PIM - Résumé

October 20, 2023

THEVENET Louis

1. Language algorithmique

1.1. Procédures et fonctions

Définition 1.1.1: Une procédure est une **abstraction de contrôle**, qui définit une **action** ou une **instruction**.

Spécification :
1. identificateur *(un verbe)*
2. Paramètres
3. Pré-condition *(assurée par le contrat)*
4. Post-condition *(assurée par le contrat)*
5. Tests/exemples
6. Exceptions

Exemple : Spécification d'une procédure

```
nom : nom de la procédure
sémantique: décrire ce que réalise la procédure
paramètres:
        F_Param_1 : Mode (In, In/Out, Out) Type; --Rôle du paramètre
        ...
        F_Param_n : Mode (In, In/Out, Out) Type; --Rôle du paramètre
pré-condition: Conditions sur les paramètres en entrée (in)
post-condition: Conditions sur les paramètres en sortie (out)
```

Définition 1.1.2: Une fonction est une **abstraction de données**, elle définit une **expression**, un **calcul**.
Elle n'a pas d'effet de bord et **retourne toujours un résultat**. La différence est que les flots de données sont limités à `in`.

Spécification :
1. Identificateur *(un nom commun)*
2. Paramètres
3. Type de retour
4. Pré-condition *(assurée par le contrat)*
5. Post-condition *(assurée par le contrat)*
6. Tests
7. Exceptions

Exemple : Spécification d'une fonction

```
Nom : nom de la fonction
Sémantique : sémantique de la fonction
Paramètres
        F_Param_1 : mode (in) Type -- rôle du paramètre
        ...
        F_Param_n : mode (in) Type -- rôle du paramètre
Type de retour : Type du résultat retourné
Pré-condition : Conditions sur les paramètres en entrée
Post-condition : Conditions sur le résultat retourné
```

2. Types de données

2.1. Algorithmique

Définition 2.1.1 : Enumération

C'est une liste de **valeurs possibles**, avec une relation d'ordre définie par l'ordre des éléments dans la définition.

Définition :

```
TYPE T_COULEUR EST ENUMERATION (BLANC, BLEU, ROUGE)
```

Ainsi on a $\text{BLANC} < \text{BLEU} < \text{ROUGE}$.

Définition 2.1.2 : Enregistrement

Produit cartésien de plusieurs domaines.

Définition :

```
TYPE Date EST ENREGISTREMENT
    Jour : entier {1 <= Jour ET Jour <= 31}
    Mois : entier {1 <= Mois ET Mois <= 12}
    Annee : entier
FIN ENREGISTREMENT
```

On peut ajouter des assertions comme ci-dessus.

Définition 2.1.3 : Tableau

Indexation de valeurs de même type.

Définition :

```
MAX_1 : CONSTANTE Entier <-- 4
MAX_2 : CONSTANTE Entier <-- 6

TYPE T_Matrice EST TABLEAU (1..MAX_1, 1..MAX_2) DE Entier

T: T_Matrice
T(2,3) <-- 2017
```

2.2. Ada

```
type T_Couleur is (BLANC, BLEU, ROUGE);

type T_Date is record
    Jour : Integer; --{1 <= Jour ET Jour <= 31}
    Mois : Integer;
    Annee : Integer;
end record;

type T_vecteur is array(1..10) of Integer;
```

3. Modules (Ada)

3.1. Spécification

```
-- Spécification d'un module Dates très simplifié.
package Dates is
    type T_Mois is (JANVIER, FEVRIER, MARS, AVRIL, MAI, JUIN, JUILLET, AOUT,
SEPTEMBRE, OCTOBRE, NOVEMBRE, DECEMBRE);
    type T_Date is record
        Jour : Integer;
        Mois : T_Mois;
        Annee : Integer;
    end record;

    -- spécifications omises
    procedure Initialiser (Date : out T_Date; Jour : in Integer; Mois : in
T_Mois; Annee : in integer )
    with
        Pre => Annee >= 0 and Jour >= 1 and Jour <= 31,
        Post => Le_Jour (Date) = Jour and Le_Mois (Date) = Mois and L_Annee
(Date) = Annee;

    procedure Afficher (Date : in T_Date);
    function Le_Mois (Date : in T_Date) return T_Mois;
    function Le_Jour (Date : in T_Date) return Integer;

```

```
20      function L_Annee (Date : in T_Date) return Integer;
    end Dates;
```

3.2. Utilisation

```
1  with Ada.Text_IO; use Ada.Text_IO;
2  with Dates; use Dates;
3  procedure Exemple_Dates is
4      Une_Date : T_Date;
5  begin
6      -- Initialiser une date
7      Initialiser (Une_Date, 2, OCTOBRE, 2020);
8      Afficher (Une_Date);
9      New_Line;
10 end Exemple_Dates;
```

4. Généricité

- Spécifier l'unité avec des **paramètres de généricité**

```
1  generic
2      type Un_Type is private;
3  procedure Permuter_Generique (X, Y : in out Un_Type);
```

- Implanter l'unité

```
1  procedure Permuter_Generique (X,Y : in out Un_Type) is
2  Memoire: Un_Type;
3  begin
4  Memoire := X;
5  X:=Y;
6  Y:=Memoire;
7  end Permuter_Generique;
```

- Instancier avec une valeur pour les paramètres de généricité

```
1  procedure Permuter_Entiers is new Permuter_Generique(Un_Type => Integer);
```

- Utiliser l'unité

```
1  A, B: Integer;
2  Permuter(A,B);
```

5. Exceptions (Ada)

```
1  procedure Fonction() is
2  begin
3  -- du code
4
5  exception
6      when Exception_name =>
7      -- bloc à exec
8  end Fonction
```

6. Structure de données dynamiques - Pointeurs

6.1. Algorithmique

```
TYPE T_Ptr_Sur_T_Nom_Type EST POINTEUR SUR T_Nom_Type
Ptr_T : T_Ptr_Sur_T_Nom_Type

PTr_T <-- Null       -- initialisation
Ptr_T <-- NEW ENTIER -- allocation
Ptr_T <-- Ptr_T_1    -- affectation
Ptr_T^               -- déréférencement
```

6.2. Ada

```
type T_Cellule_Caractere; --! pour annoncer la référence en avant

type T_Liste_Caractere is access T_Cellule_Caractere; --! Type pointeur

type T_Cellule_Caractere is record
    Element : Character;
    Suivant : T_Liste_Caractere; -- Accès à la cellule suivante
end record;

-- ...

Liste : T_Liste_Caractere;
Liste.All  --! déréférencement
Liste.All.Element --! accès à Element
```