondamentale : contrairement à char, le type Character n'est pas un type numérique : c'est un type énuméré, fournissant exactement 256 valeurs, définies très précisément comme étant les 256 caractères du jeu de caractères Latin-1 (la ligne 0 du standard ISO/IEC 10646:2003). N'utilisez donc pas le type Character pour représenter des petites valeurs entières tenant dans un unique octet. Pour cela, utilisez plutôt :

```
type Petit_Entier is new Integer range -128..127;
for Petit_Entier'Size use 8;
```

L'attribut 'Size, qui donne habituellement la taille d'un type en bits (et non en octets), est ici utilisé pour spécifier la taille voulue. Naturellement, vous êtes libre de définir l'intervalle que vous voulez, du moment que cela tient dans 8bits – dans le cas contraire, le compilateur vous signalera une erreur. Par ailleurs, la taille demandée n'est pas obligatoirement un multiple de 8, ce qui peut s'avérer pratique dans des environnements à la quantité de mémoire très limitée.

Pour mémoire, cette écriture signifie " Ada Reference Manual ", annexe A, section 3. Vous trouverez le standard Ada 2005 sur http://www.adaic.com/standards/ada05.html.

Mais revenons à nos caractères. Étant un type énuméré, Character possède tous les attributs de ces types, notamment 'Pos (pour connaître la position d'une valeur) et 'Val (pour obtenir la valeur à une position donnée). Par exemple, Character'Pos('A') retourne l'entier 65, tandis que Character'Val(65) retourne le caractère 'A'. Ada étant un langage à vocation universelle, ce seul type de caractères ne peut suffire. Aussi, en existe-il deux autres, Wide_Character et Wide_Wide_Character. Le premier représente les caractères encodés sur 16bits (UCS-2), le second les caractères sur 32bits (UCS-4). Encore une fois, ces deux types sont considérés comme étant des types énumérés, pas des types numériques : inutile d'essayer de les utiliser dans des calculs. Chaque " grand " type contient les caractères du type " plus petit ". Ainsi, les caractères de Character sont inclus dans ceux de Wide_Character, dont les caractères sont inclus dans ceux de Wide_Wide_Character :

/img-articles/lm/90/art-7/fig-1.jpg

Attention, cela ne signifie pas que les valeurs d'un type sont contenues dans l'ensemble des valeurs d'un autre type : les inclusions sont sémantiques, mais chaque type possède son propre ensemble de valeurs.
La bibliothèque Ada standard propose trois paquetages liés directement aux caractères :

- Ada.Characters.Conversions contient des sous-programmes pour convertir entre les différents types de caractères, ou pour savoir si un caractère d'un " grand " type est également un caractère d'un type plus " petit " ;
- Ada.Characters.Handling offre des fonctions de classifications des caractères de type Character, pour distinguer entre majuscules et minuscules, ponctuation, chiffres, etc. ;
- Ada.Characters.Latin_1, enfin, contient des constantes représentant chacun des caractères du jeu Latin-1.

Consultez le standard Ada pour plus d'informations (ARM-A.31).

Les chaînes de base

En Ada, une chaîne de caractères est un tableau non contraint de caractères. Avec tout ce que cela implique comme contraintes et opérations disponibles sur les tableaux. Pour être précis, les trois types chaîne sont définis ainsi :

```
subtype Positive is Integer range 1..Integer'Last;
type String      is array(Positive range <>) of Character;
type Wide_String      is array(Positive range <>) of Wide_Character;
type Wide_Wide_String is array(Positive range <>) of Wide_Wide_Character;
```

Il existe donc un type de chaîne pour chacun des types de caractères, chaque type étant un tableau dont l'indice est de type Positive, lui-même une restriction du type entier général Integer aux entiers strictement positifs. Par convention (identique au langage Pascal), l'indice du premier caractère d'une chaîne est 1.
Le fait que les chaînes soient des tableaux non contraints imposent des contraintes (sic) parfois gênantes. Par exemple, vous ne pouvez pas déclarer une variable ainsi :

```
chaine: String;
```

Car il est interdit de déclarer une variable d'un type non contraint sans lui donner une restriction ou une valeur initiale. Par exemple :

```
chaine_1: String(1..7);
chaine_2: String := "Monde";
```

Il n'est alors possible d'affecter à ces variables que des chaînes d'exactement la même longueur, ainsi :

```
chaine_1 := "Bonjour";   -- ok
chaine_1 := "Coucou";    -- erreur
```

La deuxième affectation provoquera une erreur de compilation, car la chaîne "Coucou" ne contient que six caractères, alors que la variable chaine_1 a été déclarée comme devant en contenir sept. Cela vaut également pour chaine_2, qui est implicitement une chaîne de cinq caractères. Une façon de contourner le problème est de réaliser l'affectation sur une partie de la chaîne :

```
chaine_1(1..5) := "Hello";
chaine_1(6..7) := (others => ' ');
```

Mais ce n'est guère satisfaisant. Heureusement, la bibliothèque Ada standard nous fournit différents moyens de nous simplifier la vie, comme nous le verrons plus loin.
Dernier mot, la concaténation de chaînes se fait comme pour les tableaux, à l'aide de l'opérateur &.

Manipulations sur les chaînes

Le paquetage Ada.Strings.Fixed contient tout un ensemble de sous-programmes pour manipuler les chaînes de caractères String (les paquetages Ada.Strings.Wide_Fixed et Ada.Strings.Wide_Wide_Fixed offrent les mêmes fonctionnalités sur les autres types de chaînes). Son paquetage parent, Ada.Strings, contient divers types et constantes permettant de spécifier, par exemple, l'alignement à gauche ou à droite, si une recherche doit s'effectuer vers la gauche ou la droite, etc. (voir ARM-A.4 et les sections suivantes).
On trouve ainsi dans Ada.Strings.Fixed, entre autres, une procédure Move que l'on pourrait utiliser ainsi :

```
Move("Coucou", chaine_1);
```

Ceci réalise l'affectation de la chaîne "Coucou" dans chaine_1, ce qui était une erreur en utilisant directement l'opérateur :=. Si la chaîne source est plus courte que la chaîne cible (ce qui est le cas ici), par défaut les caractères sont " calés " à gauche (paramètre optionnel Justify, de type Ada.Strings.Alignment, valeur par défaut Left) et l'espace restant comblé par des espaces (paramètre optionnel Pad de type Character). Dans le cas contraire, où la chaîne cible est plus courte que la chaîne source, un cinquième paramètre optionnel (Drop, de type Ada.Strings.Truncation) précise comment la chaîne source doit être tronquée, voire s'il faut lever l'exception Length_Error (cas par défaut).

Ce paquetage fournit également de nombreux sous-programmes de recherche d'une sous-chaîne dans une chaîne (nommés Index, tous des fonctions) : supprimer une sous-chaîne (Delete), remplacer des caractères (Overwrite), en insérer d'autres (Insert), extraire une partie d'une chaîne (Trim, Tail et Head), etc. Mais un opérateur particulièrement utile, "*", permet de construire une chaîne comme étant la répétition d'un même caractère ou d'une même chaîne, par exemple :

```
with Ada.Strings.Fixed;
use Ada.Strings.Fixed;
...
s1: String := 5*':';
s2: String := 2*"---8<--";
```

À l'issu de ces déclarations, s1 contient ":::::" et s2 contient "---8<-----8<--". On retrouve là une fonctionnalité existant entre autres en Python.

Chaînes dynamiques

Le paquetage Ada.Strings.Unbounded définit le type privé Unbounded_String permettant de manipuler des chaînes de Character dynamiques, c'est-à-dire dont la taille peut varier au cours de l'exécution. Pour les chaînes basées sur les autres types de caractères, il existe les paquetages Ada.Strings.Wide_Unbound et Ada.Strings.Wide_Wide_Unbound, définissant respectivement les types Wide_Unbounded_String et Wide_Wide_Unbounded_String, aux fonctionnalités similaires.
La création d'une chaîne dynamique se fait tout simplement à partir d'une chaîne classique, à l'aide de la fonction To_Unbounded_String :

```
us1: Unbounded_String := To_Unbounded_String("Bonjour");
```

La conversion inverse est obtenue par To_String.
L'intérêt de ces chaînes est que l'on peut leur affecter n'importe quelle autre chaîne, quelle qu'en soit la longueur :

```
us1 := To_Unbounded_String("Une autre chaîne");
```

ou encore, ce qui est plus efficace, car cela évite la création d'un objet temporaire :

```
Set_Unbounded_String(us1, "Une autre chaîne");
```

Le paquetage fournit en plus les mêmes facilités que Ada.Strings.Fixed pour ce qui est des manipulations, transformations, recherches, et ainsi de suite.

Conclusion

Selon la formule consacrée, nous n'avons fait ici que survoler toutes les possibilités des chaînes de caractères en Ada. En particulier, ont été négligés les outils permettant de convertir un ensemble de caractères en un autre (paquetage Ada.Strings.Maps), ou les chaînes à taille variable, mais dont une longueur maximale est fixée, plus efficaces que les chaînes Unbounded_String (paquetage Ada.Strings.Bounded). Plongez-vous dans le standard Ada pour avoir plus de détails.

Retrouvez cet article dans : Linux Magazine 90