Analyse et programmation langage ADA

Informatique 2ème année

Date : 16/02/2006

Articles

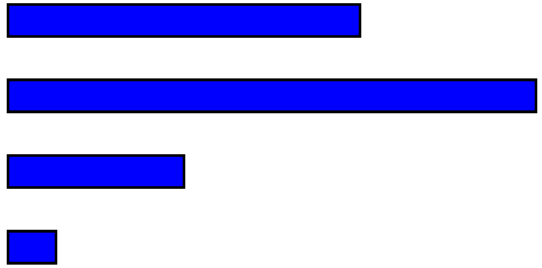
- Présentation d'un enregistrement (Article)

- Présentation d'un type enregistrement sans discriminants
  - Traduction des structures
  - Traduction des types nommés
  - Agrégats et opérations sur les articles
  - Affectation et passage en paramètre de valeurs d'un type article

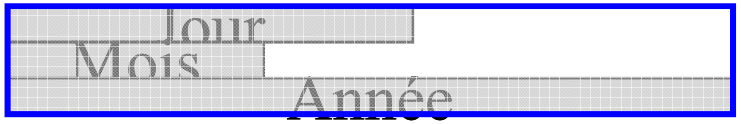
- Présentation d'un type enregistrement à discriminants

Présentation d'un enregistrement

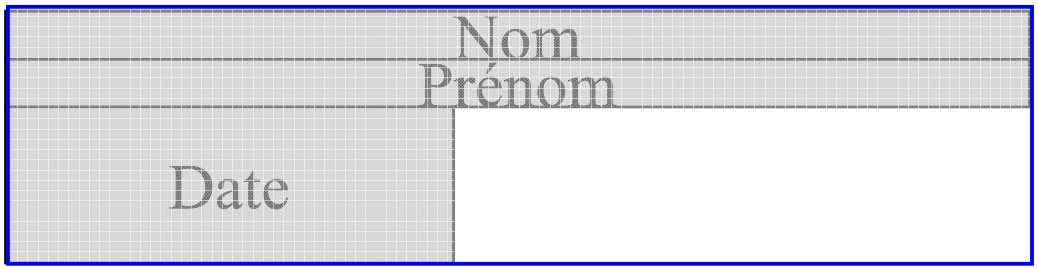
- Types simple:
  - Entier:
  - Réel:
  - Caractère:
  - Booléen



- Type structuré article:
- Date:



- Personne:



Forme générale d'un type article sans discriminants

■ Motivation

- les dates chronologiques (année, mois, jour);
- les nombres complexes (parties réelles et imaginaires, ou module et argument);
- les fiches bibliographiques (titre du livre, auteur, date de parution, ISBN...);
- les fiches personnelles (nom, prénom, âge, sexe, taille...).

```
type identificateur is
  record
    suite_de_champs_1 : identificateur_de_type_ou_sous_type := expression_1;
    suite_de_champs_2 : identificateur_de_type_ou_sous_type := expression_2;
    ...
    suite_de_champs_N : identificateur_de_type_ou_sous_type := expression_N;
end record;
```

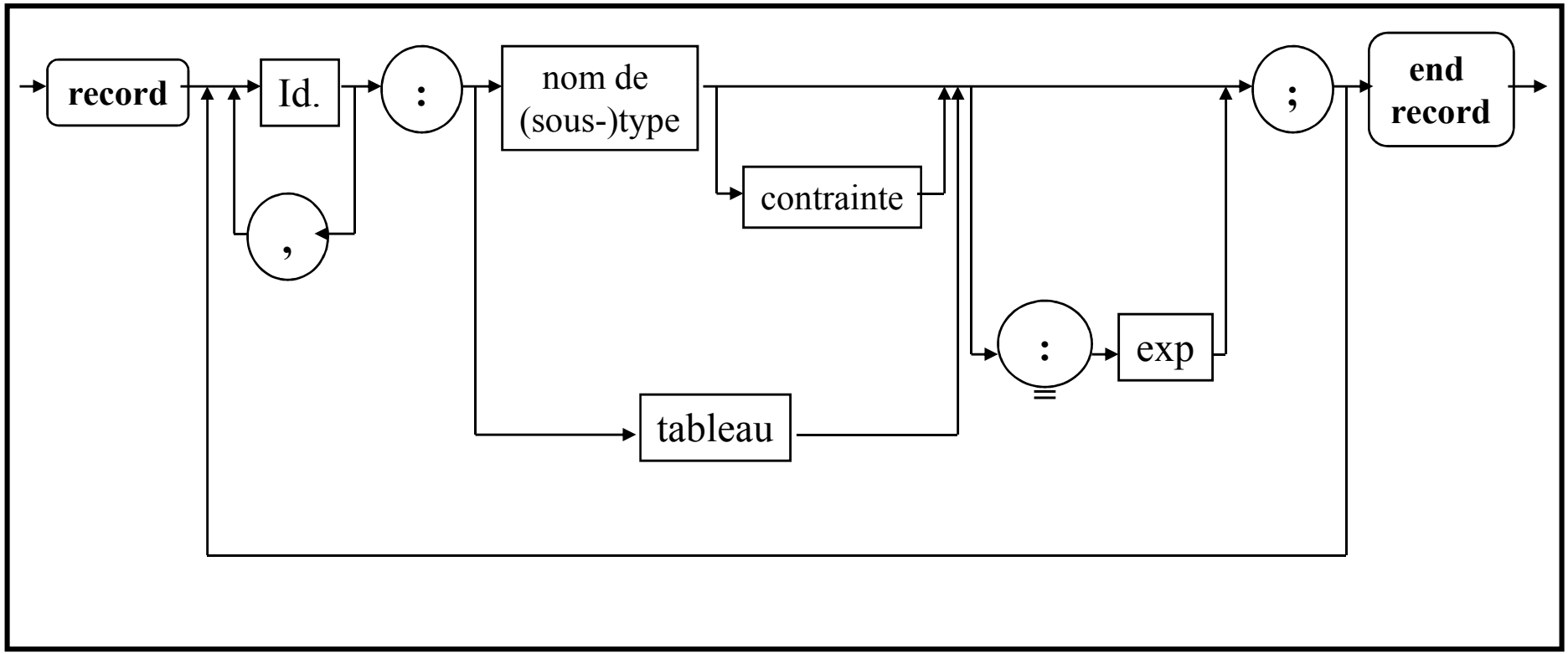
liste d'identificateurs séparés par des virgules

Optionnelle, valeur par défaut du champ

```
subtype T_Jour is Integer range 1..31;
type T_Date is            -- Pour traiter des dates
    record
       Jour  : T_Jour;
       Mois  : String;
       Annee : Integer;
    end record;
type T_Personne is              -- Pour traiter des nombres complexes
    record
       Nom           : String;   -- Champ sans valeur par defaut
       Prenom        : String;   -- Champ avec valeur par defaut
       Date_Naiss  : T_Date;
    end record;
Noel   : T_Date;                -- Pour le 25 decembre 2008
procedure Afficher ( Date : in T_Date ); -- Pour afficher une date
-- Fonction qui retourne la date de naissance d'une personne
function Donner_Date_Naissance ( Personne : in T_Personne ) return T_Date;
```

Présentation d'un enregistrement

- Un *enregistrement* ou *un article* est un ensemble de composantes de (sous-) type quelconques appelés *champ,* dont chacune est repérée par son nom.



Présentation d'un type enregistrement sans discriminants

- Les types composés : les articles

Objet article : l'ensemble du numéro de rue, du nom de rue, du code postal et du nom de la ville est un objet article.

```
record
        Le_Numero       : T_Numeros_Rues;
        La_Rue          : T_Noms_Rue;
        Le_Code         : T_Codes_Postaux;
        La_Ville        : T_Noms_Villes;
end record
```

Objet article : l'ensemble de la partie réelle et de la partie imaginaire est aussi un objet article.

```
record
        Partie_Reelle       : Float  digits 5;
        Partie_Imaginaire   : Float  digits 5;
end record
```

Présentation d'un enregistrement

Types record de types distincts

```
type T_Adresses is
   record
        Le_Numero    : T_Numeros_Rues;
        La_Rue       : T_Noms_Rue;
        Le_Code      : T_Codes_Postaux;
        La_Ville     : T_Noms_Villes;
    end record;

type T_Point is
    record
            Abscisse      : Float  digits 5 := 0.0;
            Ordonnee      : Float  digits 5 := 0.0;
   end record;
```

Traduction des structures

Notation algorithmique

Définition d'une structure dans le lexique :

T_M : < A : entier entre 1 et 99 , B : réel >

ADA

```
type T_M is                          -- T_M est de type enregistrement
    record
        A : Integer range 1..99;
        B : Float;
   end record ;
```

Traduction des structures

Notation algorithmique

Désignation d'un champ :

T_**M.A** désigne le champ A de la structure T_M

ADA

T_M.A désigne le champ A de la structure T_M

Même notation en ADA et en Notation algorithmique

Traduction des types nommés

Notation algorithmique

```
point : type < x, y : réels >

R : un point  avec
R.x = 1 et R.y = 12.2

{ constantes }
S : le point < 12,3 >

{Tableau de points}
TP : le tableau sur [1..3] de points
     [<1,12.2>, <12,3>, <-3,1>]
```

ADA

```
type  T_Point is
                record
                        x,y : Float;
                 end record;

R : T_Point ;
R.x := 1.0;   R.y := 12.2;

S : constant T_Point := ( 12.0, 3.0 );

TP : constant array(1..3) of T_Point :=
          ( (1.0,12.2),(12.0,3.0),(-3.0,1.0) );
```

Traduction des types nommés

Notation algorithmique

```
T_Jour :   type entier sur 1..31
T_Mois : type mois de l'année


T_Date : type < Jour : T_Jour,
                Mois : T_Mois,
                Annee : nombre >




D : T_Date { définition }
```

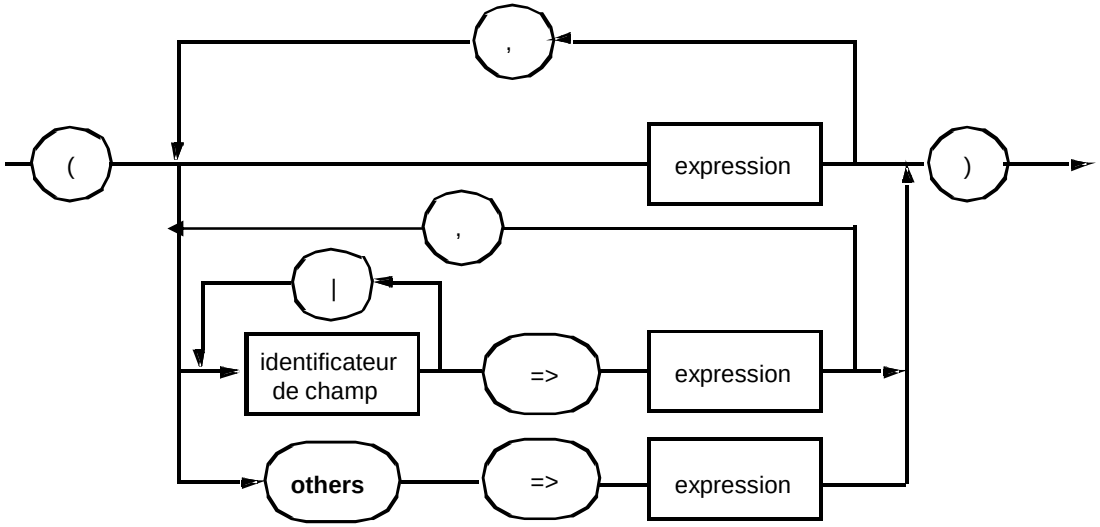
ADA

```
type T_Jour is new Integer range 1..31;
type T_Mois is (Jan, Fev, Mar, Avr,Mai,
     Jun, Jui, Aou, Sep, Oct, Nov, Dec);


type T_Date is
record
       Jour : T_Jour;
       Mois : T_Mois;
       Annee : Natural range 1901..2040;
end record;


D : T_Date;  -- définition
```

Expressions, agrégats et opérations



-- Agregats du type T_Date (probablement)

```
( 24, Novembre, 1997 )                                  -- Agregat par position
( 24, Novembre, Annee => 1997 )                         -- Deux agregats par position
( 24, Mois => Novembre, Annee => 1997 )                 -- puis par nom
( Jour => 24, Mois => Novembre, Annee => 1997 )         -- 2 agregats par nom
( Mois => Novembre, Jour => 24, Annee => 1997 )
```

--Agregats du type T_Complexe

```
( 0.0, -1.0 )
( 0.0, Partie_Imaginaire => -1.0 )
( Partie_Reelle => 0.0, Partie_Imaginaire => -1.0 )
( Partie_Reelle | Partie_Imaginaire => 0.0 )
( others => 0.0 )
( 2.0 * X, Partie_Imaginaire => X / 3.0 )   -- On suppose que X est une variable de type Float
```

Agrégats et affectation

```
D : T_Date:=(15, fev,1995)                                  -- agrégat par position
D : T_Date:=(Mois=>Fev, Jour=>15, Annee =>1995)  -- agrégat par nom
D : T_Date:=(15, Fev, Annee=>1995)                          -- agrégat mixte
D : T_Date:=(15, Annee=>1995, Mois=>Fev)            -- agrégat mixte

R1 : T_Point:=(0.0, 1.5);
R1 : T_Point:=(0.0, y =>1.5);
R1 : T_Point:=(y =>1.5, x=>0.0);
R1 : T_Point:=( others =>0.0);
R1 : T_Point:=( 1.5*a, y=>a/2.5);  -- on suppose que a est un réel
R2 : T_Point;
R2 := (0.0, 1.5);                  -- agrégat ou
R2 := T_Point'(0.0, 1.5); -- agrégat qualifié pour éviter une erreur de compilation
```

Présentation d'un enregistrement

Initialisation et utilisation du type article

```
declare
    -- agrégat par position
     Mon_Adresse : T_Adresses := (10,Marcel,1400,Yverdon);

     -- agrégat par nom
     Mon_Adresse : T_Adresses := (  Le_Numero        => 10,
                                    La_rue           => Marcel,
                                    Le_code          => 1400,
                                    La_Ville         => Yverdon);
begin

    -- Initialisation par agrégat ou champ par champ
     Mon_Adresse.Le_Numero := 10;
end;
```

Agrégats qualifiés.

```
-- Agregats du type T_Rationnel
type T_Rationnel is
        record
                        Numerateur          : Float;
                         Denominateur : Float;
      end record;
T_Rationnel '( 0.0, -1.0 )
T_Rationnel '( others => 0.0 )


-- Agregats du type T_Complexe
type T_Complexe is
        record
                        Partie_Reelle        : Float;
                         Partie_Imaginairee : Float;
      end record;

T_Complexe'( 0.0, -1.0 )
T_Complexe'( others => 0.0 )
```

Remarques

- Les opérations possibles:
  - Affectation
  - Passage en paramètre
  - = /=

- Remarques générales:
  - Les champs d'un type article peuvent être de n'importe quel type.
  - Pas d'entrées-sorties prédéfinies sur les articles.
  - Tout agrégat doit être complet

- Cas limite
  - Un article peut être vide, mais il faut le dire explicitement

```
type Bidon is
    record
        null;
    end record;
```

Exemple

```
procedure Volume is
      type T_Long is   record
                              Dx,  Dy  : Float;
                       end record;
      S1 := T_Long ;
      S2 := T_Long := (1.0, 0.5);
      Haut : constant Float := 2.5;
      V1, V2, V3 := Float;
      function Surface is (S in T_Long ) return Float is      -- S : paramètre formel
      begin
            return S.Dx * S.Dy;
      end Surface;
 begin
            -- Qualification souhaitable mais pas indispensable
            S1 := T_Long ' (others => 1.0);                   -- S1 et S2 paramètres effectifs
            V1 := Haut * Surface(S1);                         -- base d'un carré unitaire
            V2 := Haut * Surface(S2)/Float(2);                -- base d'un triangle
            -- Qualification possible mais pas indispensable
            V3 := Haut * Surface(T_Long ' (0.5,0.5));         -- base d'un carré
  end Volume;
```

Présentation d'un (sous-) type enregistrement à discriminants

- Un *discriminants* est un champ délimité par une paire de parenthèses et localisé entre l'identificateur et le mot réservé **is**.
- Le type d'un discriminant doit être de type discret ou accès
- On peut avoir plus d'un discriminant séparés par ";"

```
type T_Matrice is array (Integer range <>,
                                    Integer range <>) of Float;

-- une matrice de type T_Carrée est carrée !
type T_Carrée (Ordre : Positive) is
  record
     Mat : T_Matrice (1 .. Ordre, 1 .. Ordre);
end record;

Matrice_3 : T_Carrée (3);
subtype T_Carree_3 is T_Carree (3);
```

Discriminant
(exemple)

```
type T_Intervalle is range 0..100;
type T_Tableau (Discriminant : T_Intervalle) is
      record
                  Champ_1 : Integer;
                  Champ_2 : String(1..Discriminant);
      end record;
 Tab : T_Tableau(5);        -- contient un tableau de 5 elements

type T_Coefficients is array (T_Intervalle range <>) of Integer;
type T_Polynome (Degre : T_Intervalle) is   -- Pour traiter des polynomes
      record
            -- Coefficients des termes
            Coefficients : T_Coefficients (T_Intervalle'First..Degre);
      end record;
```

Agrégats, expressions et opérations

- En l'absence d'une valeur par défaut pour un discriminant (cf. 12.3), la déclaration d'un article (constante ou variable) à discriminant doit comporter une valeur pour chacun de ses discriminants, valeur donnée entre parenthèses ou par une valeur initiale (agrégat). Un tel article est dit **contraint**, par analogie avec la déclaration d'un tableau.

- $P(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n$ **avec** a_n **/= 0 si** n **/= 0**

- **Sous-types articles à discriminants**

  subtype identificateur **is** id_type_article (valeur_discr_1, ..., valeur_discr_N);

```
-- Pour utiliser des polynomes de degres 2 et 5
subtype T_Polynome_2 is T_Polynome (2);
subtype T_Polynome_5 is T_Polynome (5);
Quadratique : T_Polynome_2;              -- Un polynome de degre 2
Polynome_Degre_5 : T_Polynome_5;         -- Un polynome de degre 5

-- Pour afficher n'importe quel polynome
procedure Afficher ( Polynome : in T_Polynome );
```

Agrégats, expressions et opérations

-- *Le polynome 3 - 2x + 5x²*
Polynome_1: T_Polynome (2) := (2, (3, -2, 5));

-- *Le polynome x + 7x⁴ déclaré de deux manieres equivalentes*
Polynome_2: T_Polynome (4) := (4, (0, 1, 0, 0, 7));
Polynome_3: T_Polynome := (4, (0, 1, 0, 0, 7));

-- *Le polynome constant -6*
Polynome_4: **constant** T_Polynome (0) := (Degre => 0, Coefficients => (0 => -6));

-- *Le polynome 1+x+x²+...+xⁿ*
Polynome_5: T_Polynome := (N, (0..N => 1)); -- N entier et cf. 9.2.3

- Les opérations:
  - Affectation :=
  - Passage en paramètre
  - Comparaison = /=

Affectation

```
procedure Exemple is
    subtype Intervalle is Integer range 0 .. 100;
    type T_Coefficients is array (Intervalle range <>) of Integer;
    type T_Polynome (Degre : Intervalle) is -- Pour traiter des polynomes
       record
            Coefficients : T_Coefficients (Intervalle'First..Degre);   -- Coefficients des termes
       end record;
    -- Une constante et deux variables de ce type
    Deux_Plus_X : constant T_Polynome := (1, (2, 1));
    Polynome_1 : T_Polynome (1) ;
    Polynome_2 : T_Polynome := (Degre => 2, Coefficients => (2, -4, 1));
    -- Pour afficher n'importe quel polynome
    procedure Afficher ( Polynome : in T_Polynome ) is begin ... end Afficher;
begin -- Exemple
    Polynome_1 := Deux_Plus_X;                    -- Correct car memes
    Polynome_2 := (Polynome_2.Degre, (1, -2, 1)); -- valeurs de
    Polynome_2 := T_Polynome'(2, (1, -2, 1));     -- discriminant
    Afficher ( Deux_Plus_X );                     -- Affichage de deux polynomes
    Afficher ( Polynome_2 );
    Polynome_2 := Polynome_1;  -- Provoque Constraint_Error  (discriminants differents)
end Exemple;
```

Affectation (suite)

```
-- Expression comme valeur de discriminant
Polynome_1 : T_Polynome ( 3 * 4 + N )       -- N entier

-- Fonction a resultat de type T_Polynome
function Polynome_Nul return T_Polynome is
begin -- Polynome_Nul
    return ( 0, ( 0 => 0 ) );
end Polynome_Nul;

-- Correspondance entre valeurs de discriminants, N entier
Polynome_2 : T_Polynome (3) := (N, (1..N => 1) );    -- N egal a 3 sinon
                                                      -- Constraint_Error
```

Valeurs par défaut des discriminants

- L'inconvénient majeur des articles à discriminants réside dans l'impossibilité de modifier la contrainte du discriminant. Le type d'un discriminant doit être de type discret ou accès

- En introduisant une valeur par défaut pour le discriminant lors de la déclaration du type article, ce type devient alors non contraint. On a cependant la possibilité de modifier les champs par la suite.

- La modification de la valeur d'un discriminant d'un article non contraint n'est possible que par une *affectation globale* de l'article, au moyen d'un agrégat.

Valeurs par défaut des discriminants

```
procedure Exemple is  :
    subtype T_Degre is range 0..100;
   type T_Coefficients is array (T_Degre range <>) of integer;
   type T_Polynome (Degre : T_Degre := 0) is   --0 est la valeur par defaut
         record
              Coefficients : T_Coefficients(T_Degre'First.... T_Degre);
         end record;
    -- Une constante et une variable contrainte de ce type
   Deux_Plus_X : constant T_Polynome(1,(2,1));
   Polynome_1  : T_Polynome(1);
   -- Deux variables de ce type non contraintes
    Polynome_2  : T_Polynome := (Degre=>2, Coefficients=>(2, -4, 1));
   Polynome_3  : T_Polynome;
begin
  Polynome_1 := Deux_Plus_X ;
  Polynome_2 := (Polynome_2.Degre, (1, -2, 1)) ;
  Polynome_2 := T_Polynome'(2, (1, -2, 1));    -- qualification souhaitable
  Polynome_3 := Polynome_1 ;
  Polynome_3 := (Polynome_2.Degre - 1, Polynome_2.Coefficients(0..Polynome_2.Degre-1));
end Exemple;
```

Attribut Constrained

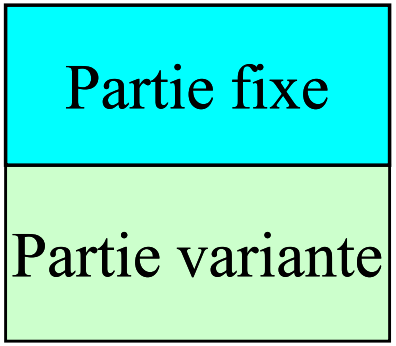
-- Cette procedure normalise le polynome P, c'est-à-dire qu'elle assure
-- que le polynome rendu dans P, de degre n, a son terme $a_n x^n$ non nul.
-- Il faut cependant que le parametre effectif soit non contraint.

```
procedure Normaliser ( P : in out T_Polynome ) is
    Degre_Polynome : Natural := P.Degre;    -- Degre du polynome
begin -- Normaliser
    -- Normalisation possible?
    if not P'Constrained then
        -- Chercher le plus grand coefficient non nul
        while Degre_Polynome>0 and P.Coefficients(Degre_Polynome)=0 loop
                Degre_Polynome := Degre_Polynome - 1;
        end loop;
        P := ( Degre_Polynome,
                P.Coefficients ( Intervalle'First..Degre_Polynome ) );
    end if;
end Normaliser;
```

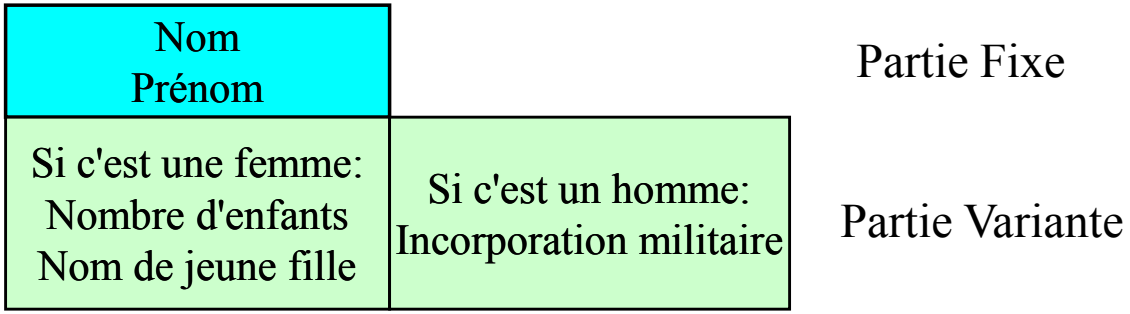
Articles à parties variantes

•Jusqu'à présent:

On va introduire:

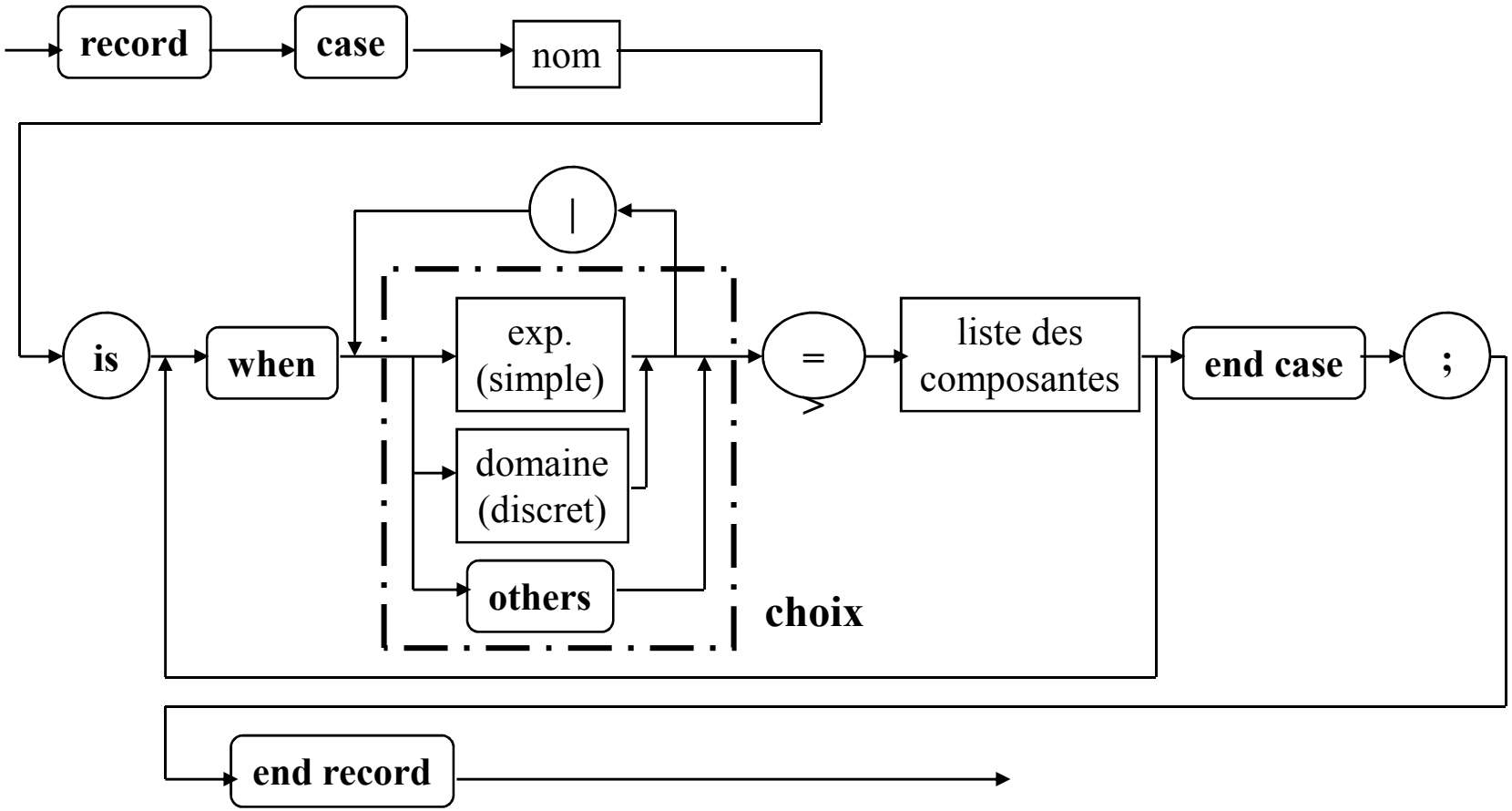


Exemple une personne:



Présentation d'un (sous-) type enregistrement à parties variantes

Une partie variante est une structure permettant de déclarer les champs particuliers de certains articles. La forme générale est :



Partie variante(Exemple)

```
type T_Genre is (Masculin, Feminin);
type T_Indidu (Sexe : T_Genre) is
record
   Naissance : T_Date;
   case Sexe is
      when Masculin =>
         Barbu : Boolean;
      when Feminin =>
         Enfants : Integer;
   end case;
end record;

Pascal : T_Individu (Masculin);
subtype T_Femme is T_Individu (Sexe => Feminin);
```

Présentation d'un (sous-) type enregistrement à parties variantes (suite)

```
case discriminant is
    when choix_1 => suite_de_declarations_de_champs_1;
    when choix_2 => suite_de_declarations_de_champs_2;
    when choix_3 => suite_de_declarations_de_champs_3;
    ...
    when others => autre_suite_de_declarations_de_champs;
end case;
```

- avec
  - discriminant d'un type discret et déclaré comme discriminant d'article;
  - les choix_n statiques, formés de valeurs ou d'intervalles séparés par des barres verticales, valeurs et bornes d'intervalle du type du discriminant;
  - les suite_de_declarations_de_champs_n composées d'une ou de plusieurs déclarations de champs (éventuellement aucune);
  - le mot réservé **others** qui représente toutes les autres valeurs possibles du discriminant.

Présentation d'un (sous-) type enregistrement à parties variantes (suite)

```
type TOrientation (Paysage, Portrait);
type TPeripherique is (Ecran, Imprimante) ;
type TStatus is (Libre, Occupe);
type TSortie(Unite : TPeripherique) is
    record
        Etat : TStatus ;                    -- champs communs
        Vitesse : integer;
        case Unite is
            when Ecran =>       Format : array(1..80, 1..24) of character;
            when Imprimant => Format : array (1..132, 1..56) of character;
                                Orientation : TOrientation;
                                Nb_Copies : integer;
        end case;
    end record;
```

Expressions, agrégats et opérations sur les articles à parties variantes

```
Max : constant := 80;                          -- Longueur maximum d'une ligne
type T_Genre_Fenetre is (Confirmation, De_Taille_Variable, Avertissement);
type T_Fenetre ( Genre : T_Genre_Fenetre := De_Taille_Variable ) is
   record
         Abscisse : Integer;                   -- Position de la fenetre
         Ordonnee : Integer;
         Longueur : Integer;                   -- Dimensions de la fenetre en pixels
         Largeur : Integer;
         case Genre is                         -- Partie variantes
                  when Confirmation =>
                           Texte_Affiche : String ( 1..Max );     -- Selon le texte affiche,
                           Reponse_OK : Boolean;                  -- confirmer ou non
                  when De_Taille_Variable =>
                           Variation_X : Integer := 0;            -- Variations par rapport
                           Variation_Y : Integer := 0;            -- aux dimensions originales
                  when Avertissement =>
                                    Texte_Avertissement : String ( 1..Max );
         end case;
   end record;
Edition : T_Fenetre ( De_Taille_Variable );    -- Article contraint
Graphique : T_Fenetre;                         -- Article non contraint
Erreur_NT : T_Fenetre ( Avertissement );       -- Article contraint
```

Expressions, agrégats et opérations sur les articles à parties variantes

```
Edition : T_Fenetre ( De_Taille_Variable ) :=        -- Utilisation des
                        ( De_Taille_Variable, 0, 0, 600, 400, 0, 0 );   -- valeurs par defaut

-- Graphique est non contraint malgre la valeur initiale
Graphique : T_Fenetre := ( De_Taille_Variable, 0, 0, 600, 400, 10, 20 );

-- Agregat tableau (cf. 9.2.3) pour le texte d'avertissement
Erreur_NT : T_Fenetre ( Avertissement ) :=
                        ( Avertissement, 0, 0, 600, 400, "Profil inconnu" & (15..80=>' ') );
```