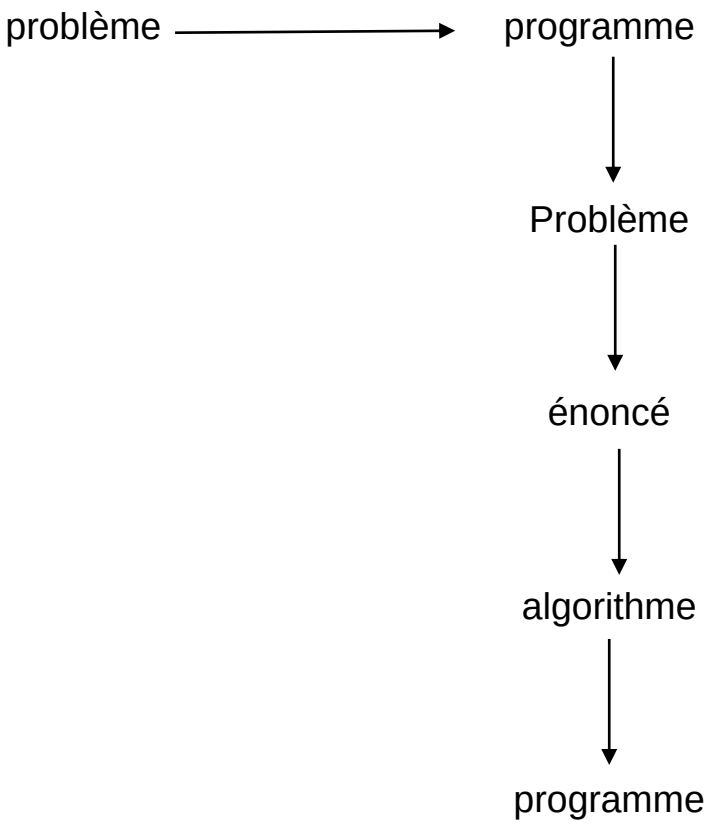
**CNAM, Université Du HAVRE,**
**Notes de cours A**lgorithme et programmation,
1994/1995, Enseignant: M. Nakechbandi

# Chapitre I

# Introduction à la programmation

## I.1/ Qu'est-ce que la programmation?

l'ensemble d'activités qui fait passer d'un problème à un programme




Pour faire une synthèse de cette activité on va remonter l'analyse à l'envers

## I.2/ Cadre formel:

**I.2.1/ Un programme** est un texte écrit en langage de programmation.

C'est donc un ensemble de phrases qu'il est possible à un être humain de rédiger, et qui décrit sans ambiguïté les actions exécutables par un ordinateur.

**I.2.2/ Notion d'algorithme** Démarche logique pour résoudre un problème;

Plus formellement: un algorithme est la description d'une **action** complexe, au moyen d'actions élémentaires, et de règles de composition de ces actions.
*Cette action complexe doit être exécutable par un processeur: si on sait exécuter les actions élémentaires, et lorsqu'on sait appliquer les règles de composition.*

Pour préciser la notion d'action il nous faut definir 2 Termes:

Processeur, et l'environnement.

Un processeur est une entité capable de comprendre un énoncé et d'exécuter le travail indiqué par cet énoncé.

Un environnement: est l'ensemble des ustensiles nécessaire à l'exécution d'un travail

Une action: un événement modifiant un environnement cette modification est défini comme une transition entre un état initial de l'univers et un état final;

spécification d'une action:

{état initial}

Action

{état final}

Notations: {P} A {Q}
{P} sera appelé précondition, et {Q} postecondition

Une action est élémentaire si l'énoncé de cette action est à lui seul suffisant pour que le processeur puisse l'exécuter sans information supplémentaire.

Exemple: Sur tout ordinateur on sais faire les actions suivantes:

- transfère de données (mémoire <---> mémoire)
- opération arithmétique
- opération comparaison
- opération d'entrée/sortie (mémoire <---> périphérique)

Un appel de procédure peut être considéré comme une action élémentaire.

## I.3/ Retour aux cycles de vie d'un programme

Problème ---> Enoncé ---> algorithme
---> programme --->Mise en route(et maintenance)

**problème ---> Enoncé**

Spécifier un problème c'est définir:

. les données,

. les résultats,

. et les relations entre données et résultats

Exemple 1/ calculer $y = x^n$

2/ calculer $y = \sin(x)$

**énoncé ---> Algorithme**

Eprimer les étapes permettant d'obtenir les résultats à partir de
données en se basant sur les relations entre les données et les résultats.

Exemple: Dans l'exemple précédent

algorithme 1 y = 1; répéter n fois y = y * x;

algorithme 2 f(x, m)= 1 si m= 0 ;

= x * f(x,m-1)  si m > 0;

affecter f(x,n) à y

Ce passage passe par plusieurs phases d'affinement progressif.

**Algorithme ---> Programme**:

Coder un algorithme dans un langage informatique.

A ce stade, l'algorithme doit être composer par des étapes qui sont toute exprimable formellement dans la grammaire d'un langage de programmation.

Un tel algorithme est appelé  algorithme détaillé

**Mise en route et maintenance**

Utiles de génie logiciel

## I.4/ Notion d'analyse descendante

Etant donné un énoncé non primitif T l'analyse descendante de T consiste à trouver une décomposition {T1,T2,... Tn} qui soit une suite d'énoncés dont l'exécution réalise T.

Pour chaque énoncé $T_k$ deux cas sont possibles:

- Soit $T_k$ est action élémentaire et on arrête l'analyse descendante pour $T_k$
- Soit $T_k$ n'est pas une action primitive et on décompose à nouveau $T_k$

Ce principe peut être applique à nouveau, jusqu'à l'obtention des actions fils ou petits fils qui seront toutes des actions élémentaires.

On appel ceci le passage d'un niveau d'abstraction à un niveau plus détaillé.

## I.5/ Problème d'expression d'un algorithme

(pseudo langage ou pseudo code)

### I.5.1/ Nécessité:

Au départ on a utilisé l'organigramme.

Cette manière d'expression :

- elle est très lord,
- elle est maintenant inadapté aux possibilités des langages de programmation modernes, qui sont très structurés et qui s'approchent de plus en plus de langages humains.

Caractéristique d'un pseudo code:

- exprimer l'analyse descendante

- proche du langage humain,
- simple (de point du vue syntaxique)

## I.5.2/ Grammaire pour un pseudo langage d'expression d'algorithme:

Un pseudo programme est un environnement + un ensemble d'actions composés selon 3 règles de composition: la composition séquentielle, la structure de choix, et la structure itérative.

- **L'environnement**: est un ensemble de variables appartenant à des types abstraits;
  un type abstrait est un ensemble de valeurs sur lesquelles on admet certains nombre d'opérations.

Exemple X, Y sont des matrices;
Z est un graphe;

On admet les notations suivantes:

X,Y : matrice;
Z : Graphe;

Dans un environnement on distingue deux genres de variables:
ceux qui seront consultées seulement, et ceux qui peuvent être modifier.

**Une action dans un pseudo programme**:
- ou bien, c'est un énoncé
- ou bien, c' est une action élémentaire.

On admet en plus l'existant d'une action nul.

- **Structure de composition**:

- **Structure séquentiel** de composition:

(Toute suite d'actions successives est une nouvelle action)

action1; action2; action3... $action_n$;

- **Structure de choix**:

if condition alors action1 sinon action2;

- **Structure itérative**:

```
tant que  condition  faire
                    action;
                refaire;
```

Exemple 1: Soit A, B, C, D, E, un ensemble d'action.

La structure suivant est un pseudo programme:

```
Environnement:  i; j : entier;
A;
i := 1;
tant que   i < 10 faire
        B;
         calculer la racine carré de j;
        si  j < 5  alors C;
                sinon D;
         i := i + 1;
         E;
 refaire
```

## **Exemple:** (on trouve en annexe I le code équivalent en ADA)

***Problème***

*Dans une entreprise, chaque succursale possède un fichier des articles qu'elle a en stock. Chaque article est référencé par un code et par la quantité disponible dans la succursale concernée.*
*On décide de fusionner les stocks des deux succursales. Ces succursales utilisent utilise les mêmes codes pour les mêmes articles, et on suppose que chacun des fichier des deux succursales est trié par ordre croissant du code d'article.*

*Ecrire un programme qui va fusionner les deux fichiers.*

**Algorithme**

**1e niveau d'abstraction (algoritme général)**

```
première lecture dans stock1
première lecture dans stock2
si stock1 est non vide alors falg1 <-- vrai
si stock2 est non vide alors flag 2<-- vrai

tq  (flag1 et flag2) faire
    si (article1.c = article2.c) alors  traiter _ égalité;
    sinon   si  (article1.c < article2.c) alors traiter_cas_article1_inf_article2;
                                          sinon  traiter_article1_sup_article2;
                                    finsi;
   finsi;
refaire;
si  flag1  alors copier_reste_stock1;
           sinon  si  flag2  alors  copier_reste_stock2;  finsi ;
finsi ;
fermer les fichier stock1, stock2, stock3
```

**2ème niveau d'abstraction de détail**

```
            /*   traiter _ égalité  */

                article3.c <--- article1.c;
                article3.q  <---  article1.q + article2.q;
                écrire (stock3, article3);
                si non fin de fichier(stock1) alors
                  lire (stock1, article1);
                sinon
                  flag1  <---  false;
                finsi ;
                si non fin de fichier (stock2) alors
                  lire (stock2, article2);
                sinon
                  flag2  <---  false;
                 finsi ;

/*  traiter_cas_article1_sup_article2  */

                écrire(stock3, article2);
                si non fin de fichier (stock2) alors
                  lire(stock2, article2);
                sinon
                  flag2  <---  false;
```

```
        finsi ;


  /* traiter_cas_article1_inf_article2 */

          écrire(stock3, article1);
         si non fin de fichier(stock1) alors
           lire (stock1, article1);
         sinon
           flag1  <---  false;
          finsi ;
        end;


  /*  copier_le reste de stock1 dans stock3   */

écrire(stock3, article1);
tq non fin de fichier (stock1) faire
   lire(stock1, article1);
   écrire(stock3, article1);
refaire;


  /*  copier_reste_stock2 dans stock3  */

         écrire(stock3, article2);
          tq non fin de fichier (stock2)  faire
            lire(stock2, article2);
            écrire(stock3, article2);
          refaire;
```

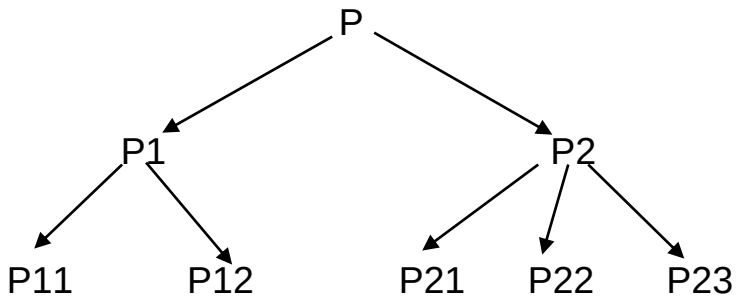
# Les sous programmes

Introduction

Un algorithme ne devait pas dépasser la longueur d'une page.

Pour parvenir, on sait que pour résoudre un problème, il faut découper l'algorithme en morceaux, ces morceaux seront assemblés, selon un schéma opératoire, pour constituer une seule action résolvant le problème d'origine.

Plus formellement: Rappelons que pour résoudre un problème, on recommande l'analyse descendant;




**Définition:** L'action qui résout un sous problème sera appelée action nommée.

Cette action est donc un algorithme qui peut être utilisé dans un autre algorithme, nous soulignons son nom, afin de bien mettre en valeur le fait que nous désirons exécuter à cet instant son code, sa description doit être faite en-dehors des limites de l'algorithme qui l'utilise.

Exemple:
Enoncé: Ecrire un algo. permettant, à partir de la saisie de trois nombre, de rechercher le minimun ou le maximun, au choix de l'utilisateur.

On peut envisager l'algorithme géneral suivant:

```
saisie des trois nombres a,b,c;
affichage de menu et saisie de choix;
si choix  = ''1'' alors calculer le minimum de a,b,c  ; fsi;
si choix  = ''2'' alors calculer le maximum de  a,b,c ;  fsi;
afficher le resultat;
```

On va nommer les actions suivantes:

saisie( ; a,b,c) : C'est l'action qui saisie les trois nombres
menu( ; choix) : L'action qui réalise l'affichage de menu et
saisie de choix;
calcul_min (a,b,c ; min): L'action qui calcule le minimum;
calcul_max(a,b,c ; max) : L'action qui calcule le Maximum;
afficher(a,b,c,choix ; min, max): L'action qui affichera le resultat

L'algorithme générale pécedent peut donc s'exprimer comme suit:

```
saisie( ; a,b,c);
menu( ; choix);
si choix  = min alors calcul_min (a,b,c ; min); fsi;
si choix  = max alors calcul_max(a,b,c ; max) ; fsi;
```

```
algorithme saisie(;a,b,c)

 ecrire (''entrer les toix nombres:'');
 lire(a); lire(b); lire(c);
```

```
algorithme menu(;choix)

 ecrire ('' voulez-vous : '');
 ecrire ('' 1 - chercher le minimum taperz 1'');
 ecrire ('' 2 - chercher le maxumum taper 2'');
```

```
algorithme calcul_max (a,b,c ; max)

  max <-- a;
  si max < b alor max <-- b fsi;
  si max <  c alors max <-- c fsi;
```

```
algorithme calcul_min (a,b,c ; min)

   min <-- a ;
   si min > b alors min <-- b; fsi;
   si min > c alrs min <-- c; fsi;
```

```
algorithme afficher(a,b,c,choix ; min, max):

si choix = ‘‘1 ’’ alors ecrire (‘‘le minimum de’’, a,b,c, ‘’est ‘’, min);
si choix = ‘’2’’ alors ecrire (‘’le maximum de’’,a,b,c,’’ est ‘’, max);
```

Exercice:

Enoncé du problème : Nous voulons écrire l’algorithme permettant de calculer la différence, exprimer en nombre de jours, entre deux datzes de la même annee, saisi au clavier sous le fromat: jj/mm/aa ou les variables  jj, mm, aa sont des entiers.

Idée : Utiliser le principe du ‘’quanrtième’’ d’une date, qui correspond au numéro sequentiel du jour dans l’année.

Dans un langage de programmation la implantation d’une action nommée est fait par la notion de procédure, cet notion permet donc:

- d’isoler et de nommer une suit d’instructions,
- d’exécuter cet suit d’instruction par une simple appel de sur nom, suivi éventuellement par des paramètres.

# CNAM LE HAVRE, U.V. Algorithme et programmation,TD1, 18-10-94,

## Enseignant: M. Nakechbandi

**Objectif:** Action, environnement, processeur, et programmation structurée.

Soit X un ensemble d'éléments sur lesquels on connaît une relation d'ordre <, soit encore S l'ensemble des suites de n éléments prisent dans X. Sur X et S on a un processeur qui sait réaliser les fonctions (opérations) suivantes:

- EGALE est une fonction de X*X ---> {vrai, faux}, EGALE(a,b)= vrai si a= b;
= faux sinon;

- MAX est une fonction de X*X ---> X telle que MAX(a,b)= le maximum de {a,b}

- ACCES S*N ---> X | ACCES(s,i) est le ième élément de s.
N = {1..n}; La notation {1..n} désigne l'ensemble {1,2,3,... ,n}

- EST_DERNIER S*N ---> {vrai,faux} | EST_DERNIER(s,i) est un prédicat qui veut vrai si i pointe sur le dernier élément de S, faux si non.

- échange S*N*N ---> S | échange(s,i,j) est la suite qui résulte de s en échangeant l'élément de rang i avec celui de rang j. N = {1..n}

En plus des opérations précédentes, notre processeur sait exécuter les opérations élémentaires classiques sur les nombres entiers (affectation, addition, comparaison, multiplication, soustraction).

**Exercice** 1/ Ecrire un pseudo programme qui détermine le plus grand l'élément d'une suite s.

**Exercice** 2/ Ecrire un pseudo programme qui calcule la taille d'une suite s

**Exercice** 3/ Soit a un élément de X, s une suite dans X. Ecrire un programme qui détermine si oui ou non a appartient à s, si oui ce programme doit renvoyer le rang de a dans s, si non il renvoie 0.

**Exercice** 4/ Soient s,r deux suites écrire un programme qui teste si ces deux suites sont identiques.

**Exercice** 5/ A notre environnement on va ajouter la fonction suivante:
valeur X ---> {1,2} qui associe à chaque élément de X une évaluation numérique de 1 ou 2. Soit s une suite, écrire un programme qui réarrange les éléments de s de telle manière que tout élément d'une évaluation qui est égale à 1 soient avant les éléments dont les évaluations sont égale à 2.

**Exercice** 6/ (généralisation de l'Exercice 5) A notre environnement on va ajouter la fonction suivante:
classer X ---> {1,2,3} qui classe les éléments de X en 3 catégories 1,2, ou 3.
Soit s une suite, écrire un programme qui réarrange les éléments de s de telle manière que tous les éléments qui appartiennent à la classe 1 soient avant les éléments de la classe 2, et ceux de la classe 3 soient après les éléments de la classe 2.

## 1.6/ Notion de Type abstrait:

**1.6.1 Définition du type:** on appel type un ensemble d'élément sur lesquels sont définies un ensemble d'opérations

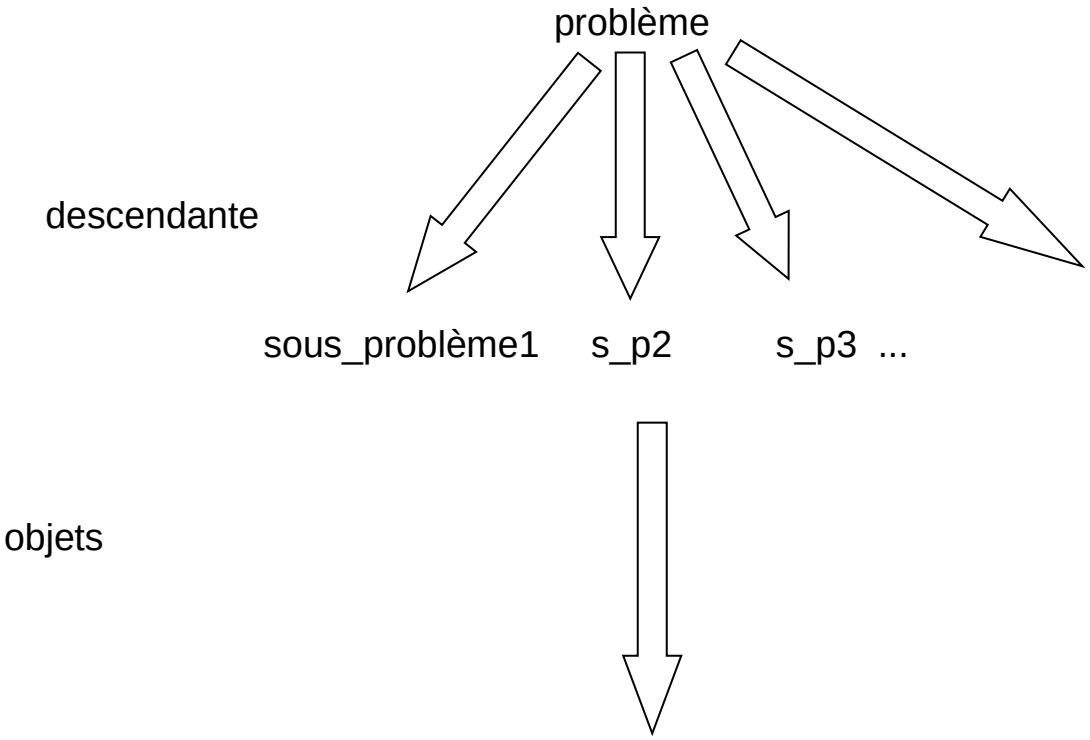
Exemple: les entiers, les caractères, les réels, les booléens

Entier : ensemble = {-32768 .. 32767}
opérations = + , -, *, div
comparaison.

caractère : ensemble = {A..Z} + {a..z} + {0,1,..,9} etc,...
opération = comparaison.

Exercice : Déterminer les caractéristique du type Réel , booléen.
ensemble, opérations

## 1.6.2/ Analyse ascendante



### 1.6.3/ Type abstrait: Description algébrique

La notion de type abstrait est un moyen formel pour identifier
un type (en d'hors du langage de programmation)

une spécification algébrique de type abstrait a la forme suivante:

**Spécification**

**type** *nom_du_type* ( paramètres formels génériques)

**opérations**

liste des fonctions et des opérations
à chacune d'elles est associée:

- une liste des paramètres données de la fonction (en entrées),
  son domaine.

- une liste des paramètres résultats de la fonction (en sortie),
  son codomaine.
  opération : domaine -> codomaine

**axiomes**

liste d'equations portant sur les opérations.

**restrictions**

liste de préconditions associées aux opérations, ces preconditions doivent être satisfaites pour que les opérations soient applicables.

**Exemple:**

type pile (X est un type de base)

opérations

vide : pile -> booléen

nouvelle -> pile

empiler : pile * X -> pile

sommet : pile -> X

Axiomes

soit, x:X, p:pile

1/ sommet (empiler(x,p)) = x

2/ dépiler (empiler (x,p)) = p

3/ vide (nouvelle) est vrai

4/ vide(empiler(x,p)) est faux

restrictions

prémonition pour utiliser dépiler (p) : la pile p doit pas être vide

vide(p) est faix

prémonition pour utiliser sommet (p) : la pile p doit pas être vide

vide(p) est faix

**CNAM LE HAVRE, U.V. Algorithme et programmation (programmation impérative), TD2, 15-11-94, Enseignant: M. Nakechbandi**

**Objectif:** Type abstrait de données.

**Problème 1**: Utilisation du Type abstrait PILE (Défini en cours)

Un garage étroit fonction comme une pile:

- on peut entrer une nouvelle voiture, et la placer au sommet de la pile,
- on peut sortir la voiture qui se trouve au somme de la pile,

1/ Ecrire un programme structuré, utilisant les primitives sur la pile, pour que 6 voiture entrées dans l'ordre 1,2,3,4,5,6 ressortent dans l'ordre :

a) 6 5 4 3 2 1
b) 1 2 3 4 5 6
c) 3 2 5 6 4 1
d) 1 5 4 6 2 3

2/ Dans la question précédent, quelles sont toutes les sorties possibles. Y-a-t-ils des ordres de sortie impossibles ?

**Problème 2:** Définition d'un Type abstrait TABLEAU qui correspond au structure de tableau d'une dimension.

On doit spécifier donc les paramètres formels, les opérations, les axiomes, et les restrictions.

# CHAPITRE II

# LE LANGAGE ADA

## II.1 Introduction

Le langage ada est né suite d'un concours internationale
Lancé par dod département of défense des etats unis

C'était une equipe français qui a gagné le concours. (1978)

Retenons quelques dates:

- 1974, lancement du concours
- 1977, 4 langages sélectionnés
- 1978, le langage français était retenu
- 1979, 1 ér manuel de référence,
- 1983, définition du 1 ère standard qui doit être examiné chaque 10 ans.
- 1985 première compilateur validé.

**Dans le cahier des charges:**

- Le langage doit supporté:
- la modularité,
- le type abstrait,
- la gestion des taches parallèle,
- la portabilité des application,
- la sécurité,
- la lisibilité,
- l'efficacité,

On a précisé aussi le besoin d'un langage qui sert à la fois:

pour les application de gestion,

pour les application scientifique,

et pour les application temps réel (calculateur embarqué)

ADA est une marque déposé par le gouvernement des états Unis et que tout compilateur ada doit être certifier par un organisme officiel qui contrôle la conformité à la norme.

Ceci est réalisé en passant le compilateur par une batterie de programmes de tests officielles.

Un compilateur n'ayant pas satisfait à cette batterie de tests n'aura pas qualité de compilateur ada normalisé.

Par contre celui qui réussit sera appelé compilateur ada normalisé.

Normalement la majorité des concept usuels qu'on rencontre dans les langages classique tel que pascal, cobol, c fortran ont leur équivalant en ADA.

## II.2 La structure d'un programme ada

Un programme ADA est un ensemble d'unités.

Une unité peut être:
une fonction
une procédure
ou un module

Unité principale: C'est la procédure qui doit être exécuté lors du lancement du fichier exécutable correspondant au programme.

*Exemple d'autres langages:*

- en PASCAL, c'est le mot clé program
- en C c'est le mot clé main

En ada c'est la procédure d'en-tête qui représente l'unité principal.
Dans un programme destiné à être exécuter, il y a toujours une unité principale.

**Exemple:**

```
With ada_io; use ada_io;
procedure toto is
    put_line(‘’ mon premier programme ‘’);
end;
```

# Exemple introductif d'un programme ada

Supposons qu'on a développé un paquetage pour la gestion de la pile de voitures (une voiture est repérée par un nombre entier). Supposons maintenant que une voiture est repérée par son numéro immatriculation, qui peut être du type suivant:

```
tvoiture is record
                    n  : integer range 0..9999;
                    m : string(1..2)
                    d  :  integer range 1..100;
                 end recodr;
Pour gérer une pile de voiture il faut réécrire toutes les procédure.
Avec la Générecité de ADA on peut écrire une procédure avec des paramètres génériques

generic
      type element is private;
package g_pile is
      taille : constant integer := 50;
      subtype indice is integer range 0..taille;
       type integer;
       type pile is record
            corps :  array(indice) of  element;
              tete  :  indice;
       end record;
      procedure nouvelle (p: out pile) is
      procedure empiler(x : element; p: in out pile) is
      function vide (p: in  pile) return boolean is
      procedure sommet(p: in pile; x: out element) is
      procedure depiler(p: in out pile) is
end tapile;
with ada_io ; use ada_io;

package body tapile is
        procedure nouvelle (p: out pile) is
        begin  p.tete := 0; end;
        function vide (p: in  pile) return boolean is
         begin
          if p.tete = 0 then put(" pile vide "); return false;
                        else return true;
          end if;
         end;
        procedure empiler(x : element; p: in out pile) is
        begin
          if p.tete = taille then
            put(" erreur ");
          else
            p.tete := p.tete + 1; p.corps(p.tete) := x;
          end if;
         end;
        procedure depiler(p: in out pile) is
        begin
          if p.tete = 0 then put(" erreur ");
                        else p.tete := p.tete - 1;
          end if;
         end;
        procedure sommet(p: in pile; x: out element) is
        begin   x:= p.corps(p.tete) ;   end;
end  g_pile;

WITH  g_pile;
procedure teste_pile_g  is
tvoiture is record
                    n  : integer range 0..9999;
                    m : string(1..2)
                    d  :  integer range 1..100;
                 end recodr;
```

```
package g_pile_voiture is new g_pile(tvoiture);
use g_pile_voiture;
p :pile;
x, v :tvoiture;
ile_g
begin
    nouvelle (p);
    v.n := 1111;
    v.m := ''PQ'';
    v.d  := 12;
    empiler(v, p);
    sommet (p, x);
     depiler(p);
     put_line(...  x.n ... x.m... x.d);
end;
```

## II.3/ La notion de Bibliothèque:

Les unités peuvent être compilées séparément

Toute nouvelle unité va ajouter des nouvelles fonctionnalités à l'environnement.

L'ensemble de ces fonctionnalités seront rassemblées dans la bibliothèque sous forme de programmes objets issu des compilations séparées.
Exemple:

```
--      Cette procédure affiche les nombres de  1  à  x       --
with ada_io; use ada_io;
procedure simpl  is
x :integer;
begin
  new_line;
  put("donner x ");
  get(x);
  for i in 1..x loop
        put(i); put("  ");
  end loop;
  new_line;
end;
```

```
with ada_io; use ada_io;
procedure plus_s  is
begin
  put(" encore plus simple qui utilise la procédure simpl"); new_line;
  simpl;
end;
```

**EXEMPLE DE PROGRAMME AVEC BIBLIOTHEQUE:** LE TYPE ABSTRAIT PILE

```
Package tapile is
```

```
taille : constant integer := 50;
 subtype voiture is integer range 1..taille;
 subtype indice is integer range 0..taille;
 type tab is array(indice) of integer;
 type pile is record
      corps : tab;   tete  :  indice;
 end record;
procedure nouvelle (p: out pile) ;
procedure empiler(x : voiture; p: in out pile) ;
function vide (p: in  pile) return boolean ;
procedure sommet(p: in pile; x: out voiture) ;
procedure depiler(p: in out pile) ;
end tapile;
with ada_io ; use ada_io;
package body tapile is
procedure nouvelle (p: out pile) is
begin  p.tete := 0; end;
function vide (p: in  pile) return boolean is
 begin
  if p.tete = 0 then put(" pile vide "); return false;
                else return true;
  end if;
 end;
procedure empiler(x : voiture; p: in out pile) is
begin
  if p.tete = taille then
    put(" erreur ");
  else
    p.tete := p.tete + 1;
    p.corps(p.tete) := x;
  end if;
 end;
procedure depiler(p: in out pile) is
begin
    if p.tete = 0 then put(" erreur ");     else  p.tete := p.tete - 1;
    end if;
 end;
procedure sommet(p: in pile; x: out voiture) is
begin   x:= p.corps(p.tete) ;   end;
end  tapile;

with tapile ; use tapile;
with ada_io ; use ada_io;

procedure essai is
 p : pile;
 i, a: voiture;
begin
  nouvelle (p);
  new_line; put_line("  6 voitures seront empilees: "); new_line;
   i := 1  ;
  while i <= 6 loop
        put(" On va empiler la voiture numero: ");put(i); new_line;
        empiler(i,p);
```

```
        i := i + 1;
  end loop;

  new_line; put_line(" L etat de sortie est: "); new_line;

  while vide (p) loop
       sommet(p,a);
       put(a); new_line;
       depiler(p);
  end loop;
  new_line; put_line(" Fin du programme Garage "); new_line;
end;
```

## II.4/ Eléments lexiques

Le jeu de caractères:

Les délimitateurs longs:

Les  mots réservés:

les séparateurs:

Les commentaires:

Les caractères littéraux:

Les littéraux numériques:

les identificateurs:

## II.5 / Les Types prédifinis en ADA

entier:  integer  natural  positive
réel  :  float
énumératif:  character  boolean
II.6/ Les Instructions en ADA

La structure séquentielle:
c'est un ensemble d'instructions,  ces instructions serons séparées par « ; », et éventuellement  l'ensemble sera encadrées par  begin end;

la structure de choix:

```
if condition then   suites d'instructions
                    else une autre suite d'instruction
end if  ;
```

La structure itérative  (la boucle)

```
loop
   suite d'instruction;
   exit when condition;
   une autre suite d'instructions;
end loop;
```

```
while condition   loop
  suite d'instructions;
end loop;
```

## Les entrées sorties:

ces instructions se trouvent dans une bibliothèque d'entrées sorties

**1/ affichage;**

```
put('caractère');
put('' chaîne de caractères'');
put(valeur numérique);
put-line;
new_line;
```

**2/ Lecture**

```
get(variable_à_récupérer_au_clavier);
```

## II.7/ Les types en ada

### II.7.1/ Introduction

Un objet qqc qui existe: variable, procédure, tache.

Règle générale: Chaque objet en ADA doit être déclaré.

La déclaration de chaque objet spécifie son type.

Tout opération sur un objet doit en préserver le type;

Autrement dit pas deconversion implicite. Par contre il existe des opérateurs pour la conversion explicite.

## II.7.2/ Type prédéfini

integer; (short_integer, long_ineger)
float; (short_float, long_float)
boolean;
character;
string.

**Déclaration d'un objet:**

somme : integer := 0;
pi : constant float := 3.14;

## II.7.3/ Les conversions de types

Deux objets sont considérés de même type s'ils sont associé au même nom de type.

**Exemple**:

type couleur is (rouge, jaune, gris);
type color is (rouge, jaune, gris);

a,c: couleur;
b,d: color;

a, b sont de type différent.
b,d sont du même type.

Exemple d'une conversion:

longueur : integer;

largeur :float;
surface : **short_integer**;
aire : integer;

surface := **short_integer** ( float (longueur) * largeur) ;

## II.7.4/ Sous type

un sous type est un autre type défini sur un sous ensemble de valeurs d'un type (dite type de base) et possède le même ensemble d'opérations. il se défini en ada par le mot clé **SUBTYPE .**

un sous type ne crée pas un nouveau type il sert à faire des restrictions sur un type donné.

**Subtype** identificateur **is** indication_de_sous_type;

marque de sous type [ contrainte]

type | sous type intervalle |

**Exemple**:

subtype pepits_entier is integer range -128.. +127;

subtype mes_jours is jour_sem range mardi..jeudi;

subtype positive is integer range 0.. integer'last;

*Il est permis de mélanger, dans une expression sans faire de conversion, des objets appartenant à des sous-types différents issus du même type de base.*

## II.7.5/ Dérivation (l'instruction new)

**Exemple introductif**:

```
type mesure is new float
type distance is  new mesure;
type température is  new mesure;
```

si on a

```
x : distance;
y : température;
z: mesure;
```

il est impossible de faire  z :=  x + Y;

par contre, il possible de faire:  z  := mesure(x) * mesure(y);

un type dérivé possède le même ensemble de valeurs que son type parent.

Dérivation **sans contraint**

Dérivation **avec contraint**:

```
type ma_valeur is new integer range 1..100;
```

## II.7.6/ Les types scalaires:

### Propriétés communes

Les opérations suivantes sont définies pour tout type scalaire:

- égale =
- différent /=
- comparaison < , > , <= , >=

- les testes d'appartenance à un intervalle: **IN , NOT IN**
  Les deux opérateurs **first** et **last** sont définis:

Les types scalaires correspondent aux classes suivantes:

- type numérique
  type entier,
  type réel: point flottant, point fixe,

- type discret
  type integer,
  type boolean: not, or, and xor
  type caractère
  type énumératif

    Opérations (attribut):
        Image,
        value,
        pos,
        pred,
        succ,
        image

- exemples:

  integer'first, integer'last;
  couleur'first, couleur'last

**Cnam, le Havre, 17/1/1995, UV Algorithme et programmation.**
**Enseignant: NAKECHBANDI**. **Thème**: TP/TD3 Les Types en ADA.

**Exercice 1/** Déclarez deux variables booléennes flag ,et teste. La valeur initiale de falg est faux, et celle de teste est vrai.

**Exercice 2/**

a.. Déclarez un constant N_MAX qui correspond au nombre maximal d' étudiants qui peuvent s'inscrire au CNAM du Havre.

b. Déclarer un constant N_MAX_UV qui correspond au nombre maximal d'étudiant de d'un UV.

c. Préparez un type T_INSCRIT pour gérer le nombre des étudiants inscrit au CNAM du Havre

d. Utilisez le type précèdent pour faire un sous type T_INSCRIT_UV , qui correspond au nombre des étudiants inscrit dans une unité de valeur au CNAM du Havre

e. Déclarez: Deux variables X , Y de type T_INSCRIT . Une variable Z de type T_INSCRIT_UV dont la valeur initiale est 0 .

f. Déclarer un nouveau type entier T_NOTE pour la gestion des notes (une note maximale est 20.

**Exercice** 3/ Déclarer et initialiser une variable N de ce type. g. Soit M une variable de type entier. Les affectations suivantes sont-elles possibles: (Expliquez)

```
N_MAX := 30;
FLAG := 0;
 X := FLAG;
 X := M;
  N := M;
 Y :=  X;
 N := 10;
 X := 200;
 Y := N + 1;
 X := Y * 2
 X := Y / 2;
 M := Y;
  M := N;
```

h. Faire le nécessaire pour que l'opération suivante soit acceptable: M := X + N;

**Rappel** de cours sur les nombres réels et les nombres en virgule flottante: Un réel peut être déclarer en virgule fixe ou en virgule flottante. Pour décrire un nombre réel il faut décrire son modèle, ceci revient à décrire:
l'intervalle: **A : FLOAT range 0.1 .. 1.0;**
le nombre de digits significatifs: **T : DIGITS 6 range 0.0 .. 1.0;**
le pas de discritisation (échelle): **Type p_fix is DELTA 0.01 range 0.0 .. 1000000.0;**
X , Y , Z: p_fix ; Dans ce cas, l'opération de la division (/) n'est acceptable que après une conversion:

```
Z := p_fix( X / Y);
```

3/ a. Décrire un type flottant T_ANGLE qui varie de 0.0 à 360.0

b. Trouver plusieurs méthodes pour déclarer une variable ANGLE qui prend ses valeurs dans l'intervalle
(0.0, 90.0).

**Exercice** 4/
a. Déclarer une variable SOMME de type réel qui ne peut varier que entre 0.0 et 1000.0 avec une précision de deux digit après la virgule .
b. Donner l'instruction ada pour affecter la valeur (1.4 * 100) / 0.3) à la variable somme.

**Exercice** 5/ On suppose qu'on a les déclarations suivantes:

```
TYPE couleur IS (black, white, red, green, bleu);
TYPE autre_couleur in New couleur;
monochrome : couleur range black.. white;
c := black;
```

a. Les affectations suivantes sont-elles possibles: (Expliquez):
monochrome := black; monochrome := red; monochrome := c; c:= value(''red'');

b. Donner les valeurs des fonctions (attributs) suivantes: couleur'imge(bleu); couleur'pos(red);

couleur'succ(monochrome);
couleur'first; couleur'last; couleur'width; couleur'base ;

**Exercice** 6/ Ecrire une variable qui peur prendre les valeurs: basic, fortran, cobol, c, ada, PL1.

**Exercice7/**
**I Présentation :** Un train est composé de 10 wagons, dont chacun comporte 60 places assises dont 30 places non fumeur numérotées de 1 à 30 , et 30 places fumeur numérotées de 31 à 60. On veut réaliser un logiciel permettant la réservation des places et l'édition d'une liste de voyageurs destinée aux contrôleurs.

**II Questions**

1. Décrire les variables et les structures de données qui sont utiles pour la résolution de ce problème, puis donner les instructions ADA permettant la déclaration de cet ensemble de données.
2. En utilisant la méthode de programmation structurée, construire l'algorithme détaillé permettant de faire la réservation des places et d'arrêter la réservation lorsque il n'y a plus aucune place assise libre dans les 10 wagons.
3. Traduire l'algorithme précédent en ADA.

**Exercice8/**

**I Présentation :** On souhait utiliser la structure de la liste chaînée pour représenter un ensemble de polynômes (à coefficients entiers) :

**II Questions**

1. Donner le type ***t_polynome*** correspondant à représentation d'un polynôme .

2. Faire une procedure ***aff_pol(p : in t_polynome)*** permettant d'afficher un polynome p.

3. Sur cette ensemble de polynômes on souhait implementer l'opération de la dérivation de polynômes. Faire une ***procedure dériver(p : in t_polynome; pp : out t_polynome);*** qui calcule la dérivation pp de un polynôme p.

4. Réaliser une procédure ***mul_con(p :in : t_polynome, c : in integer, cp : out t_polynome)*** permettant de multiplier un polynôme p par une constante c.

# Chapitre III

# Types composés en ADA

## III.1/ Les TABLEAUX

Un TABLEAU est un objet composé formé d'un certain nombre de composants de même type. La position d'une composante dans un TABLEAU est désignée par une valeur appelée indice qui appartient à un type discret.

### III.1.1/ **Déclaration**:

**array** (<type d'indice>
range <borne inférieur> .. <borne supérieur>)

Exemples de déclaration de variable:

A : array (integer range 1..6) of float

A : array (1..6) of float

A est un objet variable à 6 éléments de type réel flottant.

Exemples de déclaration de type:

Type vecteur is array (1..100) of integer;

Type TABLEAU is array (1..4, 1..3) of integer;

type jour is
(lundi, mardi mercredi, jeudi, vendredi, samedi, dimanche)

Type carnet_heb is array(jour) of string;

c : carnet_heb;

Exercice1: Décrire un type HEURE_O pour gérer les heures ouverture hebdomadaire d'un magasin.

Exercice 2: Généraliser la déclaration précédente pour gérer dans le même tableau les heures d'ouvertures et de fermetures. Ce nouveau type sera appelé HEURE_O_F.

Exercice 3: Décrire le type d'un tableau pour gérer un emploi de temps hebdomadaire.

## III.1.2/ **Accès à un élément d'un tableau**:

```
A(2) := 10;

x := A(2);

-------------------

TABLEAU(1,3) :=  100.0;

y := TABLEAU(1, 3);
```

## **III.1.3/ Initialisation d'un tableau**:

```
for  i  in  1..6 loop
                    a(i) := 0.0;
              end loop;

A := (1..6 => 0.0);

A := (0.0, 0.0, 0.0, 0.0, 1.0, 1.0)

A := (1..4 => 0.0,  5 | 6 => 1.0);

A(2..4) := (1.0, 5.0, 3.0);

 A(1..4 => 0.0, other => 0.1);
```

```
For i := lundi .. vendredi  loop
                              c := ‘’  ‘’  ;
                           end loop;
```

```
TABLEAU := ( (1,1,1), (2,2,2), (3,3,3), (4,4,4) );


For   i  1..4 loop
         for j  in 1..3  loop
              TABLEAU(i,j) := float(i);
         end loop;
end loop;
```

Exercice 1: Déclarer et initialiser une variable HO de type  HEURE_O .

Exercice 2: Déclarer et initialiser une variable HOF de type  HEURE_O _F .

III.1.4/ **Les Attributs associés à un tableau**:

A'first  donne la borne inférieur de l'intervalle de l'indice de A

A'last  =  =  supérieur =  =  =  = A

A'range donne le type de l'indice;

pour un tableau à deux indice:

TABLEAU'last(1)  donne  ---------> 4
TABLEAU'last(2)  donne  ---------> 3

III.1.5/ **Affectation globale**:

```
type V6 is array (1..6) of flaot;

A, B  : V6;

A := (1..6 => 0.1);

B := A;
```

Remarque:  Si on a  Z : array (1..6) of float;

l'affectation Z := A; est illégale.

On peut comparer deux tableaux

if A = b ...

if A < B ...

Si E, F, et G sont de type booléen les trois opérateurs bouléens sont applicables:

Exemple : E := (true, true, false); F:=(False, true, true);

l'affectation G := E and F; donne G:= (false, true, false)

Exercice: calculer G := (not(F) or E );

**III.1.6/ Chaîne de caractères**:

Une chaîne de caractères en ada est un tableau (à une dimension) de type caractère dont l'indice est de type **positive**

Ce type est prédifini par le mot clé STRING;

Les littéraux sont délimités par guillemet ''GIGZAG''

Utilisation:

```
c1 : string (1..9);
c2 : string := ''Yves Montant'';
c3 : constant string := ('B', 'T');
```

la valeur de c2(2) est 'e'
la valeur de c2(6) est 'M'
la valeur de c3(2) est 'T'

si on fait c2(5) := '-' la chaîne c2 sera ''Yves-Montant''

**Operateur supplémentaire**:

concaténation: &

c1 := C2 & c3

Dans ce cas c3 sera égale à ‘’Yves MontantBT’’

Entrée/Sortie : put_line(c2);
get(c1);

**Exercice I**

Soit A un tableau de caractères à une dimension et comportant N éléments (N ≤ M), où M est la dimension de A.

1. Ecrire un algorithme (pseudo code+ lexique) permettant d’insérer un élément x à la k-ième place, k < M.
2. Soit B un autre tableau de même type que A et contient J caractères. Développer un algorithme permettant de ‘’concatener’’ les deux tableaux A et B.
3. On veut utiliser une structure de liste chaînée pour représenter le tableau de caractères décrit plus haut :
   A/ Donner en ADA une structure de données convenable pour représenter une liste de caractères.
   B/ Développer (en ADA) une procédure ***concatener*** permettant de ‘’concatener’’ deux listes de caractères L1 et L2.

**Exercice II**

On suppose qu'on a défini un type Pile (pile de caractère). On donne l'algorithme suivant:

| Lexique | algorithme |
|---|---|
| c : caractère<br><br>P : Pile de caractères | début<br>nouvelle(P)<br>répéter<br>lire(c)<br>si c /= '#' alors<br>si non vide(P) alors<br>dépiler(P)<br>fsi<br>sinon<br>si c = '@' alors<br>nouvelle(P);<br>sinon<br>si c /= '$' alors<br>empiler (c, P)<br>fsi<br>fsi<br>fsi<br>jusqu'à c = '$'<br><br>tannique non vide(P) faire<br>sommet (P, c)<br>écrire ( c ) |

| | |
|---|---|
| | dépiler (P)<br>ftq<br>fin |

A l'exécution, on saisit abc@lest#o##on$

a/ Donner les états successifs de la pile au cours de l'exécution.
b/ Donner le résultat de cette exécution.
c/ Pour manipuler une pile, on suppose l'existant d'un paquetage g_pile (déjà vu en cours).
Adapter puis utiliser ce paquetage, pour traduire l'algorithme précédent en ADA.

## --- exemple de paquetage de pile générique

```
generic
   type element is private;
package g_pile is
    taille : constant integer := 50;
    subtype indice is integer range 0..taille;
    type tab is array(indice) of element;
    type pile is record
        corps : tab;
        tete : indice;
    end record;
    procedure nouvelle(p:out pile);
    procedure empiler(x:element; p:in out pile);
    function vide(p:in pile) return boolean;
    procedure sommet (p:in pile;x:out element);
    procedure depiler(p:in out pile);
end g_pile;

with text_io;use text_io;

package body g_pile is
    procedure nouvelle (p:out pile) is
    begin p.tete:=0 ; end;
    function vide (p:in pile) return boolean is
    begin
        if p.tete= 0 then put ("pile vide"); return false;
            else return true;
        end if;
    end;
    procedure empiler(x:element;p:in out pile) is
    begin
        if p.tete = taille then
            put("erreur");
        else
            p.tete:=p.tete + 1;
            p.corps(p.tete):=x;
        end if;
    end;
    procedure depiler(p:in out pile) is
    begin
        if p.tete = 0 then put ("erreur");
```

```
            else p.tete:=p.tete - 1;
        end if;
    end;
    procedure sommet (p:in pile;x:out element) is
    begin x:=p.corps(p.tete); end;
end g_pile;

---------------------------------
-- Programme de teste pour le paquetage précédent

with g_pile;
with text_io;use text_io;

procedure tg_pile is
    type tvoiture is record
        n:integer range 0..9999;
        m:string(1..2);
        d:integer range 1..100;
    end record;

package g_pile_voiture is new g_pile(tvoiture);
use g_pile_voiture;
p:pile;
x,v:tvoiture;

begin

    nouvelle (p);

    for i in 1..52 loop
    v.n:=1111;
    v.m:="PQ";
    v.d:=12;
    empiler(v,p);
    sommet(p,x);
    put_line(integer'image(i) & integer'image(x.n) & x.m &
integer'image(x.d));
    end loop;
end;
```

# Exemple complet : Dossier de programmation

***I/ Problème*** : Gestion d'agenda hebdomadaire.

Etant donné un ensemble de jour de travail, et un ensemble de tranche d'horaire d'occupation, qui sont les mêmes pour chaque jour de travail. On défini un agenda comme est un ensemble de phrases (une phrase est une chaîne de caractères), chaque phrase est accossiée à un couple de jour et de tranche horaire.
On veut automatiser la consultation et mise à jour de cet agenda .

***II/ Algorithme*** :

Premier niveau d'abstraction:

```
Lexique                          algorithme général

  choix: un entier pour            Afficher le menu général;
        récupérer le
        choix de                   lecture et control de choix
        l'utilisateur
  agenda: un tableau               si choix = 1 alors
         de 2 dimension              mise à jour agenda fsi;
         pour gérer
         l'agenda,il est           si choix = 2 alors
         indexé par                   affiche planning hebdomadaire fsi;
         les jours de
         de la semaine             si choix = 3 alors affiche
         et les tranches             affiche le planning d'une journée;
         horaires.                 fsi;

                                    si choix = 4 fin de traitement;
```

2ème niveau d'abstraction

```
    affiche menu général
      effacer l'ecran;
      écrire(" donner votre choix en tapant:
              1  pour la mise à jour de l'agenda;
              2  pour afficher le planning hebdomadaire
              3  pour afficher le planning d' une journée
              4  pour sortir ");

    lecture et contrôle de choix;

       lire (choix);
      tq choix (  (1,2,3,4) faire
         écrire( "message d'avertissement");
         lire( choix);
      refaire;


lexique                          mise à jour de l'agenda
```

lecture et contrôle de jour;

jour: une journée

lecture et contrôle de la tranche
horaire (th);

th : tranche
horaire

lecture de l'occupation;

occupation est une
caractère represen-
ant l'occupation de
une tranche d'horaire

agenda(jour, th) := occupation;

lexique

houres, th sont défini

précédemment

jours: les jours de
occupation

heures: les tranches
horaires

Afficher un planning hebdomadaire

pour tout th ( heures faire
écrire th; (* sans retour à la ligne *)
refaire;

pour tout jour ( jours faire
pour tout th de heures faire

écrire(jour, occupation(jour,th);

refaire;
refaire;

lexique

heures ,
jour, et th sont
défini précédemment

Afficher un planning journalier

écrire(" planning de la journée", jour);
pour tout th ( heures faire
écrire th; (* sans retour à la ligne *)
refaire;

pour tout th de heures faire

jours: les jours de
occupation

heures: les tranches
horaires

écrire(jour, occupation(jour,th);

refaire;

3ème niveau d'abstraction

lecture et contrôle de jour

```
effacer l'ecran;
lire(jour) ;
tq jour ( jour faire
    écrire (" message d'erreur ");
    lire ( jour );
refaire;
```

lecture et contrôle de th

```
effacer l'ecran;
lire(th) ;
tq th ( heures faire
   écrire (" message d'erreur ");
   lire ( th );
refaire;
```

***III/ programme en ADA***

---------- Gestion d'un agenda

```
with text_io;use text_io;

procedure planning is

type Tjour is (lundi,mardi,mercredi,jeudi,vendredi,samedi,dimanche);
type Theure is
(H8_9,H9_10,H10_11,H11_12,H12_13,H13_14,H14_15,H15_16,H16_17);
type Tagenda is array (Tjour,Theure) of string (1..21);
choix : integer;
agenda : Tagenda;

package h_enum is new enumeration_io(Theure);
package j_enum is new enumeration_io(Tjour);

-------------------------------------------------------------------------
--              Afficher le menu général
procedure affimenu is
 begin
        put_line ("               Menu  ");
        put_line ("1. Mise a jour de l'agenda");
        put_line ("2. Consultation du planning hebdomadaire");
        put_line ("3. Consultation du planning journalier");
        put_line ("4. Sortie");
        put_line (" ");
        put ("          Votre choix : ");
 end;

-------------------------------------------------------------------------
-                   lecture et control de choix
procedure lectchoix is

  c : character;                                  ----- declaration d'une
variable c : pour eviter un conflit de type
  cc : string(1..2) := "  " ;                ----- en travaillant avec la
variable Choix si l'utilisateur entre
                                                      ----- une lettre
par exemple.
  begin
```

```
         get (c);
         while c not in '1'..'4' loop
                put_line ("Entrer un chiffre de 1 a 4");
                get (c);
         end loop;
         cc(1):=  c;
         choix := integer'value (cc);         ----- conversion de C en
type choix.
  end;

-------------------------------------------------------------
mise à jour agenda
procedure miseajour is
jour : Tjour;
        th : Theure;
--      occup : string;
begin
        put_line ("Entrer le jour :");
        j_enum.get(jour);
        put_line ("Entrer la tranche horaire a mettre a jour (entre 8h
et 17h)");
        put_line ("sous forme H12_13 par exemple : ");
        h_enum.get(th);
        put_line ("Entrer l'occupation (Entrer exactement 20
caracteres):");
        get (agenda(jour,th));
 end;

-------------------------------------------------------------
affiche planning hebdomadaire

procedure planhebdo is
ths : Theure;
begin
        new_line(2);
        put_line ("Planning hebdomadaire :");
        while jour in lundi..dimanche loop
                put ((jour) & " a " & (th) &" : ");
                put (agenda(jour,th)); new_line;
        end loop;
end;
-------------------------------------------------------------------
affiche le planning d'une journée
procedure planjour is
jour : Tjour;
        th : Theure;
 begin
        put_line ("Entrer le jour :");
        j_enum.get(jour);
        put_line ("Entrer la tranche horaire a mettre a jour (entre 8h
et 17h)");
        put_line ("sous forme H12_13 par exemple : ");
        h_enum.get(th);
```

```
        put ("Le " & tjour'image(jour) & " a "  & theur'image(th) & "
vous devez faire: " );
        put (agenda(jour,th));    newline;
end;

-------------------------------------------------------------
algorithme général
Begin
                affimenu;
                lectchoix;
                if choix = 1 then miseajour;
                end if;
                if choix = 2 then planhebdo;
                end if;
                if choix = 3 then planjour;
                end if;
                if choix = 4 then
                  put_line ("Fin de traitement");
                end if;
 end;
```

***IV/ jeux d'essai***

à faire;

# Le type d'accès en ada

## 1/ Introduction

Nouveaux problèmes

- le traitement d'informations dont on ne connaît pas le nombre,
- la construction de structure de données complexe (arbre, graphe, ...);

## 2/ La déclaration et la manipulation du type Accès

Pour faire face aux problèmes précédents Ada offre le type accès dont la description et les caractéristiques sont :

- Il existe un type de base que l'on appellera type accédé pouvant être de n'importe quel type (sauf un type accès).

- Le type accès est définie à partir du type de base signalé précédemment en utilisant l'instruction :

  **Type** acces_exemple **is acces** integer;

  Les variables x, y de type accès se déclare comme suite :

  x, y : acces_exemple;

- Les objets de type accédé peuvent être créés en cours d'exécution par la commande NEW qui crée une valeur de type accès.

- L'accès aux objets créés par accès se fait en utilisant la notation pointée. L'utilisation du mot réservé ALL permet d'accéder globalement l'objet.
  Exemples

- l'instruction **New** integer permet la création d'un objet de type integer,

- avec l'instruction x := **new** integer; un objet entier est créé; il est accessible via l'objet x
- maintenant avec l'instruction y := x ; l'entier créé précédemment est aussi accessible via la variable y.
- observons l'exécution du programme suivant:

```
-- affectation d'un objet par acces with text_io;
use text_io;
procedure a1 is
package es_e is new integer_io(integer); use es_e;
type a_e is access integer;
x, y : a_e;
begin
        new_line;
        x := new integer;

        x.all := 5;
        put( x.all); new_line;              --ecrit 5
        y := x;
        put( y.all); new_line;              --ecrit 5

        Y.all := 7;
        put( y.all); new_line;              --ecrit 7
        put( x.all); new_line;              --ecrit 7
end a1;
```

## Utilisation d'une variable de type accès :

### Exemple 1(avec un type de base structuré)

```
type date is record
                j   : integer range 1.. 31;
                    m : integer range 1..12;
                    a  : integer range 1000 .. 2999;
                end record;
    type a_d  is access date;
    p, q : a_d;
    d : date;
    d.j :=  1;  d.m := 1;  d.a := 1997;
    p := new date;
    p.all :=  d;
    q := p ;
```

- à sa déclaration p, q contient une valeur spéciale (NULL), cette valeur indique que la variable ne contient pas d'accès sur un objet.

- Dans l'exemple suivant : p := q;
  on utilise p sans notation pointée p dans cette exemple contient un accès au même objet accédé par q.

Si on utilise p avec une notation pointée comme par exemple
p.j := 1 ; qui affect la valeur 1 au champ j de l'article accéder par p.

Après l'exécution de l'instruction p.all := d ; la cellule accédée par p contient l'enregistrement d.

## V.3/ Les structures de données récursives

Une telle structure se définie par rapport à elle-même.

```
-- Exemple 2 : Ecrire un programme qui lit au clavier une suite d'entiers
--          positifs ou nuls et qui cree une liste chainee de ces entiers.
--          Le contenu de la liste est ensuite ecrit sur l'ecran.

with text_io; use text_io;
procedure a2 is
 package es_e is new integer_io(integer); use es_e;
 type cellule;
 type lien is access cellule;
 type cellule is record
                        valeur  : integer;
                        suivant : lien;
                 end record;
 l : lien;

 procedure cree_liste(la_liste : out lien) is
  un_entier : integer := 0;
  tete, courant : lien;
  begin
   while un_entier >= 0 loop
     put(" donner un entier a inserer dans la liste (si negatif
terminer):");
     get(un_entier);
     if  un_entier >=  0 then  courant := new cellule;
                               courant.valeur := un_entier;
                               courant.suivant := tete;
                               tete := courant;
                          end if;
   end loop;
   la_liste := tete;
  end cree_liste;

 procedure ecrire_liste(liste : in lien) is
  courant : lien;
  begin
    courant := liste;
    while courant /= null loop
      put(courant.valeur); put("  ---->");
      courant := courant.suivant;
    end loop;
    put("  null ");
  end ecrire_liste;

 begin
   new_line;
   put_line(" lecture de la liste : ");
   cree_liste(l);
   new_line;
   put_line(" affichage de la liste : ");
   ecrire_liste(l);
 end a2;
```

**CNAM LE HAVRE, U.V. Algorithme et programmation (programmation en ADA),TD4, 9/05/1997, Enseignant: M. Nakechbandi**

**Thème:** Le type d'accès en ADA, la liste chainée.

Exercice : On souhait utiliser la structure de la liste chaînée pour représenter un ensemble de polynômes (à coefficients entiers) :

1. Donner le type ***t_polynome*** correspondant à représentation d'un polynôme .

2. Faire une procedure ***aff_pol(p : in t_polynome)*** permettant d'afficher un polynome p.

3. Sur cette ensemble de polynômes on souhait implementer l'opération de la dérivation de polynômes. Faire une ***procedure dériver(p : in t_polynome; pp : out t_polynome);*** qui calcule la dérivation pp de un polynôme p.

4. Réaliser une procédure ***mul_con(p :in : t_polynome, c : in integer, cp : out t_polynome)*** permettant de multiplier un polynôme p par une constante c.

## ANNEXE 1

Le développement en ADA du pseudo code décrit en page 9

```
 with text_io;use text_io;
  with sequential_io;
  procedure creation_fichier is
     type jugement is (rare,tres_beau,beau,petit_defaut);
     type article is
       record
       code  : integer range 1..100;
       valeur : integer range 1..10000;
       proprietaire : string (1..10);
       appreciation : jugement;
       divers :  string (1..100);
       end record;

 package ES_ENT is new integer_io (integer);
 package ES_ART is new sequential_io (article);
 use ES_ENT; use ES_ART;

 FICHIER : ES_ART.FILE_TYPE;
 REPONSE : character;

 procedure saisie ( FICHIER : IN OUT ES_ART.FILE_TYPE) is
 REP: character;
 X : article;
 lg : natural;
 begin
    while (REP = 'O') loop
      X.code := (others => ' ');
      put_line ( " code article :  ");
      get_line ( X.code,lg);
      put_line ( " valeur article :  ");
      get_line ( X.valeur,lg);
      put_line ( " proprietaire article :  ");
      get_line ( X.proprietaire,lg);
      put_line ( " appreciation :  ");
      get_line ( X.appreciation,lg);
      put_line ( " divers :  ");
      get_line ( X.divers,lg);
      WRITE (FICHIER,X);
      new_line;
      put ( " voulez-vous continuer la saise? (O/N) ");
      get (REP);
      skip_line;
   end loop;
end saisie;

procedure afficher (FICHIER : IN OUT ES_ART.FILE.TYPE) IS
x : article;
begin

      put_line (" Fichier d'articles : ");
      new_line;
        while not END_OF_FILE (FICHIER) loop
```

```
          READ (FICHIER,X);
          PUT ("code: ");PUT (X.code);PUT (" ");
          PUT ("valeur : ");PUT (X.valeur);PUT (" ");
          PUT ("proprietaire : ");PUT (X.proprietaire);PUT (" ");
          PUT ("appreciation: ");PUT (X.proprietaire); PUT (" ");
          PUT ("divers: ");PUT (X.divers); PUT (" ");
          new_line;
          end loop;
end afficher;

type cellule;
type lien is access cellule;
type cellule is record
X : cellule;
valeur : article;
suivant : lien;
end record;
l : lien;

procedure charger_liste is
X.code : integer := 0;
tete,courant : lien;
begin
while X.code := (other => ' ') loop
     put ("donner un code … inserer: ");
     get (X.code);
     end loop;
end charger_liste;

procedure  affiche_liste is
courant : lien;
begin
courant := liste;
while courant /= null loop
PUT (courant.valeur); put ( " ---> ");
courant := courant;suivant;
end loop
put ("null");
end affiche_liste;

procedure fusionner_fichier is
f,f1,f2 : ES_ART.FILE_TYPE;
a1,a2 : valeur;

open(f1,name==>"f1.dat");
open(f2,name==>"f2.dat");
create(f,name==>"f.dat");

READ (f1,a1); READ (F2,a2);
  while not eof(F1) and not eof(F2) loop
    if a1.code < a2.code then write  (F, a1); read (F1,a1);
                         else  write  (F, a2); read (F2,a2);
    end if;
    end loop;

if not eof(f1) then
while not eof(f1) loop
  read(f1,a1);
  write(f,a1);
```

```
end loop;
if not eof(f2) then
while not eof(f2) loop
  read(f2,a2);
  write(f,a2);
  end loop;

close(f); (f1);(f2);
 end fusionner_

---PROGRAMME---
procedure affiche_menu is
begin
new_line(25);
put_line ("                MENU");
new_line; new_line;
put_line ("1: saisie fichier");
new_line;
put_line ("2: afficher fichier");
new_line;
put_line ("3: charger liste");
new_line;
put_line ("4: afficher liste ");
new_line;
put_line ("5: fusionner fichier ");
new_line;
put_line ("6: fin du traitement ");
new_line;new_line;
  put  ("  votre choix  ");
end;

procedure lecture_choix is
c:character; cc : string(1..2) := " ";
begin
get (c)
while c not in '1'..'6' loop
put_line ( entrer un chiffre de 1 … 5 ");  get(c);
end loop;
cc(1):= c;
choix:=integer'value(cc);
end;

BEGIN
put_line (" debut du traitement " ); stop_ecran;
charger_liste;
affiche_menu;
lecture_choix;
while ((choix >=1) and (choix <=5)) loop
case choix is
  when 1  => saisie;
  when 2  => afficher;
  when 3  => charger_liste;
  when 4  => affiche_liste;
  when 5  => fusionner_fichier:
  when others => put_line ("erreur");
end case
affiche_menu; lecture_choix;
end loop;
end;
```

# ANNEXE 2
# Sujet d'examens

## Sujet 1
**CNAM, Le Havre, Algorithme et programmation, Devoir Surveillé N° 1, 7/2/1995.**
Enseignant: NAKECHBANDI M. Document autorisé: notes de cours

**Problème I**

On donne l'algorithme suivant

| lexique | algorithme |
|---|---|
| a, b, x, y, s (entier) | debut<br>lire (a,b)<br>s < 0<br>x < a<br>y < b<br>tant que y =/= 0 faire<br>si y mod 2 =/= 0 alors<br>y < y - 1<br>s < s + x<br>sinon<br>x < x * 2<br>y < y div 2<br>finsi<br>refaire<br>ecrire (s)<br>fin |

I.1/ Compléter les tableaux suivants, donnant les valeurs successives de a,b,s,x, et y au cours de l'exécution, les valeurs initiales a et b étant:

- 8 et 7;
- 5 et 4.

| a | b | x | y | s |
|---|---|---|---|---|
| 8 | 7 | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

| a | b | x | y | s |
|---|---|---|---|---|
| 5 | 4 | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

I.2/ Que calcule cet algorithme ? Justifiez votre réponse.

I.3/ Ecrire un programme ADA correspondant à l'algorithme précèdent.

**Problème II**

Un organisme de location de voitures propose deux formules de location:

a) location au kilomètre:
Jusqu'à 100 km: tarif t1 au kilomètre;
Pour un kilomètre compris entre 101 et 1000 km: tarif t2 au kilomètre;
Au-delà de 1000 km: tarif t3 au kilomètre;
b) forfait journalier:
un prix forfaitaire pj par jour, kilomètre illimité.

Dans les deux cas, il convient d'ajouter une assurance dont le coût à la journée est a1 .
Les quantités t1, t2, t3, pj, et a1 sont considérées comme constante ; le nombre de jours de location et le nombre de kilomètre sont données. (à saisir).

II.1/ Ecrire un algorithme qui permet de trouver le coût total suivant les deux tarifications, et qui indique par un libellé la solution la plus avantageuse.

II.2/ Ecrire un programme ADA correspondant à votre algorithme.

Sujet 2

**CNAM, Le Havre, Algorithmique et Programmation A.**

**DS2 - 23 mais 1997.**

Document autorisé: notes de cours. Durée 2 heures. Enseignant: NAKECHBANDI M.

Barème indicatif : exercice 1 (1,5 pt), exercice 2 (1,5 pt), exercice 3 (4 pts), exercice 4 (4 pts), exercice 5 (4 pts), exercice 6 (5 pts).

---

## I Présentation

Dans une entreprise, il y 2 succursales, chaque succursale possède un fichier des articles qu'elle a en stock. Chaque article est référencé par un code et par la quantité disponible dans la succursale concernée. Les deux succursales utilisent utilise les mêmes codes pour les mêmes articles. On suppose aussi tous les fichiers sont triés par l'ordre croissant du code d'article.

## II Questions :

1. Donner les instructions ADA permettant de déclarer le fichier d'articles de stock.

2. On veut placer le fichier de stock dans une structure d'une liste chaînée. Donner les déclarations nécessaires.

3. Développer en ADA une procédure ***saisir*** permettant de saisir un fichier de stock.

4. Développer en ADA une procédure ***afficher*** permettant d'afficher un fichier de stock.

5. Développer en ADA une procédure ***charger*** permettant de placer un fichier de stock dans une liste chaînée.

6. On décide de fusionner les stocks de deux succursales. Ecrire en ADA la procédure ***fusionner*** permettant la fusion des stocks de deux succursales et d'obtenir un nouveau fichier de stock trié.

**Remarque :** Pour chaque procédure donnez tout d'abord le schéma de votre algorithme général (un pseudo code + lexique). Vous pouvez ensuite détailler votre algorithme directement en ADA.