
Chapitre 8

Suggestions

OBJECTIFS

- ✍ Donner un résumé des bonnes pratiques de programmation en Ada.

DÉCLARATION ET TYPAGE

✎ Déclaration de constantes.

Utiliser des constantes déclarées plutôt que des littéraux numériques afin d'améliorer la portabilité et l'efficacité.

```
PI : constant := 3.1416;
```

✎ Sous-type.

Si la même contrainte s'applique à différents objets, considérez de déclarer un sous-type. Ainsi, un changement de contrainte n'affecte qu'une seule déclaration.

DÉCLARATION ET TYPAGE (Suite)

✎ Allocation dynamique.

- Première approche

```
begin
```

```
-- lire les bornes
```

```
declare
```

```
    DYNAMIQUE : array ( 1 .. borne) of COMPOSANT;
```

```
begin
```

```
-- utilisation du tableau
```

```
end;
```

```
end;
```

DÉCLARATION ET TYPAGE (Suite)

- Allocation dynamique.
 - Deuxième approche (plus générale)

declare

```
type ARRAY_TYPE is array (NATURAL range <> ) of COMPOSANT;  
type REFERENCE is access ARRAY_TYPE;  
DYNAMIQUE : REFERENCE;
```

begin

```
-- lire les bornes  
DYNAMIQUE := new ARRAY_TYPE(1 .. borne);  
-- utilisation DYNAMIQUE.all (indice )
```

end;

Le pointeur peut référer à des tableaux de différentes dimensions. Attention à l'implémentation

Ingénierie du Logiciel avec Ada - ? - École Polytechnique - Ada80-0-2002-01-17-12:12 chapppt

DÉCLARATION ET TYPAGE (Suite)

- Type articles (**record**) avec discriminant.
 - Si le discriminant est utilisé pour déclarer la dimension d'un tableau, e.g.

declare

```
type NON_CONTRAINED (SIZE : NATURAL := ...) is  
  record  
    DATA : STRING ( 1 .. SIZE);  
  end;  
ALPHA : NON_CONTRAINED;
```

begin

...

end;

Certains compilateurs réservent la dimension maximum. (Pour sous-types NATURAL, NATURAL'LAST)

Ingénierie du Logiciel avec Ada - ? - École Polytechnique - Ada80-0-2002-01-17-12:12 chapppt

DÉCLARATION ET TYPAGE (Suite)

- ✍ Type article (**record**) avec discriminant.
Déclaration d'un sous-type pour le discriminant avec une valeur raisonnable.

```
declare
  subtype RAISONNABLE is NATURAL range 1 .. 255;
  type NON_CONTRAIANT (SIZE : RAISONNABLE := ...) is
    record
      DATA : STRING ( 1 .. SIZE);
    end;
  ALPHA : NON_CONTRAIANT;
begin
  ...
end;
```

NOMS ET EXPRESSIONS

- ✍ Utilisation d'attribut
Pour améliorer l'entretien, et la réutilisation, utiliser au maximum les attributs.

Exemple typique:

```
for I in BETA'RANGE loop
  GO_FOR_IT(BETA(I));
end loop;
```

BETA'RANGE ou *BETA'FIRST .. BETA'LAST*

SOUS-PROGRAMMES

✂ Fonction vs procédure

Si une procédure n'a qu'un paramètre de sortie, considérez en faire une fonction. Les fonctions améliorent la lisibilité des programmes.

✂ Association des paramètres par noms

Choisir des noms courts et appropriés et utiliser l'association par nom pour améliorer la lisibilité.

✂ Surcharge d'opérateurs

La surcharge d'opérateurs pour des programmes mathématiques améliore la lisibilité des programmes.

PAQUETAGES

✂ Abstraction

Les paquetages sont utilisés (fréquemment pour déclarer des types de données. Le type **private** dans la spécification d'un paquetage permet de limiter les opérations sur les objets. Les constantes différées font aussi partie de tels paquetages.

✂ Exceptions dans les paquetages.

Les sous-programmes dans un paquetage doivent être construits de façon telle qu'ils ne propagent pas d'exception au module appelant, Les exceptions particulières doivent être déclarées dans la partie visible d'un paquetage.

RÈGLES DE VISIBILITÉ (Suite)

☞ Contrôler la visibilité des noms importés de paquetage.

Les trois techniques sont:

- Si la clause "**use**" est utilisée, il n'est pas nécessaire de préfixer les noms importés (paquetage.entité)
- S'il n'y a pas de clause "**use**", mais un ensemble de clauses "**renames**", alors les valeurs non-surnomées doivent être préfixées, tandis que les surnoms peuvent être utilisés directement.
- S'il n'y a pas de clause "**use**", ni de surnoms, alors les entités doivent être préfixées. En particulier, les opérateurs doivent être appelés avec la notation fonctionnelle.

RÈGLES DE VISIBILITÉ (Suite)

☞ Quelle technique choisir ?

Dépend d'un grand nombre de facteurs:

- Combien d'entités déclarées dans le paquetage sont requises par le programme et le lecteur est-il familier avec le contenu du paquetage.
- La clause "**renames**" a certains avantages sur la clause "**use**" étant donné qu'elle clarifie quel paquetage contient les déclarations originales.
- Il est suggéré d'utiliser la clause **use** pour les paquetages utilisés fréquemment et avec lesquels le programmeur est familier (TEXT_IO).

TÂCHES

- ⚡ Attention aux tâches appelant des sous-programmes déclarés dans un paquetage utilisant ses propres variables locales (paquetage). Déclarer une tâche locale pour permettre l'exclusion mutuelle des données.
- ⚡ Utilisation de modèle
Les programmes utilisant des tâches suivent souvent les mêmes modèles, e.g. moniteur, sémaphore, tampon, producteur / consommateur. Faites en des unités génériques.
- ⚡ Terminaison d'une tâche
L'instruction **select** avec une alternative **terminate** est recommandée pour terminer une tâche. Une clause **terminate** ne peut être utilisée dans une tâche déclarée dans une unité de bibliothèque.

STRUCTURE DE PROGRAMME -- UNITÉ DE COMPILATION

- ⚡ Sous-unités de compilation
Réduit le temps de compilation
 - ⚡ Minimiser les dépendances des compilations
L'importance d'une unité (**with**) doit être la plus profonde possible dans l'arbre des unités de compilation.
 - ⚡ La clause **with** doit accompagner le corps du paquetage et non les spécifications.
 - ⚡ De façon analogue, une clause **with** doit faire partie de la sous-unité de compilation et non de l'unité parent.
- Ne pas combiner dans la même unité de compilation, les spécifications et le corps de l'unité.

EXCEPTIONS

- Traitement local des exceptions.
Un bloc peut être utilisé pour gérer localement les exceptions. Par exemple, un bloc peut être utilisé à l'intérieur d'une boucle, si on ne désire pas quitter la boucle après le traitement de l'exception.
- Erreur d'entrée / sortie
Il est fréquent d'intégrer les énoncés d'entrée / sortie à l'intérieur d'un bloc contenant une partie traitement_d_exception.
- **Exception** dans les déclarations.
Il est préférable de définir ses propres exceptions plutôt que d'utiliser les exceptions prédéfinies. S'il est possible que l'initialisation d'une variable déclenche une exception, il est suggéré d'initialiser cette variable dans le corps du module (programme, sous-programme, etc...).

UNITÉS GÉNÉRIQUES

- Réutilisation de logiciel

Les unités génériques sont un élément important de Ada, elles doivent être construites de la façon la plus générale possible. Par exemple, si un type tableau est un paramètre formel d'une unité générique, son indice devrait aussi être un paramètre formel de type discret) pour plus de généralité).