CENTRALE
L Y O N

Langages et le Génie Logiciel

Définition
Classifications
Ossature
Surcharge
Packages
Unités génériques
Exceptions
Réutilisation
Langages orientés objets
Réutilisation (principaux concepts)
Polymorphisme
Liaison dynamique

CENTRALE
L Y O N

Bertrand DAVID : Génie Logiciel

Langages de programmation

- **Définition :** Les langages de programmation sont des codes utilisés pour décrire des algorithmes et leurs données sous une forme permettant à la fois leur exécution par une machine et leur compréhension par des lecteurs humains.
- **Programme = Algorithmes + Structures de données** (N. Wirth)

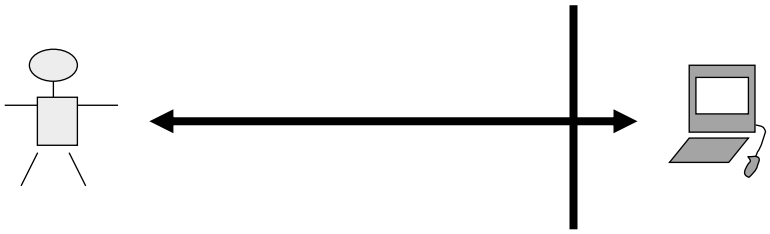
CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Différentes classifications (1)

1/ Selon le niveau de la machine abstraite :

- **machine réelle (microprogrammée)**
  - **LANGAGE D'ASSEMBLAGE**



BTD/GL/GL-LP

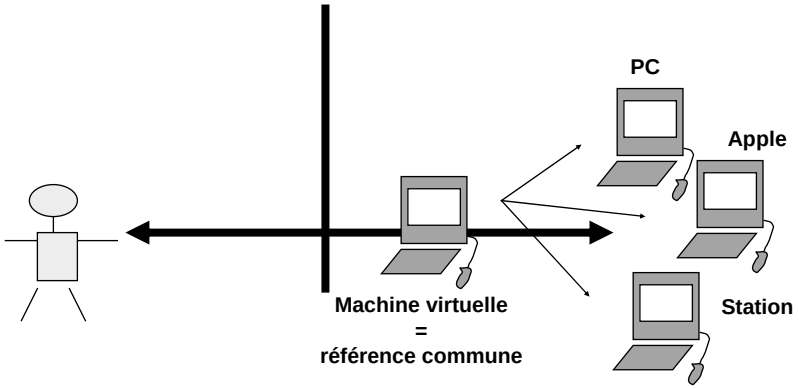
3

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

1/ Selon le niveau de la machine abstraite :

- **machine virtuelle (comportement plus commun)**
  - **LANGAGE DE HAUT NIVEAU (portabilité)**



BTD/GL/GL-LP

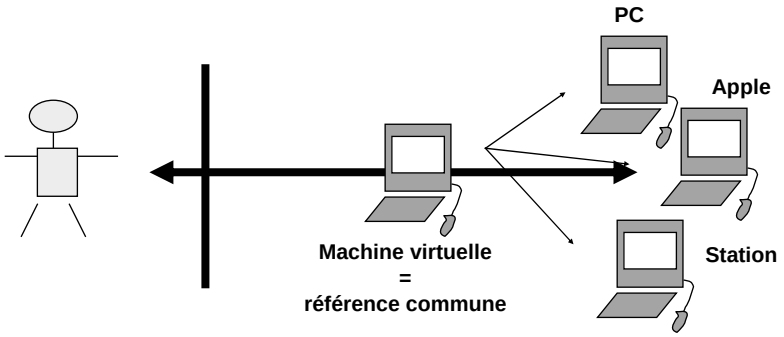
4

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

1/ Selon le niveau de la machine abstraite :

- **machine virtuelle (comportement plus commun)**
  - **LANGAGE DE TRES HAUT NIVEAU (Expression du but)**



BTD/GL/GL-LP

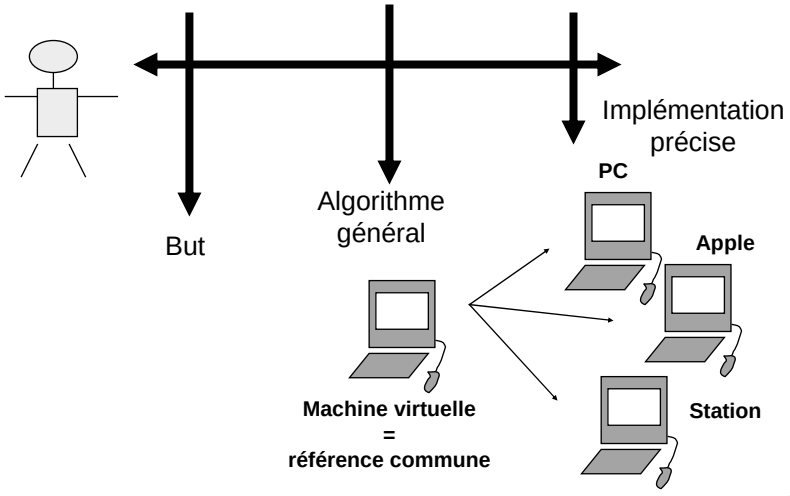
5

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

1/ Selon le niveau de la machine abstraite :

- **Problème majeur : LANGAGES D'ECRITURE DE SYSTEMES**



BTD/GL/GL-LP

6

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Différentes classifications (2)

2/ Par le domaine visé :

- **LANGAGES GENERAUX**
  - **plus ou moins (PASCAL, ADA, FORTRAN, JAVA)**
- **LANGAGES SPECIALISES**
  - **COBOL (gestion), FORMAC (calcul formel), LISP (traitement de listes), PROLOG (IA)**

BTD/GL/GL-LP

7

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Différentes classifications (3)

3/ Par la mise en œuvre :

- **Langages compilés** **(traduction au préalable)**
  - **avantages : vérification préalable, rapidité d 'exécution**
  - **inconvénient : lourdeur du processus**
- **Langages interprétés** **(traduction simultanée)**
  - **avantages : souplesse du processus**
  - **inconvénient : lenteur d'exécution**

BTD/GL/GL-LP

8

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Différentes classifications (4)

4/ Par la façon de gérer les données :

- **Langages statiques**
  - **Allocation statique de variables avec durée de vie égale à la durée du programme**
- **Langages à structure de bloc**
  - **Allocation automatique de variables dans chaque bloc, avec durée de vue égale à la vie du bloc**
- **Langages dynamiques**
  - **Allocation à la demande, durée de vie choisie (suppression à la demande) ou implicite (plus d 'utilisation - ramasse miettes)**

BTD/GL/GL-LP

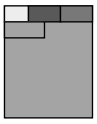
9

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Gestion de mémoire (1)

- **Statique**
  - **Allocation séquentielle**



  - **Durée de vie**
  - **Visibilité**

BTD/GL/GL-LP

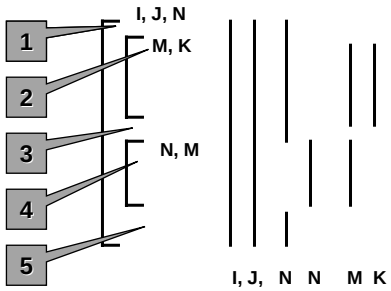
10

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Gestion de mémoire (2)

- **Automatique**
  - Sur une pile



  - Durée de vie
  - Visibilité

1	2	3	4	5
	K		M	
	M		N	
N	N	N	N	N
J	J	J	J	J
I	I	I	I	I

BTD/GL/GL-LP

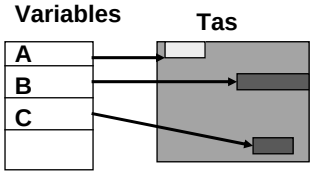
11

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Gestion de mémoire (3)

- **A la demande**
  - sur un tas (heap)



  - Durée de vie
  - Visibilité

BTD/GL/GL-LP

12

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Différentes classifications (5)

5/ Par la façon de contrôler des données :

- **Langages typés**
- **Langages non typés**

BTD/GL/GL-LP

13

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Typage

Notion de type :

- **un ensemble de valeurs possibles,**
- **un ensemble fini d'opérations s'appliquant aux éléments du type et possédant des propriétés connues.**

BTD/GL/GL-LP

14

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Différentes classifications (6)

6/ Par la façon de travailler :

- **Langages impératifs** (variables, affectation et l'ordre)
- **Langages fonctionnels** (ensemble de fonctions et leur composition)
- **Langages logiques** (séparation entre le problème et le contrôle de sa solution : la logique du problème est décrite par le programmeur, la résolution est contrôlée par l'interpréteur)

BTD/GL/GL-LP

15

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Caractéristiques d 'un langage

Deux aspects :

- **Programmation en petit (simplicité, clarté, orthogonalité, homogénéité, régularité, lisibilité, abstractions de contrôle, sécurité, fiabilité, compilabilité, extensibilité)**
- **Programmation en grand (Gestion de la complexité par abstraction – modularité, abstractions, compilations séparées)**

BTD/GL/GL-LP

16

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Programmation en petit (1)

Simplicité :

Définition de

- **COBOL 3 cm,**
- **PASCAL 3 mm,**
- **ADA 300 pages,**
- **COBOL 300 mots réservés**

BTD/GL/GL-LP

17

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Programmation en petit (2)

Clarté :

- **syntaxe : IF (expr.) L1, L2, L3**
- **équivalence de types en PASCAL :**

```
type
    vecteur= array [ 1..10 ] of integer;
var
    vecteur1 : vecteur;
    vecteur2 : vecteur;
    vecteur3 : array [1..10] of integer;
    vecteur4 : array [1..10] of integer;
```

Lesquelles des variables vecteur1, vecteur2, vecteur3, vecteur4 sont considérées du même type?

- **Deux schémas d'interprétation:**
- **- équivalence structurelle: toutes**
- **- équivalence de nom: vecteur1 et vecteur2**

BTD/GL/GL-LP

18

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Programmation en petit (3)

Orthogonalité :

Contre-exemple : Lecture et écriture de scalaires en PASCAL

- **valeurs logiques peuvent être écrites mais pas lues**
- **valeurs numériques peuvent être écrites et lues**
- **valeurs des types énumérés ne peuvent être ni lues ni écrites**

BTD/GL/GL-LP

19

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Programmation en petit (4)

Homogénéité, régularité, lisibilité :

Exemples : le point-virgule

- séparateur en PASCAL

- symbole de terminaison en ADA

(10 fois moins de fautes)

- if ....then .....else ......;

=> begin .....end

=> interprétation de else dans des if imbriqués

if .... then if .... then .... else ....

soit else vide, soit begin .....end

=> utilisation du mot-clé explicite endif, ...

BTD/GL/GL-LP

20

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Programmation en petit (5)

Lisibilité :

- **Exemples :**
  - **indentation,**
  - **identificateurs significatifs,**
  - **opérateurs avec une seule signification (=)**
  - **mot-clés réservés (if)**

BTD/GL/GL-LP

21

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Programmation en petit (6)

Abstractions de contrôle :

séquence,
sélection,
répétition

```
if cond1 then ins1
elsif cond2 then ins2
elsif cond3 then ins3
else ins4
endif
```

BTD/GL/GL-LP

22

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Programmation en petit (7)

Sécurité, fiabilité, compilabilité :

Exemples: DO 10 I= 1.100

Typage et typage fort (à la compilation)
typage statique (vérifiable à la compilation)
typage dynamique (vérifiable seulement à l'exécution)

BTD/GL/GL-LP

23

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Programmation en petit (8)

Extensibilité (1) :

- **Types de base et opérations associées**
- **Déclarations explicites (pas de déclarations implicites comme en FORTRAN: 'I',...,'N')**
- **Abstraction des données par la construction de nouveaux types constructeurs ensemble, type énuméré, intervalle,…**
- **types structurés: tableaux et structures - bornes fixes ou variables, initialisation, affectation entre tableaux, structures à variantes**

BTD/GL/GL-LP

24

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Programmation en petit (9)

Extensibilité (2) :

- **Abstraction procédurale (quoi avant comment)**
  - **procédures et fonctions**
  - **passages de paramètres : IN, OUT, IN-OUT (par valeur ou par référence)**
  - **portée et visibilité des variables**

BTD/GL/GL-LP

25

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Programmation en grand (1)

Gestion de la complexité par abstraction :

Modularité :

- **abstraction des données et des traitements**

Encapsulation :

- **types abstraits (objets et traitements associés)**
- **séparation entre la spécification et la réalisation**
- **protection des informations privées**

BTD/GL/GL-LP

26

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Programmation en grand (2)

Compilations séparées :

- compilations indépendantes
- accès facile à la bibliothèque des modules précompilés
- gestion des recompilations

BTD/GL/GL-LP

27

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Ossature d'un langage de programmation

Syntaxe :

Ensemble de règles gouvernant la formation de programmes.

Sémantique :

Définition du sens associé au programme.

BTD/GL/GL-LP

28

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Définitions

Si d'après Wirth :

Programme = Algorithmes + Structures de données

on peut compléter :

Système logiciel = Programmes + Modularité

Un langage de programmation permet d'écrire des systèmes logiciels dans un formalisme adéquat constitué de trois éléments:

- **les données (objets manipulés par le programme),**
- **les traitements (unités d'écriture des algorithmes),**
- **la modularité (mode de division du système).**

BTD/GL/GL-LP

29

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Données (1)

Donnée : Information interne et externe, que le programme peut créer, utiliser, combiner, échanger et détruire.

A chaque donnée est associé un certain nombre d'attributs dont les plus importants sont le droit d'accès et le type (ensemble de valeurs et opérations possibles).

Droits d'accès:

constante- nom symbolique d'une valeur fixe

variable - nom symbolique dont valeur peut changer pendant l'exécution grâce à l'instruction dite affectation

d'autres caractéristiques des droits d'accès sont :

opérations permises

visibilité, modifiabilité, partage, ...

BTD/GL/GL-LP

30

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Données (2)

Notion de type :

- **un ensemble de valeurs possibles,**
- **un ensemble fini d'opérations s'appliquant aux éléments du type et possédant des propriétés connues.**

Tous les langages offrent des types de base, prédéfinis, et des procédés de construction, simples ou évolués, permettant d'obtenir des objets de types plus complexes.

Types prédéfinis les plus courants sont :

- **types numériques (entiers, réels, complexes),**
- **type logique ou booléen,**
- **type chaîne de caractères,**
- **type programme, dont les éléments désignent des sous-programmes, des fonctions avec des opérations permises limitées (passage comme argument à un sous-programme, affectation)**

BTD/GL/GL-LP

31

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Données (3)

Constructeurs de données composées:

- **tableaux : collections homogènes et finies de données,**
- **fichiers séquentiels : collections homogènes de données, en nombre généralement inconnu au programme a priori,**
- **enregistrements ou articles : collections hétérogènes et finies de données,**
- **types énumérés : des objets appartenant à un ensemble donné par énumération des noms symboliques de ses éléments,**
- **types ensembles : les sous-ensembles d'un ensemble fini donné,**
- **type liste : des suites finies d'objets dont chacun est soit un atome, soit à son tour une liste,**
- **type pointeur : références aux éléments d'un type donné.**

BTD/GL/GL-LP

32

CENTRALE L Y O N

Bertrand DAVID : Génie Logiciel

Traitements

Il s'agit d'exprimer des manipulations sur des objets: calculs numériques, formels, textuels,...

Il s'agit de prendre en compte des traitements élémentaires (opérations) et des mécanismes de combinaison d'actions (structures de contrôle).

- **Opérations élémentaires : s'appliquent à un ou plusieurs objets et permettent d'en examiner et/ou d'en modifier certains attributs.**
- **Structures de contrôle : permettent de combiner des opérations pour constituer de véritables algorithmes, c'est-à-dire des modèles décrivant des suites avec une structure de décision éventuellement complexe. Les trois structures principales sont: enchaînement, choix et boucle**

BTD/GL/GL-LP

33

CENTRALE L Y O N

Bertrand DAVID : Génie Logiciel

Modularité

Organisation d'un système complexe en éléments compris et maitrisables.

- **Sous-programmes : une étape identifiable, répétable, réutilisable d 'un traitement (factorisation ou paramétrage)**
- **Module : ensemble de données et de sous-programmes contribuant à la résolution d'un sous-problème.**

BTD/GL/GL-LP

34

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA

ADA est unique dans les anales de l'informatique...

- **Unique dans la manière dont ses caractéristiques ont évolué, pour la qualité et le nombre de ses commentateurs, par le système de mise en concurrence par lequel il fut sélectionné et finalement par ce qui a résulté de ces efforts.**

Commentaires dans la presse :

- **ADA un nouveau langage de programmation**
- **Enfin un langage à la hauteur de la technologie actuelle et future,**
- **Un monstre de complexité technique et logique que l'on ne maîtrisera jamais,**
- **Un autre PL1 qui ne parviendra jamais au stade industriel.**

Le choix du nom ADA rend hommage à Lady Augusta ADA Byron, comtesse de Lovelage (1815-1852), qui fut le premier programmeur de l'histoire sur la machine analytique de Babbage.

BTD/GL/GL-LP

35

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA - Un peu d'histoire (1)

En 1974 le constat du Ministère américain de la Défense (D.O.D.) :

- **Il y a trop de langages et trop de dialectes incompatibles.**
- **Le logiciel est trop difficile et trop cher à maintenir.**
- **Ce coût dépasse de beaucoup le coût de développement initial.**
- **Les programmes sont trop difficiles à adapter quand le système change.**

En 1975-1976 élaboration d'un ensemble d'exigences à satisfaire par un langage unique de haut niveau qui apportera une solution à ces problèmes (rapports Strawman, Woodenman, Tinman).

En 1976 étude des langages existants (COBOL, FORTRAN, PASCAL, JOVIAL, PL1,...) pour savoir s'ils satisfont ces exigences. Réponse : NON

BTD/GL/GL-LP

36

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA - Un peu d'histoire (2)

Rapport IRONMAN introduit la notion de composant logiciel (un produit commercialisable, fiable, pouvant être integré dans un ensemble) à travers de trois concepts :

1/ Les langages fortement typés (PASCAL, ALGOL68) :
pour la rigueur et la simplicité qu'ils apportent dans l'écriture des algorithmes,

2/ La modularité (SIMULA, LIS) :
pour les gains de temps et la simplicité qu'elle introduit dans le processus de compilation,

3/ La flexibilité (EUCLID, CLU) :
pour les types abstraits, qui fournissent un niveau supplémentaire de paramétrisation des applications.

BTD/GL/GL-LP

37

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA - Un peu d'histoire (3)

Objectif : Un langage universel, portable, doté de structures modernes

Exigences à remplir :

Langage algorithmique : types définissables par le programmeur et le contrôle d'utilisation de ces types,

Langage modulaire : qui intègre largement la compilation séparée et la rend finalement accessible à l'utilisateur, et qui permet la paramétrisation des entités (les unités génériques),

Langage temps-réel : permettant le parallélisme des tâches et des entrées/sorties, le contrôle du temps et le traitement des exceptions,

Langage d'écriture de systèmes : avec la possibilité de définir la localisation et la représentation des données et de faire des conversions forcées de types.

BTD/GL/GL-LP

38

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA - Un peu d'histoire (4)

Processus de sélection

En 1977 : appel d'offres avec 17 réponses.

Première sélection 17 -- > 4

GREEN - CII HB
RED - INTERMETRICS
BLUE - SOFTECH
YELLOW- SRI

Deuxième sélection 4 -- > 2 (GREEN & RED)

Complément de spécifications avec des précisions sur des techniques d'implémentation et de faisabilité par rapport à l'état de l'art.

le 2 mai 1979 GREEN is ADA

Norme ANSI (American National Standard Institute) en1983,
Norme ISO et sa version française en 1987

BTD/GL/GL-LP

39

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA - Un peu d'histoire (5)

ADA entre dans sa phase industrielle :

Le processus de validation :

Pour pouvoir s'appeler compilateur ADA

"ADA is a registered trademark of the US Government ADA Joint Program Office"

le produit doit subir sous contrôle officiel une batterie de tests

ACVC: ADA Compiler Validation Capability

Il s'agit de vérifier la conformité à la norme et de mettre en évidence les choix d'implémentation (et donc leur incidence sur la portabilité)

Délivrance d'un certificat valable un an

BTD/GL/GL-LP

40

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (1)

Le typage correspond à plusieurs besoins :

- la maintenabilité,
- la lisibilité,
- la fiabilité,
- la réduction de la complexité.

Un type de donnée abstrait caractérise :

-
- un ensemble d'opérations associées
- un ensemble de valeurs,

BTD/GL/GL-LP

41

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (2)

On dispose en ADA de plusieurs types :

- **les types de données scalaires (**entier, réel, énumératif)
- **les types de données composites (**tableau, enregistrement)
- **les types de données accès**
- **les types de données privés**
- **les sous-types et types dérivés**

BTD/GL/GL-LP

42

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (3)

Type scalaire :

Types entiers :

```
SHORT_INTEGER                -- type prédéfini
LONG_INTEGER                 -- type prédéfini
INTEGER                      -- type prédéfini
type INDICE is range 1..50   -- type défini par l'utilisateur
```

Types réels :

```
LONG_FLOAT                   -- type prédéfini
SHORT_FLOAT                  -- type prédéfini
FLOAT                        -- type prédéfini
type  MASSE is digits 10;    -- type défini par l'utilisateur
type VOLTAGE is delta 0.01 range  -12..+24.0
                             -- type défini par l'utilisateur
```

BTD/GL/GL-LP

43

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (4)

Types énumératifs :

```
BOOLEAN                      -- type prédéfini (FALSE,TRUE)
CHARACTER                    -- autre type prédéfini
type COULEUR is (NOIR, ROUGE, VERT, BLEU, CYAN,
MAGENTA, ORANGE, BLANC)      -- type défini par l'utilisateur
```

Déclaration du type :

```
type ETAT is (NORMAL, FATIGUE, SAOUL,ENERVE)
```

Déclaration de variables :

```
ETAT_DU_CLODO : ETAT:= SAOUL ;
ETAT_NORMAL : constant ETAT := NORMAL ;
MON_ETAT : ETAT ;
```

La variable peut être initialisée à la déclaration.

BTD/GL/GL-LP

44

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (5)

- **<u>Un type énumératif</u> définit une liste ordonnée de gauche à droite. Les seules valeurs autorisées sont celles de la liste.**
- MON_ETAT := BOF sera refusé à la compilation
- Les opérations applicables sont le test d'(in)égalité, le test d'ordre (ENERVE > NORMAL) et l'affectation et quelques attributs.

BTD/GL/GL-LP

45

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (6)

<u>Attributs :</u>

```
type JOUR is (LUN, MAR, MER, JEU, VEN, SAM, DIM);
        JOUR'first              -- LUN
        JOUR'last               -- DIM
        JOUR'succ (MER)         -- JEU
        JOUR'pos (LUN)          -- 0 (LUN=0,..., DIM=6)
        JOUR'val (1)            -- MAR
```

Exemples de déclarations :

```
I : INTEGER ; P: INTEGER := 38 ;
I, J, K : FLOAT ; P, Q, R : FLOAT := 38.0 ;
```

Déclaration de constantes :

```
DIX : constant := 10 ;   N : constant INTEGER := 10;
```

on peut écrire

```
M : constant  := 10; MM : constant := M*M;
```

BTD/GL/GL-LP

46

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (7)

Types composites :

Tableaux

Un tableau est un objet composé, formé d'un certain nombre de composants de même type.

un tableau peut avoir une dimension fixe :

```
A: array ( INTEGER range 1..6 ) of REEL ;
```

Un tableau peut avoir plusieurs dimensions, auquel cas un intervalle distinct est donné pour chaque dimension.

```
AA : array (INTEGER range 0..2, INTEGER range 0..3) of REEL;
```

Un indice de tableau peut être de n'importe quel type discret.

```
Type JOUR is ( LUN, MAR, MER, JEU, VEN, SAM, DIM);
HEURES_OUVREES : array (JOUR) of REEL;
```

BTD/GL/GL-LP

47

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (8)

```
JOUR_OUVRABLE : JOUR range LUN..VEN;
for D in JOUR_OUVRABLE loop
        HEURES_OUVREES (D) := 8.0 ;
end loop;
HEURES_OUVREES (SAM) := 0.0;
HEURES_OUVREES (DIM) := 0.0;
```

Les tableaux possèdent plusieurs attributs relatifs à leurs indices.

A'First : Donne la borne inférieure du premier (ou l'unique) intervalle d'indice de A.

A'Last : Donne la borne supérieure du premier (ou l'unique) intervalle d'indice de A.

A'Length : Donne le nombre de valeurs du premier (ou l'unique) intervalle d'indice de A.

A'Range : est une forme abrégée de A'First..A'Last.

BTD/GL/GL-LP

48

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (9)

Exemples :

```
HEURES_OUVREES'First     -- donne LUN
HEURES_OUVREES'Last      -- donne VEN
HEURES_OUVREES'Length    -- donne 7
HEURES_OUVREES'Range     -- donne LUN..VEN
```

En ADA, on a la possibilité de ne pas fixer les bornes à la déclaration de type :

```
type ANNEE is array (INTEGER range <>) of JOUR;
```

Elles sont fixées à la déclaration des variables :

```
ANNEE_87 : ANNEE (1..365);
ANNEE_88 : ANNEE (1..366);
```

BTD/GL/GL-LP

49

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (10)

Tableau à dimension variable :

Une règle subsiste : les indices de tableaux doivent être constants dans le bloc où ils sont visibles:

```
Procedure RUSEE (A: INTEGER; DEMAIN: JOUR) is
        TABLEAU_CHANGEANT: ANNEE (1..A);
        subtype JOUR_PROCHE is JOUR range LUN..DEMAIN;
        BEGIN  ….  END;
```

```
Procedure SUPER_RUSEE is
   A: INTEGER; DEMAIN: JOUR;
BEGIN …  A:=A+1; DEMAIN := MER;
   DECLARE
        TABLEAU-CHANGEANT: ANNEE (1..A);
        CES_JOURS_CI: JOUR range LUN.. DEMAIN;
   BEGIN … END;
END SUPER_RUSEE;
```

BTD/GL/GL-LP

50

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (11)

Type composite Article

Un article est un objet composite constitué de composants nommés pouvant être de types différents.

Exemples :

```
type NOM_DE_MOIS is (JAN, FEV, MAR, AVR, MAI, JUN, JUL, AOU,
    SEP, OCT, NOV, DEC);
type DATE is record JOUR : INTEGER range 1..31;
       MOIS : NOM_DE_MOIS;
       ANNEE : INTEGER;
       end record;
déclaration de variables :  D: DATE;
initialisation : D.JOUR := 14; D.MOIS := JUL; D.ANNEE := 1789;
nous pouvons écrire : D: DATE := (14, JUL, 1789);   E: DATE;
puis  E:= D;
ou encore E:= (ANNEE => 1789, MOIS => JUL, JOUR =>14);
```

BTD/GL/GL-LP

51

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (12)

Articles à variantes

```
types GENRE is (M, F);
type PERSONNE is (SEXE: GENRE := F) is
   --  valeur par défaut de sexe à F
record NAISSANCE: DATE;
   case SEXE is       when M => BARBU: BOOLEAN;
                      when F => ENFANTS : INTEGER;
   END case;
END record;
```

Déclaration et initialisation par nom :

```
   PIERRE: PERSONNE :=
   (SEXE => M, NAISSANCE => (6,8, 1947), BARBU => FALSE);
```

Notation pointée pour accès aux articles :

```
   PIERRE.SEXE     -- M
   PIERRE.NAISSANCE        -- (6, 8, 1947)
   PIERRE. BARBU -- FALSE
```

Mais il n'est pas possible de faire PIERRE.ENFANTS

BTD/GL/GL-LP

52

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (11)

Types accès

Ada fournit le type accès pour permettre de manipuler des objets dynamiques :

- **des objets peuvent être créés et détruits de façon imprévisible,**
- **plusieurs noms peuvent se référer au même objet,**
- **les relations entre les objets peuvent changer au fil du temps.**

```
type CELLULE ;
type LIEN is access CELLULE;
type CELLULE is record
        VALEUR : INTEGER;
        SUIVANT : LIEN ;
        end record;
Déclaration        L : LIEN;
Allocation L := new CELLULE ;
  L.VALEUR := 37 ;
```

BTD/GL/GL-LP

53

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (12)

Supposons que nous voulons créer un autre article : N: LIEN

puis **N:= new CELLULE'(10,L);**

L:= N; -- copie les valeurs accès

Remarques :

1/ l'instruction L:= N ; copie les valeurs accès

L.all := N.all ; équivaut à

L.VALEUR := N.VALEUR;

L.SUIVANT := N.SUIVANT;

2/ on peut déclarer une constante d'un type accès mais, bien sûr, on doit lui fournir une valeur initiale.

C : constant LIEN := new CELLULE'(0,null);

En tant que constante, C doit toujours se référer au même objet. Cependant, la valeur de l'objet elle-même peut être changée. Ainsi :

C.all := L.all ; est autorisé, mais :

C:= L; ne l'est pas.

BTD/GL/GL-LP

54

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (13)

3/ Dans l'exemple précédent, nous n'avions pas réellement besoin de la variable N pour étendre la liste puisque nous pouvions simplement écrire :

```
L:= new CELLULE'(V,L);
```

Cette instruction peut être transformée en une procédure pour créer un nouvel article et le rajouter en tête de liste.

```
procedure  AJOUTER_A_LISTE (LISTE: in out LIEN;
                            V in INTEGER) is
begin
    LISTE := new CELLULE'(V, LISTE);
end;
```

BTD/GL/GL-LP

55

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (14)

Types privés

ADA offre une déclaration de type privé qui permet de décrire explicitement des abstractions de plus haut niveau (NOMBRE_COMPLEXES, PILES **etc..).**

Les types de données privés se déclarent dans une spécification de paquetage.

```
package MOT_DE_PASSE is
        type VALEUR is limited private ;
        function EST_VALIDE (CODE : VALEUR) return BOOLEAN;
        procedure DONNER (CODE : out VALEUR;
    NIVEAU_DE_SECRET : in INTEGER);
private
        type VALEUR is range 0..1_000_000;
end MOT_DE_PASSE;
```

BTD/GL/GL-LP

56

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (15)

Utilisation du package :

AUTORISATION_UTILISATEUR: MOT_DE_PASSE.VALEUR;

Ce package définit un type privé appelé
MOT_DE_PASSE.VALEUR

Les seules opérations disponibles sont :
EST_VALIDE et **DONNER**

L'affectation, les tests d'égalité et d'inégalité ne sont pas autorisés (elles le seraient si on avait déclaré VALEUR comme private (simple et non pas limited).

BTD/GL/GL-LP

57

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (16)

Sous-types

Un sous-type ne définit pas un nouveau type, il fournit un nouveau nom pour un autre type de données, (potentiellement) contraint :

```
type NOM_DE_MOIS is ( JAN, FEV, MAR, AVR, MAI, JUN,
JUL, AOU, SEP, OCT, NOV, DEC);
subtype ETE is NOM_DE_MOIS range JUN..AOU;
MOIS_COURANT : NOM_DE_MOIS;
VACANCES : ETE
```

Les opérations suivantes sont légales :

```
MOIS_COURANT := VACANCES;

VACANCES := ETE'SUCC (MOIS_COURANT);
```

Dans le dernier cas, si MOIS_COURANT **avait une valeur autre que** MAI, JUN ou JUL**, l'exception** CONTRAINT_ERROR **serait levée au moment de l'affectation à** VACANCES**.**

BTD/GL/GL-LP

58

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Les types (17)

Types dérivés

Les types dérivés, contrairement aux sous-types, définissent des types distincts, mais dont la structure est similaire.

```
type S is new T;
TX: T;
SX: S;
```

les instructions:

TX := SX; ou SX := TX; **sont illégales**

mais, on peut écrire : TX := T(SX); SX := S(TX);

BTD/GL/GL-LP

59

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Les instructions classiques (1)

Les instructions ADA comprennent :

Contrôle séquentiel : affectation, appel de sous-programme, bloc, null, return

– **Instruction d'affectation** (:=), donne à une variable une valeur nouvellement calculée.

```
COMPTEUR := COMPTEUR+1;          -- affectation simple
ANNIVERSAIRE.ANNEE := 1955;      --affectation à un article
```

– **Les sous-programmes** permettent d'invoquer des algorithmes nommés.

```
REMPLIR_RESERVOIR ;    -- appel de procédure sans paramètre
VALEUR := TAN (ANGLE);  -- appel de fonction
FAIRE_TOURNER (POINT,30.0) -- appel de procédure
                           -- avec paramètres par position
FAIRE_TOURNER (POINT, ANGLE => 17.6);
-- appel de procédure avec paramètres premier par position, second par nom
```

BTD/GL/GL-LP

60

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Les instructions classiques (2)

- **Instruction null :** le seul effet de l'instruction null est de passer simplement le contrôle à l'instruction suivante.
- **Instruction return :** termine l'exécution d'une procédure ou d'une fonction. Lors de l'exécution de cette instruction, l'exécution de la procédure s'arrête et le contrôle retourne à l'environnement appelant.
- **Instruction bloc** permet d'encapsuler textuellement une séquence d'instruction, accompagnée de déclarations locales.

```
ECHANGE:
        declare  TEMP : FLOAT ;
        begin
        TEMP := VOLTAGE_1;
        VOLTAGE_1:= VOLTAGE_2;
        VOLTAGE_2:= TEMP;
        end ECHANGE;
```

BTD/GL/GL-LP

61

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Les instructions classiques (3)

Contrôle conditionnel

- **Instructions de contrôle conditionnel : SI et CHOIX**
- **Instruction SI** sélectionne pour l'exécution une séquence d'instructions au plus parmi plusieurs, selon la valeur logique d'une ou de plusieurs conditions correspondantes.
- **Il existe trois formes de base de SI:**

```
        -- simple  si...alors..
if  COMPTE_1 < 5 then  COMPTE_1 := 9; end if

        -- si...alors...sinon
if COMPTE_1 < 5 then COMPTE_1 := 9;
 else  COMPTE_1 := 18; end if;

        -- structure de if imbriqués
if VOLTAGE_1>VOLTAGE_2 then VOLTAGE_1 := VOLTAGE_2;
elsif VOLTAGE_1<VOLTAGE_2 then VOLTAGE_2:= VOLTAGE_1;
end if;
```

BTD/GL/GL-LP

62

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Les instructions classiques (4)

Instruction CHOIX permet de sélectionner un chemin d'exécution parmi plusieurs, d'après la valeur d'une expression discrète.

```
case ETAT_DE_PROCESSUS is
    when ACTIF => TABLE_ORDONNANCEMENT (ACTIF) := 1;
                  EST_ACTIF := TRUE;
    when PRET => TABLE_ORDONNANCEMENT (PRET):=
                  TABLE_ORDONNANCEMENT (PRET)+ 1;
                  EST_ACTIF := FALSE;
     when  BLOQUE  => TABLE_ORDONNANCEMENT(BLOQUE):=
TABLE_ORDONNANCEMENT (BLOQUE)+ 1;
                  EST_ACTIF := FALSE;
    when MORT => TABLE_ORDONNANCEMENT (MORT) :=
                  TABLE_ORDONNANCEMENT (MORT)+ 1;
                  EST_ACTIF := FALSE;
 end case;
case COMPTE_1 is
    when  1 => ARTICLE_VANNE(COMPTE_1).OPEN := TRUE;
    when  2/3  => ARTICLE_VANNE(COMPTE_1).OPEN := FALSE;
    when 5..10 => ARTICLE_VANNE(COMPTE_1).OPEN := FALSE;
    when others => ARTICLE_VANNE(COMPTE_1).OPEN := TRUE;
 ARTICLE_VANNE(COMPTE_1).DEBIT := 1.0;
end case;
```

BTD/GL/GL-LP

63

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Les instructions classiques (5)

Contrôle iteratif :

La structure de contrôle itérative permet d'exécuter une suite d'instructions plusieurs fois (éventuellement zéro). Trois formes de boucles :

boucle de base, boucle for, boucle while

La boucle de base est la forme la plus simple d'interaction; elle est structurée de façon à boucler éternellement : **loop .....… end loop;**

Nous utiliserons l'instruction exit pour quitter une boucle :

```
exit;      -- quitter la boucle
exit when TEMP > MAX_TEMP ;  -- quitter la boucle si la condition est vrai
```

Si nous avons plusieurs niveaux de boucles, nous pouvons les nommer explicitement pour quitter un niveau.

```
BOUCLE_EXTERNE: loop  .....
   BOUCLE_INTERNE: loop
                  .…. exit BOUCLE_EXTERNE ; -- quitter la boucle nommée
   end loop BOUCLE_INTERNE;
.....
end loop BOUCLE_EXTERNE;
```

BTD/GL/GL-LP

64

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Les instructions classiques (6)

La boucle POUR est également appelée boucle à compteur, car la boucle se répète un nombre de fois que l'on peut compter.

```
for INDICE in ACTIF..MORT loop
    TABLE_ORDONNANCEMENT( INDICE) := 0;
end loop;

for INDICE in reverse TOTAL_DE_VANNE loop
    ARTICLE_VANNE (INDICE).OUVERT := FALSE;
end loop;
```

Dans la boucle TANTQUE une suite d'instructions est répétée tant qu'une certaine condition est vraie.

```
while ( ARTICLE_VANNE(1).DEBIT > 10.0) and (not EST_VIDE))
loop
    .........
end loop;
```

BTD/GL/GL-LP

65

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Les sous-programmes

Les sous-programmes constituent les unités exécutables de base des programmes ADA, il en existe 2 formes :

Fonctions et Procédures

Une procédure nomme un ensemble d'instructions qui définissent une action de haut niveau.

Les fonctions font la même chose, mais leur rôle principal est de renvoyer une valeur calculée.

Les sous-programmes peuvent être déclarés partout où une déclaration d'objet est autorisée.

BTD/GL/GL-LP

66

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Les fonctions (1)

```
function SIGNE ( X: INTEGER) return INTEGER is
        begin
                if X>0 then
                        return +1;
                elsif X<0 then
                        return -1;
                else
                        return 0;
        end SIGNE;
```

En ADA, il est possible d'appeler une fonction de façon récursive :

```
function FACTORIELLE (N : POSITIVE) return POSITIVE is
        begin
                if N= 0 then
                        return 1;
                else    return N*FACTORIELLE(N-1);
                end if;
        end FACTORIELLE;
```

BTD/GL/GL-LP

67

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Les fonctions (2)

Remarques :

Un paramètre formel peut être de n'importe quel type, mais ce type doit avoir un nom.

Un paramètre formel peut être un type tableau non contraint tel que:

```
type VECTEUR is array (INTEGER range <>) of REEL;
```

Dans un tel cas les bornes du paramètre formel sont déduites de celles du paramètre effectif.

```
function SOMME ( A : VECTEUR) return REEL is
    RESULTAT : REEL := 0.0;
    begin
      for I in A'Range loop
              RESULTAT := RESULTA+A(I);
      end loop;
      return RESULTAT;
    end SOMME;
```

BTD/GL/GL-LP

68

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Les fonctions (3)

Nous pouvons alors écrire

```
V: VECTEUR (1..4) := (1.0,2.0,3.0,4.0);
S: REEL;
S := SOMME(V);     --   S=10
```

Pour la fonction suivante :

```
function INVERSE ( X: VECTEUR) return VECTEUR is
   INV : VECTEUR ( X'RANGE);
              begin
                        for I in X'RANGE loop
                        INV(I) := X(X'First+X'Last -I);
              end loop;
                        return INV;
   end INVERSE;
```

on peut écrire :

```
INVERSE(Y) (I);
```

qui dénote le composant indexé par I du tableau retourné par l'appel à INVERSE.

BTD/GL/GL-LP

69

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Les procédures (1)

La différence avec les fonctions :

- **une procédure débute avec 'Procedure ’,**
- **elle ne retourne pas de résultat,**
- **les paramètres peuvent être de 3 modes :**

IN : le paramètre formel est une constante et il n'est possible que de lire la valeur du paramètre effectif associé.

IN OUT : le paramètre formel est une variable et permet à la fois la lecture et la mise à jour du paramètre effectif associé.

OUT : le paramètre formel est une variable et permet la mise à jour du paramètre associé.

BTD/GL/GL-LP

70

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Les procédures (2)

Déclaration :

```
procedure AJOUTER ( A,B : in INTEGER; C: out INTEGER) is
    begin
            C:= A+B;
    end AJOUTER;
```

Utilisation:

```
P,Q : INTEGER;  .....  AJOUTER(2+P,78,Q);
```

Déclaration :

```
procedure INCREMENTER (X: in out INTEGER) is
    begin
        X := X+1;
    end;
```

Utilisation :

```
I: INTEGER;  .....  INCREMENTER(I);
```

BTD/GL/GL-LP

71

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Les procédures (3)

Déclaration :

```
I: INTEGER ;
A: array (1..10) of INTEGER ;
procedure FARFELU (X: in out INTEGER) is
    begin
            I := I+1;
            X:=X+1;
    end FARFELU;
```

Utilisation :

```
A(5) := 1;
I:= 5;
FARFELU(A(I));
```

Quelle est la valeur de A(6) ?

BTD/GL/GL-LP

72

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Les procédures (4)

Paramètres par position et paramètres par défaut

Les paramètres n'ont pas besoin de figurer dans l'ordre de déclaration.

```
procedure SECOND_DEGRE (A,B,C : in REEL;
   RACINE_1, RACINE_2 : out REEL;
   OK : out BOOLEAN) is
   begin
        .......
     end ;
```

nous pouvons écrire :

```
SECOND_DEGRE ( A=>L, B=>M, RACINE_1=> P,
                RACINE_2=>Q, OK => STATUS);
```

ou

```
SECOND_DEGRE (L,M,N, RACINE_1 => P, RACINE_2 => Q,
              OK => STATUS);
```

BTD/GL/GL-LP

73

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Les procédures (5)

A la spécification de la procédure, on peut signaler les paramètres par défaut.

```
type VIN is (MUSCADET, CHAMPAGNE);
type STYLE is (GLACE,NATURE);
type EXTRA is (SIROP, ZESTE);
```

Les expressions de défaut peuvent être données dans la déclaration de la procédure :

```
• procedure KIR (BASE: VIN := MUSCADET;
                 COMMENT :STYLE := GLACE;
                 PLUS: EXTRA := SIROP);
```

des appels typiques peuvent être:

```
KIR (COMMENT => NATURE);          --  nommé
KIR(CHAMPAGNE, PLUS => ZESTE)     -- par position et nommé
KIR;                              -- par défaut
KIR(MUSCADET, NATURE);
```

BTD/GL/GL-LP

74

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Surcharge

Polymorphisme des procédures, fonctions et opérateurs

Possibilité de définir plusieurs procédures portant le même nom, ainsi que de donner de nouvelles significations pour les opérateurs préexistants, mais sans changer la syntaxe d'appel.

```
Procedure RECHERCHE (x : INTEGER; t : H_TABLE);
    begin .... end;
Procedure RECHERCHE (x : INTEGER; t : TABLE_CHAINEE);
    begin .... end;
Procedure RECHERCHE (x : INTEGER; t : TABLE_BINAIRE);
    begin .... end;
function "+" (A:VECTEUR) return REEL is
    RESULTAT : REEL :=0.0;
    begin
            for I in A'RANGE loop
                    RESULTAT := RESULTAT + A(I);
             end loop;
                    return RESULTAT ;
    end "+";
```

et nous pouvons alors écrire :

```
S:= +V;
```


CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Package (1)

Objectif : indépendance, composition d'éléments

– **PACKAGE : Son but est la séparation textuelle entre l'interface et l'implémentation.**

Comparons les deux structures suivantes:

1/ les variables HAUT et P ne sont pas protégées.

```
MAX : constant := 100;
P : array (1..MAX) of INTEGER;          -- déclaration d'une pile
HAUT : INTEGER range 0..MAX;
procedure EMPILER(X: INTEGER) is
    begin
      HAUT := HAUT+1;
      P(HAUT):= X;
    end;
function DEPILER(X: INTEGER) is
    begin
      HAUT := HAUT-1;
      return P(HAUT+1);
    end;
```

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Package (2)

2/ Le package encapsule la ressource PILE :

```
package  PILE is
   procedure EMPILER (X: INTEGER);      --Spécification
   function DEPILER return INTEGER;
end PILE;
package body PILE is                    -- corps
MAX : constant := 100;
P : array (1..MAX) of INTEGER;
HAUT : INTEGER range 0..MAX;
        procedure EMPILER(X: INTEGER) is
           begin
                    HAUT := HAUT+1;
                    P(HAUT):= X;
           end;
        function DEPILER(X: INTEGER) is
           begin
                    HAUT := HAUT-1;
                    return P(HAUT+1);
           end;
        begin
     HAUT := 0;        -- Initialisation
end PILE;
```

BTD/GL/GL-LP

77

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Package (3)

Différentes formes de packages :

1/ collection nommée de déclarations: exporte des objets et des types. N'exporte pas d'autres unités de programmes.

```
package CONSTANTES_METRIQUES is
    RAYON_EQUATORIAL : constant := 6378.145;     -- Km
    GRAVITATION : constant := 3.986_012e5 ;    -- Km3/sec2
.....
end CONSTANTES_METRIQUES ;

package  INFORMATION_DE _DATE is
    type NOM_DE_JOUR is
      (LUN, MAR, MER, JEU, VEN, SAM, DIM);
    type VALEUR_DE_JOUR  is range 1..31;
    type NOM_DE_MOIS is
      (JAN, FEV, MAR, AVR, MAI, JUN, JUL, AOU, SEP, OCT, NOV,
    DEC);
    type VALEUR-D_ANNEE is range 0..INTEGER'LAST;
end INFORMATION_DE _DATE ;
```

BTD/GL/GL-LP

78

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Package (4)

Différentes formes de packages :

2/ Groupement de sous programmes reliées sans objets rémanents.

```
package PRESENTER_DATE  is
    subtype TEXTE is STRING (1..20) ;
    type DATE is
      record
                JOUR: INTEGER range 1..31;
                MOIS : INTEGER range 1..12 ;
                ANNEE : INTEGER range 1901..2000;
      end record;
    procedure DATE_TEXTE_FR (d: DATE; out texte: TEXTE) ;
    procedure DATE_CHIFFRES_FR (d: DATE; out texte: TEXTE) ;
    procedure DATE_TEXTE_AN (d: DATE; out texte: TEXTE) ;
    procedure DATE_CHIFFRES_AN (d: DATE; out texte: TEXTE) ;
    procedure DATE_TEXTE_US (d: DATE; out texte: TEXTE) ;
    procedure DATE_CHIFFRES_US (d: DATE; out texte: TEXTE) ;
    procedure CONV_ TEXTE_DATE (texte: TEXTE; out d: DATE) ;
end PRESENTER_DATE ;
```

BTD/GL/GL-LP

79

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Package (5)

```
package TRANSFORMATION_2_D  is
        type COORDONNEE is record
          X: FLOAT;
          Y: FLOAT;
        end record;
        procedure TOURNER (POINT : in out COORDONNEE;
          ANGLE : in  FLOAT);
        procedure METTRE_A_L_ECHELLE
          (POINT : in out COORDONNEE;  X,Y: in  FLOAT);
        procedure TRANSLATER (POINT : in out COORDONNEE;
          X,Y : in  FLOAT);
    end TRANSFORMATION_2_D;
    package body TRANSFORMATION_2_D  is
        procedure TOURNER (POINT : in out COORDONNEE;
          ANGLE : in  FLOAT) is begin ... end ;
        procedure METTRE_A_L_ECHELLE
          (POINT : in out COORDONNEE;  X,Y: in  FLOAT) is
        begin ... end ;
        procedure TRANSLATER (POINT : in out COORDONNEE;
          X,Y : in  FLOAT) is begin ... end ;
    end TRANSFORMATION_2_D;
```

BTD/GL/GL-LP

80

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Package (6)

Différentes formes de packages :

3/ Machine abstraite: exporte les opérations et conserve les informations sur son état dans le corps.

```
package CLASSER_NOMS  is
    subtype NOM is STRING (1..70) ;
    procedure INITIALISER ;
    procedure AJOUTER_NOM (CE_NOM: in NOM) ;
    procedure SUPPRIMER_NOM (CE_NOM : in NOM) ;
    procedure IMPRIMER ;
end CLASSER_NOMS ;
package body CLASSER_NOMS is
    type ETAT is (PRESENT, ABSENT) ;
    function SUPERIEUR (NOM_1, NOM_2: in NOM) RETURN NOM
    is begin .....end;
    procedure AJOUTER_NOM (CE_NOM: in NOM) is
    begin  ... end;
    procedure SUPPRIMER_NOM (CE_NOM : in NOM) is
    begin  ... end;
    …...
    end CLASSER_NOMS ;
end INFORMATION_DE _DATE ;
```

BTD/GL/GL-LP 81

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

Langage ADA : Package (7)

Différentes formes de packages :

4/ Type abstrait de données: exporte des opérations et des types.
Ne conserve aucune information sur l'état dans le corps.

```
package  PILE is
    type PILE is private ;
    procedure EMPILER (P: in out PILE; X: INTEGER);
    procedure DEPILER (P: in out PILE; X: out INTEGER);
private
    type I_VECTEUR is array (1..10) of INTEGER ;
    type PILE is  record            espace: I_VECTEUR ;
                                    PTS: NATURAL := 0 ;
                    end record ;
end PILE;
package body PILE is
    procedure EMPILER (P: in out PILE; X: INTEGER) is
      begin  P.PTS:= P.PTS+1 ; P.ESPACE (P.PTS) := X ;
      end;
    procedure DEPILER (P: in out PILE; X: out INTEGER);
      begin   X := P.ESPACE (P.PTS) ;          P.PTS:= P.PTS-1 ;
    end;
end PILE;
```

BTD/GL/GL-LP 82

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA: Unités génériques (1)

- **Problème :**
  - **Dans la réalisation de systèmes informatiques nous trouvons toujours des sous-programmes ou des modules qui ont des objectifs similaires.**

  Tri d'un tableau d'entiers :

  - **type TABLEAU_ENTIER is array (NATURAL range <>) of INTEGER;**
  - **procedure TRIER (MON_TABLEAU : in out TABLEAU_ENTIER);**

  Tri d'un tableau de réels : même traitement sur des réels. Nous devons réécrire la procédure TRIER

  - **type TABLEAU_REEL is array (NATURAL range <>) of FLOAT;**
  - **procedure TRIER (MON_TABLEAU : in out TABLEAU_REEL);**

  - **Le composant du logiciel écrit en ADA peut être paramétré : il s'agit d'une unité générique.**

BTD/GL/GL-LP

83

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA: Unités génériques (2)

- **Soit le sous-programme qui permute deux éléments de type INTEGER .**

```
procedure PERMUTE_INTEGRE (PREMIER, SECOND: in out
        INTEGER ) is
                TEMPORAIRE : INTEGRE;
        begin
                TEMPORAIRE  := PREMIER;
                PREMIER := SECOND;
                 SECOND  := TEMPORAIRE;
   end PERMUTE_INTEGER;
```

- **Problème :** il n'est pas possible de permuter des éléments d'un autre type

BTD/GL/GL-LP

84

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA: Unités génériques (3)

- **Solution : on définit l'unité générique suivante**

```
generic
   type ELEMENT is private ;
   procedure PERMUTE (PREMIER, SECOND: in out ELEMENT );
```

  - **nous pouvant maintenant écrire le corps:**

```
procedure PERMUTE (PREMIER, SECOND: in out ELEMENT ) is
   TEMPORAIRE :ELEMENT;
   begin     TEMPORAIRE  := PREMIER;
             PREMIER := SECOND;
             SECOND  := TEMPORAIRE;
   end PERMUTE;
```

Instanciation : création d'une instance du paquetage ou du sous-programme génériques.

```
procedure PERMUTE_INTEGER is new PERMUTE(INTEGER);
procedure PERMUTE_FLOAT is
             new PERMUTE (ELEMENT=>FLOAT);
procedure PERMUTE_CHARACTER is
             new  PERMUTE(ELEMENT=>CHARACTER);
```

BTD/GL/GL-LP

85

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA: Unités génériques (4)

- **Paramètres génériques :**

Les différentes sortes de paramètres génériques sont possibles :

  - **Paramètres types génériques**
  - **Paramètres valeurs et objets génériques**
  - **Paramètres sous-programmes génériques**

BTD/GL/GL-LP

86

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA: Unités génériques (5)

1/ Paramètres types génériques :

Les formes de tous les paramètres types génériques sont :

type USAGE_GENERAL **is limited private;**

Le paramètre réel peut être de n'importe quel type, seule opération autorisée: instanciation.

type ELEMENT **is private ;**

Le paramètre réel peut être de n'importe quel type, avec comme opérations possibles: l'instanciation, l'affectation et le test d'(in) égalité.

type LIEN **is access** UN_OBJECT**;**

Le paramètre réel peut être de n'importe quel type accès désignant le même type objet.

type ENUMERATION **is (**<>**);**

Le paramètre réel peut être tout type discret (type entier et énumération).

type ELEMENT_ENTIER **is range** <> **;**

Le paramètre réel peut être tout type entier.

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA: Unités génériques (6)

type ELEMENT_FIXE **is delta** <>**;**

Le paramètre réel peut être tout type point fixe.

type ELEMENT_FLOTTANT **is digits** <>**;**

Le paramètre réel peut être tout type point flottant.

type TABLEAU _CONTRAINT **is array (**INDEX**) of** ELEMENT**;**

Le paramètre réel peut être tout type tableau contraint de mêmes dimensions, types d'index, et type de composant

type TABLEAU _NON_CONTRAINT **is array (**INDEX **range** <>**) of** ELEMENT**;**

Le paramètre réel peut être tout type tableau non contraint de mêmes dimensions, types d'index, et type de composant

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA: Unités génériques (7)

2/ Paramètres valeur et objet générique :

– **ADA permet également de définir des valeurs et des objets en tant que paramètres formels génériques.**

```
generic
   RANGEES : in INTEGER := 24;
   COLONNES : in INTEGER := 80;
   package TERMINAL is  .....
```

– **On pout créer plusieurs instances de ce paquetage générique:**

```
package MICRO_TERMINAL is new TERMINAL (24, 40);

package TERMINAL_VAX  is new TERMINAL (RANGEE => 66,
   COLONNES => 132);

package TERMINAL_DE_PROGRAMMATION  is new TERMINAL ;
```

BTD/GL/GL-LP

89

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA: Unités génériques (8)

3/ Paramètre sous-programme générique:

– **ADA permet également de passer des sous-programmes comme paramètres à une unité générique.**

```
generic
   RANGEES : in INTEGER := 24;
   COLONNES : in INTEGER := 80;
   with procedure ENVOYER (valeur : in CHARACTER);
   with procedure RECEVOIR (valeur : out CHARACTER);
package TERMINAL is  .....

A l'instanciation :

procedure MICRO_ENVOYER (valeur : in CHARACTER) is...
procedure MICRO_RECEVOIR (valeur : out CHARACTER) is...
package MICRO_TERMINAL is new
   TERMINAL (RANGEE =>24,
   COLONNES => 40,
   ENVOYER => MICRO_ENVOYER,
   RECEVOIR => MICRO_RECEVOIR);
```

BTD/GL/GL-LP

90

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA: Unités génériques (9)

Remarque :

le sous-programme effectif doit être compatible avec le sous-programme générique formel, c.à.d : il doit avoir des paramètres de même type, mode, ordre et contraintes que le sous-programme générique formel, et en plus une valeur retournée conforme dans le cas des fonctions.

```
generic
   type ELEM is private;
   with function "+" (A, B : ELEM) return ELEM;
function DOUBLER (R, "+" : ELEM) return ELEM is
   begin     return R+R ;   end ;
```

A l'instanciation il faut préciser le type remplaçant ELEM et la fonction remplaçant "+" associée au type effectif:

```
function AD_MAT (X, Y : MATRICE) return MATRICE.
function DEUX_FOIS is new DOUBLER (MATRICE, AD_MAT) ;
```

Les unités génériques permettent la paramétrisation des procédures, des paquetages et des fonctions, évitent la multiplication d'erreurs tout en donnant une meilleure lisibilité.

BTD/GL/GL-LP

91

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : entrées - sorties (1)

- **ADA fournit des paquetages prédéfinis pour trois niveaux d'entrées-sorties, qui sont:**
  - **Deux paquetages génériques pour tout type d'élément.**
    - **SEQUENTIAL_IO E/S sur fichiers séquentiels.**
    - **DIRECT_IO E/S sur fichiers à accès direct.**
  - **Deux paquetages de base :**
    - **TEXT_IO Paquetage pour les E/S de caractères.**
    - **LOW_LEVEL_IO Paquetage défini par l'implémentation, pour les E/S primitives.**

BTD/GL/GL-LP

92

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : entrées - sorties (2)

1/ SEQUENTIAL_IO et DIRECT_IO fournissent toutes les fonctions primitives nécessaires aux E/S d'un type d'élément donné.

les opérations définies :

CLOSE (fermer),
CREATE (créer),
DELETE (effacer),
OPEN (ouvrir) ,
READ (lire),
RESET (reprendre),
WRITE (écrire)
...

Ils comprennent également les fonctions suivantes :

END_OF_FILE (fin de fichier),
IS_OPEN (est-ouvert),
MODE,
NAME
...

BTD/GL/GL-LP

93

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : entrées - sorties (3)

– **DIRECT_IO définit en plus des procédures qui permettent d'accéder directement aux éléments**

SET-INDEX, SIZE, INDEX ...

– **Ces paquetages sont génériques, il faut donc les instancier avec un type particulier (via le paramètre générique** ELEMENT_TYPE**).**

```
with DIRECT_IO, SEQUENTIAL_IO;
procedure PRINCIPALE is
   package INTEGER_IO is
         new SEQUENTIAL_IO (TYPE_ELEMENT => INTEGER);
   package FLOAT_IO is
         new SEQUENTIAL_IO (TYPE_ELEMENT => FLOAT);
   begin ... end;
end PRINCIPALE;
```

Remarque: on peut également utiliser des types composites (tableaux, articles) du moment qu'ils sont contraints.

BTD/GL/GL-LP

94

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : entrées - sorties (4)

Structure et traitement de fichiers :

- Un fichier en ADA possède un type particulier (FILE_TYPE). Tous les éléments du fichier doivent appartenir au même type: ELEMENT_TYPE .
- Un fichier est associé à un périphérique physique comme un disque, un terminal, une imprimante ou une boite noire (un capteur, par exemple).

Le mode d'un fichier est déterminé lors de son ouverture, ou sa création.

IN_FILE	**(fichier d'entrée)**
OUT_FILE	**(fichier de sortie)**
INOUT_FILE	**(fichier d'E/S)**

Déclaration de fichiers:

```
FICHIER_INTEGER : INTEGER_IO.FILE_TYPE;
FICHIER_FLOAT : FLOAT_IO.FILE_TYPE;
```

BTD/GL/GL-LP

95

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : entrées - sorties (5)

- **Utilisation:**

```
- CREATE (FILE_FICHIER_INTEGER,
            MODE => OUT_FILE,
            NAME => "MON-FICHIER-DISQUE",
            FORM => "DISQUE");

- OPEN (FILE_FICHIER_FLOAT,
            MODE => IN_FILE,
            NAME => "MA_BOITE_NOIRE",
            FORM => "CAPTEUR");

- CLOSE (FICHIER_FIXED);
- DELETE (FICHIER_INTEGER);
- SET_INDEX (FICHIER_FIXED, TO => 137);
- READ (FILE => FICHIER_INTEGER, ITEM => MON_INTGER);
- WRITE (FILE =>FICHIER_FLOAT, ITEM => 3.14);
```

BTD/GL/GL-LP

96

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : entrées - sorties (6)

2/ Entrées-sorties de texte :

ADA définit un paquetage séparé, pour les données composées que de caractères ASCII.

les procédures offertes par TEXT_IO :

PUT : écrire
GET : lire

COL : renvoie la valeur de colonne courante.
SET_COL : change la colonne courante.

LINE : renvoie la valeur de ligne courante.
SET_LINE : change la ligne courante.

NEW_LINE : uniquement pour les fichiers OUT_FILE.
SKIP_LINE : uniquement pour les fichiers IN_FILE.

PAGE : renvoie le numéro de page courante.
NEW_PAGE: uniquement pour les fichiers OUT_FILE, saute SPACING pages.

```
1/  NEW-LINE (SPACING =>7);   SET_COL (TO => 26);
2/  PUT ( " hello , it is me");     PUT ( "  salut");     NEW_LINE ;     PUT (" bof ");
```

BTD/GL/GL-LP

97

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Exceptions (1)

- **Une exception désigne un événement qui provoque la suspension de l'exécution normale du programme. Cet événement peut être:**
  - Une erreur
  - Une condition exceptionnelle qui demande un traitement spécial.
- **Utilisation des exceptions :**
  - Abandonner l'exécution de l'unité.
  - Essayer l'opération de nouveau.
  - Utiliser une méthode différente.
  - Réparer la cause de l'erreur.

BTD/GL/GL-LP

98

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Exceptions (2)

- **ADA fournit un mécanisme en trois temps :**
  - déclaration d'une exception,
  - signalement d'une exception,
  - traitement d'une exception
- Une **déclaration d'exception** est similaire dans sa forme à une déclaration d'objets.

```
HORS_LIMITE1, HORS_LIMITE2: exception;
ERREUR_DE_PARITE : exception;
ERREUR_DISQUE : exception;
```

Il existe des exceptions prédéfinies :

- **NUMERIC_ERROR;**
- **PROGRAM_ERROR;**
- **STORAGE_ERROR;**
- **TASKING_ERROR;**

BTD/GL/GL-LP

99

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Exceptions (3)

- **Signalement d'une exception** a pour effet **de** suspendre le traitement séquentiel normal et passer le contrôle à un traitement-exception approprié. L'instruction **raise** permet de lever une exception :

```
raise HOR_LIMITE1 ; ou raise;
```

- **Traitement des exceptions** s'effectue dans un bloc particulier "exception".

```
        Declare NIVEAU_BAS_DU_LIQUIDE : exception ;
        begin ....
exception
        when NIVEAU_BAS_DU_LIQUIDE => OUVRIR_VANNE;
        DECLENCHER_ALARME;
        when NUMERIC_ERROR =>
            FERMER_VANNE;
         when others => ENREGISTRER_ERREUR;
end ;
```

Remarque : Après avoir terminé l'exécution du traitement d'exception le contrôle ne retourne pas où l'exécution a été levée; il continue simplement après la fin de l'unité où l'exception a été traitée.

BTD/GL/GL-LP

100

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Exceptions (4)

```
procedure REMPLIR_BOUILLOTTE (X : INTEGER) is
   DEBORDEMENT : exception     -- déclaration
   ....
   begin
   ....
   if X > MAXIMUM then raise DEBORDEMENT
                               -- signalement
   end if;
   ....
exception
   when DEBORDEMENT => ESSUYER
                               -- traitement
   when others => ....

end REMPLIR_BOUILLOTTE ;
```

BTD/GL/GL-LP

101

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Compilations séparées (1)

- **Intérêt : Découpage de gros programmes en plusieurs parties dans un souci de simplicité et d'efficacité :**
  - **les unités de compilation sont plus simples et plus faciles à gérer.**
  - **organisation du travail de plusieurs programmeurs travaillant sur le même projet.**
- **Unité de compilation est constituée d'une ou plusieurs unités de compilation, qui comprennent :**
  - **Déclaration générique.**
  - **Instanciations de génériques.**
  - **Déclaration de sous programmes.**
  - **Corps de sous programmes.**
  - **Déclaration de paquetages.**
  - **Corps de paquetages.**
  - **Sous Unité.**

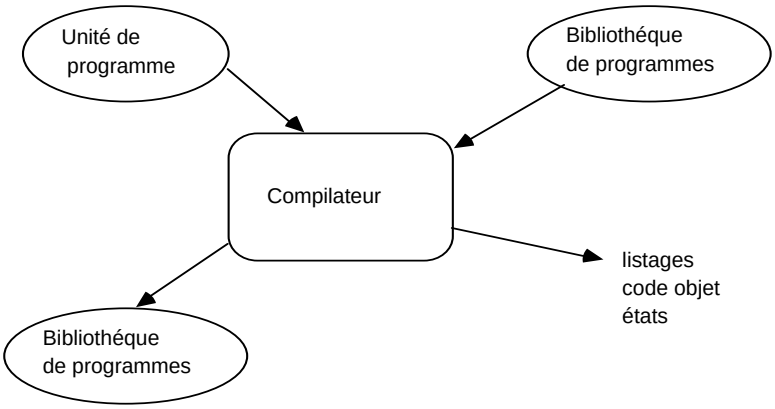
BTD/GL/GL-LP

102

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Compilations séparées (2)



BTD/GL/GL-LP

103

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Compilations séparées (3)

- **La clause WITH indique utilisation d'un package**
- **la clause SEPARATE définit une dépendance entre sous-unités de programme.**

```
procedure PRINCIPALE is
   package TRANSFORMATION is
            procedure DECRYPTER (MESSAGE : in out STRING);
            procedure ENCRYPTER (MESSAGE : in out STRING);
   end TRANSFORMATION ;
package body is separate ;
begin
--  corps de PRINCIPALE avec
      procedure DECRYPTER (MESSAGE : in out STRING) is separate;
end PRINCIPALE;
```

Nous pouvons compiler les sous-unités séparées en utilisant :

```
separate ( DECRYPTER)
procedure DECRYPTER (MESSAGE : in out STRING) is  ... end ;
```

BTD/GL/GL-LP

104

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Compilations séparées (4)

Remarque:

- **with** est utilisée dans le développement ascendant (on réutilise les composants déjà faits).
- **separate** est utilisée dans le développement descendant (on réalisera plus tard).

- **Ordre de compilation :**
  - Une unité ne peut être compilée que si les spécifications des packages, interfaces de l'unité déclarée par la clause with, l'ont été.
  - Les corps de ces packages (package body) peuvent être compilés ou recompilés après, séparément des spécifications.

BTD/GL/GL-LP

105

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

ADA : Compilations séparées (5)

- **Règles de recompilation :**
  - ➔ **Spécifications puis corps**
  - ➔ **Corps librement**
  - ➔ **des séparés (clause « separate ») librement**
  - ➔ **des réutilisations (clause « with ») d'abord, des modules qui réutilisent doivent être recompilés si ces modules changent**

BTD/GL/GL-LP

106