TP5 : paquetages et généricité

1 Objectifs

L'objectif de cette séance est de se familiariser avec le concept de paquetages et de généricité en Ada, en écrivant un paquetage générique de tri.

2 Paquetages et généricité en Ada

2.1 Paquetages

Un paquetage Ada se compose de deux parties :
— la *spécification* contient la déclaration des types, variables, fonctions et procédures rendues disponibles par l'utilisation de ce paquetage ;
— l'*implémentation* contient les types et variables locales au paquetages, et l'implémentation des fonctions et procédures déclarées dans la spécification.

Pour un paquetage `Exemple`, la spécification est dans un fichier nommé `exemple.ads` (*Ada Specification*), de structure suivante :

```
-- déclarations with/use

package Exemple is

  -- Déclarations de types, variables, fonctions et procédures

end Exemple;
```

L'implémentation d'un paquetage se trouve dans le fichier `exemple.adb` (*Ada Body*) :

```
-- déclarations with/use

package body Exemple is

  -- Types et variables locales au paquetage, implémentation des
  -- fonctions et procédures déclarées dans la spécification

end Exemple;
```

Rappel : Il ne faut pas confondre «implémentation d'un paquetage» et «programme principal» : les deux sont placés dans des fichiers suffixés par `.adb`, mais la structure d'un programme principal diffère :

```
-- déclarations with/use

procedure Exemple is

  -- Types, variables, fonctions et procédures locales

begin

  -- programme principal

end Exemple;
```

2.2 Généricité

La *généricité* permet, lors de la réalisation d'un paquetage, d'abstraire certains éléments de ce paquetage : certains types ou fonctions, par exemple. Ces types ou fonctions abstraits deviennent alors des paramètres (dits *génériques*) de ce paquetage.

Les paramètres génériques d'un paquetage Ada sont placés en tête de la spécification (donc dans le `.ads`), dans une section délimitée par le mot-clé **`generic`** :

```
-- déclarations with/use

generic

  -- Paramètres génériques

package Exemple is

  -- Déclarations de types, variables, fonctions et procédures,
  -- utilisant les paramètres génériques

end Exemple;
```

Les paramètres génériques peuvent être :
— des variables, exemple : `N : integer`
— des types :
  — type indéterminé : **`type`** `T` **`is private`**
  — type entiers (`integer`, `natural`, etc.) : **`type`** `T` **`is range`** `<>`
  — etc., cf la documentation de référence : `http://www.adahome.com/rm95/rm9x-12-05.html`
— des procédures ou fonctions (attention au mot-clé **`with`**) : **`with procedure`** `P(X : in integer; Y : out T);`

Pour utiliser un paquetage générique il faut tout d'abord l'instancier, en fournissant des paramètres effectifs aux paramètres formaux génériques :

```
with Exemple;

package MonExemple is new Exemple(...);
```

`MonExemple` devient alors un paquetage (non générique), la déclaration ci-dessus étant équivalente à **`with`** `MonExemple`.

Exemple : lors de la séance de TP n° 2, il était demandé d'utiliser un paquetage de génération de nombres pseudo-aléatoires. La déclaration fournie est une instanciation du paquetage `Ada.Numerics.Discrete_Random`, avec comme paramètre le type `Intervalle` :

```
package NombreAleatoire is new Ada.Numerics.Discrete_Random (Intervalle);
```

Remarque : les *paquetages génériques* Ada correspondent aux *foncteurs* OCaml.

3 Exercices

Exercice 1 : réalisation d'un paquetage générique

Définissez un paquetage de tri générique (vous pouvez vous inspirer des programmes fournis pour les séances précédentes). Commencez par bien identifier les paramètres du paquetage.

Exercice 2 : instanciation

Réalisez un programme principal, instanciant correctement votre paquetage générique, afin de :

1. Trier un tableau d'entiers.
2. Trier un tableau de nombres complexes.
   Vous pourrez définir votre propre type complexe en utilisant un record contenant la partie réelle et la partie imaginaire du nombre complexe :

```
type Complexe is record
    R : Float;
    I : Float;
end record;
```

Pour pouvoir utiliser le programme, il faut bien sûr prévoir les procédures permettant la lecture/écriture de nombres complexes.

On peut ensuite redéfinir l'opérateur $<$ comme suit :

```
function "<"(C1, C2: Complexe) return Boolean;
```

— Instanciez votre paquetage pour trier les nombres complexes par leur partie réelle.
— Instanciez votre paquetage pour trier les nombres complexes par leur module [1].

Chaque programme principal réalisé doit être testé sur un jeu de tests pertinent.

1. Rappel : le module d'un nombre complexe $z = a + b.i$ est le nombre réel positif $|z| = \sqrt{a^2 + b^2}$