Algorithmique & Programmation

Introduction

Algorithmique & Programmation

- Algorithmique
  - ***L'art*** de concevoir un algorithme
- Algorithme
  - Une suite d'actions (**instructions**) que doit effectuer un automate (**ordinateur**) …

  … pour produire en un **temps fini**,

  … à **partir d'une situation donnée** (***précondition***)

  … un **résultat déterminé** (***postcondition***),

  - Écrit dans un langage algorithmique

Types d'algorithmes (1/2)

- **fonction** : retourne un résultat
  - Analogie avec une fonction mathématique
    - f(x) → y
      - f appliquée à x retourne le résultat y
      - x s'appelle un paramètre
  - Exemples
    - Fonction **minimum** qui retourne le plus petit parmi deux entiers x et y
      - fonction **minimum**(x,y : entier) retourne un entier ;
    - Fonction **majeur** qui retourne vrai si le paramètre entier x est supérieur ou égal à 1 & retourne faux sinon
      - fonction **majeur**(x : entier) retourne vrai ou faux ;

Types d'algorithmes (2/2)

- **procédure**
  - Ne retourne pas de résultat au sens d'une fonction
    - p(x)
      - p appliquée à x ne retourne pas de résultat
      - x s'appelle un paramètre
  - Exemples
    - Procédure **permute** qui permute la valeur de deux entiers x et y
      - procédure **permute**(x,y : entier);
    - Procédure **affiche10** qui affiche sur l'écran de l'ordinateur les entiers de 1 à 10
      - procédure **affiche10**;

Algorithmique & Programmation

- Programmation
  - ***L'art*** d'écrire un programme informatique
- Programme
  - La traduction d'un ensemble d'algorithmes qui vont

  … s'exécuter en un **temps fini**,

  … pour des **préconditions toujours vérifiées**

  … produire les **mêmes résultats déterminés**
    - postconditions

  - Écrit dans un langage de programmation
    - Ada, C, C++, Java, …

Le langage `ada`

- Langage proposé en 1983
  - par une équipe française (Ichbiah)
  - en réponse à une demande du Département de la Défense Américain (DoD)

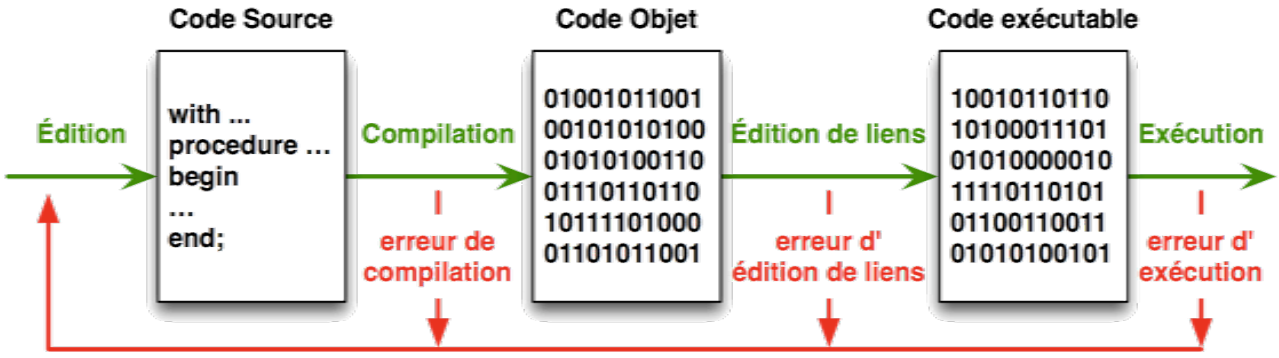
- Tente de résoudre le problème de la crise du logiciel dont les symptômes sont :
  - Logiciels ne répondant pas au cahier des charges des utilisateurs
  - Pas fiables
  - Plus coûteux que prévus
  - Livrés en retard
  - Non portables
  - Gourmands en temps d'exécution et en ressources matérielles

Création d'un programme

- Ce n'est qu'une des étapes d'un long cycle
  - le **cycle de vie** d'un système
- Étapes du cycle de vie :
  - L'analyse du problème
  - La conception :
    - Définition, choix des **structures de données**
    - Définition, choix des **algorithmes**
  - La programmation & le test
    - Écriture, création d'un programme
  - La livraison
  - La maintenance

Création d'un programme

- Les différentes étapes

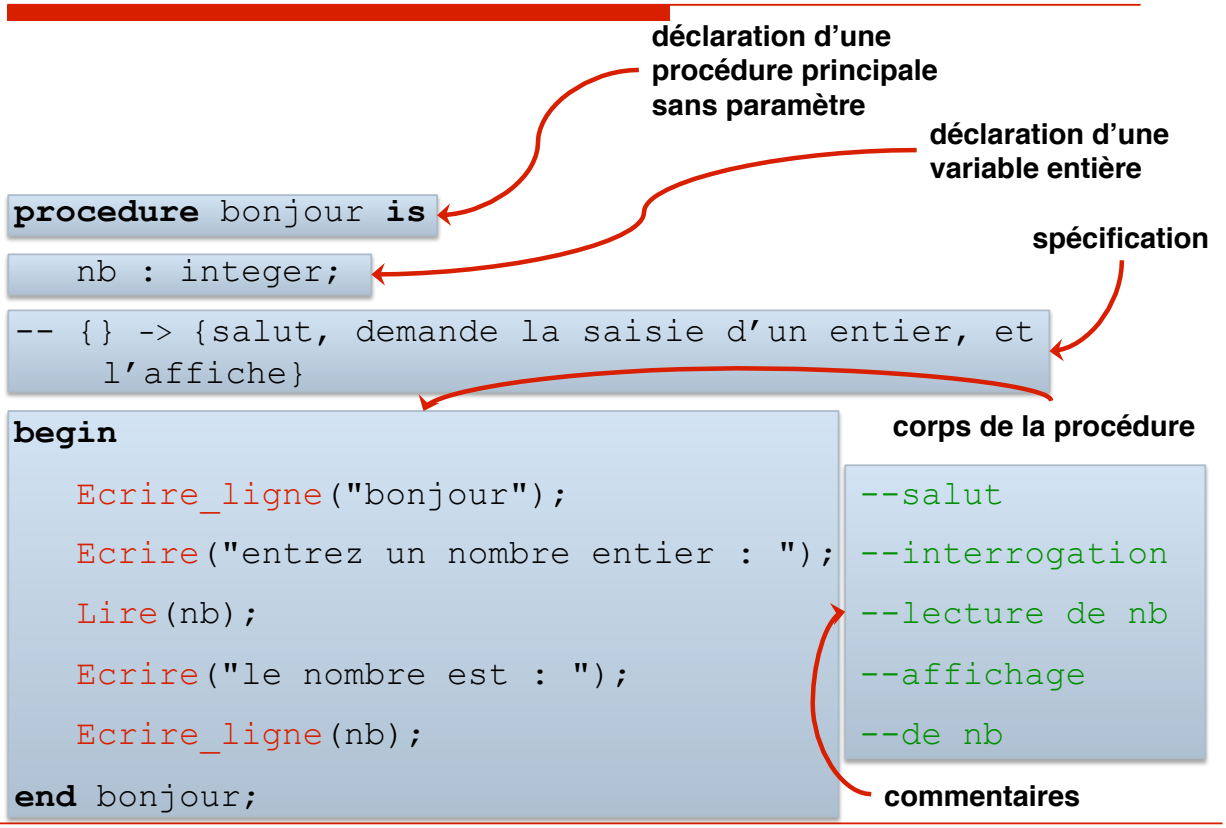


- **Édition** : écriture du code source dans le langage de programmation choisi
- **Compilation** : production d'un fichier binaire (code machine + informations pour l'édition de lien et le débogage)
- **Édition de liens** : production d'un fichier binaire exécutable

Programme **ada**

- Composé de :
  - Un algorithme principal qui est une procédure
  - Un ensemble de fonctions et de procédures qui sont utilisées …
    - … par la procédure principale
    - … par une autre fonction ou une procédure autre que la procédure principale

Premier programme **ada**



Premier programme **ada**

corps de la procédure

```
procedure bonjour is
   nb : integer;
-- {} -> {salut, demande la saisie d'un entier, et
     l'affiche}
begin
   Ecrire_ligne("bonjour");
   Ecrire("entrez un nombre entier : ");
   Lire(nb);
   Ecrire("le nombre est : ");
   Ecrire_ligne(nb);
end bonjour;
```

Liste d'instructions :
– Appels de procédures

D'où viennent-elles ???

Premier programme **ada**

```
with P_ESiut;
use P_Esiut;
```

bibliothèque (package) à utiliser

Les procédures
- Ecrire_ligne()
- Ecrire()
- Lire()

sont définies dans le package **P_ESiut**

```
procedure bonjour is
   nb : integer;
-- {} -> {salut, demande la saisie d'un entier, et
     l'affiche}
begin
   Ecrire_ligne("bonjour");
   Ecrire("entrez un nombre entier : ");
   Lire(nb);
   Ecrire("le nombre est : ");
   Ecrire_ligne(nb);
end bonjour;
```

Liste d'instructions :
– Appels de procédures

Structure d'un programme simple

```
procedure nom_proc is
   PARTIE DECLARATIVE
   -- ensemble de déclarations des
   -- données et procédures du programme
begin
   CORPS de la PROCEDURE
   -- ensemble des instructions à exécuter
end nom_proc ;
```


Instructions

- Un programme est composé d'instructions. Une instruction peut elle-même contenir d'autres instructions.
- Par défaut, les instructions s'exécutent en séquence.
- En **ada** les instructions sont terminées par un point virgule.
- Une instruction peut s'étendre sur plusieurs lignes.
- Pas de différence entre majuscules et minuscules

Commentaires

- Les commentaires ont pour objectif de faciliter la compréhension du programme
- Les commentaires débutent par -- et se poursuivent jusqu'à la fin de ligne
  - `-- ceci est un commentaire`

Notion de variable

- Chaque donnée (nombre entier, caractère, …) que l'on veut réutiliser ultérieurement en changeant sa valeur doit être stockée dans un emplacement mémoire appelé « **variable** ».

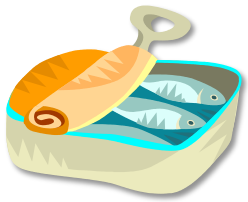
- Les 2 langages, **algorithmique** et **ada**, sont « fortement typés »

  
  une variable ne peut contenir qu'une seule sorte ou « type » de données (caractère, nombre entier, …)

- Les variables utilisées par un programme doivent être « déclarées » avant leur utilisation
  - on précise leur nom et l'unique type de valeur qu'elles peuvent contenir.

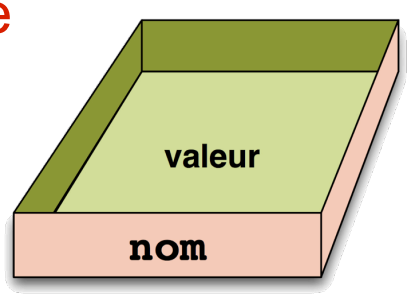
Notion de variable

- Un algorithme ou un programme ne manipulent pas que des constantes
  - on a besoin de **variables**
- Analogie avec une boîte de la vie réelle
  - Elle ne peut contenir qu'un (des) objet(s) d'un certain type, elle a une étiquette, et un contenu
    - un carton à vêtements, une boîte de sardines
    - **on ne range pas un vêtement dans une boîte de sardines, ni l'inverse !!**



Notion de variable

- En algorithmique & programmation, une variable à
  - un type de donnée
  - un **nom**, et
  - une **valeur**
- On ne peut donner à la variable qu'une **valeur** du type de donnée



Types simples disponibles

Donnée	en ada	pour les algorithmes
type entier	`integer`	entier
type réel	`float`	réel
type booléen	`boolean`[1]	booléen
type caractère	`character`[2]	caractère
type chaîne de caractères	`string`[3]	chaîne

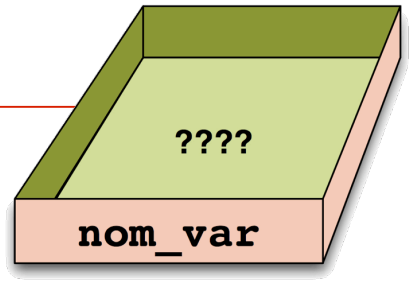
1. il existe deux valeurs de ce type **true** et **false** (***vrai*** et ***faux***)
2. un caractère est encadré par deux **'**
   - exemples : 'a', 'b', '.', …
3. une chaîne de caractères est encadrée par deux **"**
   - exemples : "banane", "pomme"

Notions de valeur et de type

- Dans le langage Ada, qui est un langage fortement typé, toute valeur a **un type et un seul**.
  - `3` est un **entier**
  - `4.0` est un **réel**
  - `'c'` est un **caractère**
  - `banane` est un **fruit** *(cf. type énuméré)*
  - `"test"` est une **chaîne de caractères**

- Notez bien que c'est par la **manière** dont on écrit une valeur que l'on spécifie son type :
  - `3` est un **entier**
  - `'3'` est un **caractère**
  - `3.0` est un **réel**
  - `"3"` est une **chaîne de caractères**

Déclaration de variable



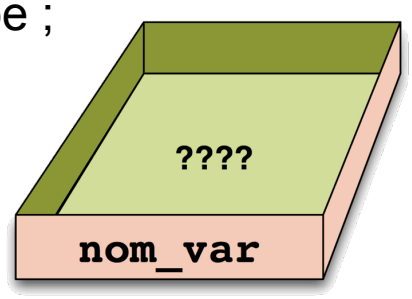
- Forme générale :
  - nom_var : type ;

en ada	dans un algorithme
`nom : string(1..15);`	nom : chaîne ;
`nom, prénom : string(1..15);`	nom, prénom : chaîne ;
`an_naiss : integer;`	an_naiss : entier ;
`trouvé : boolean;`	trouvé : booléen ;

⚠ en Ada il faut préciser la taille des chaînes (`string`)

Variable : initialisation

- La déclaration d'une variable permet de disposer d'un espace pour stocker sa valeur
  - nom_var : type ;



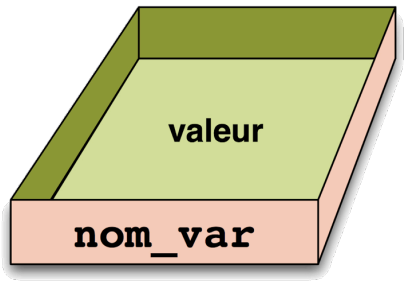
⚠ Avant de pouvoir utiliser une variable, il faut lui donner une valeur initiale (l'initialiser)

Variable : initialisation

- 2 méthodes
  - par une affectation
  - par une procédure de lecture

Variable : initialisation par affectation

- Instruction d'affectation «:=»

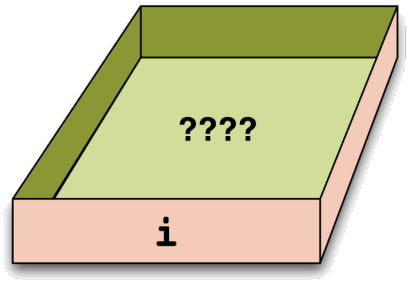


  - nom_var := valeur ;
  - *{la variable* nom *prend la valeur* Dupont*}*

en ada	dans un algorithme	commentaire
`i:=3;`	i:=3;	si **i** est un entier
`nom:="Dupont"`	nom:="Dupont"	si **nom** est une chaîne
`trouvé:=true;`	trouvé:=vrai;	si **trouvé** est un booléen

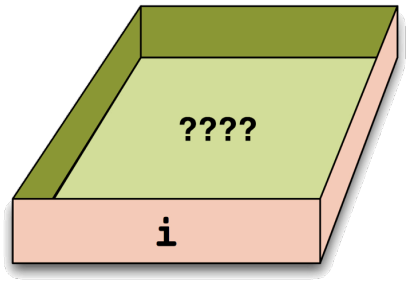
Évaluation d'une affectation

- Exemple
  - `i : integer;`
  - `...`
  - `i:=3;`



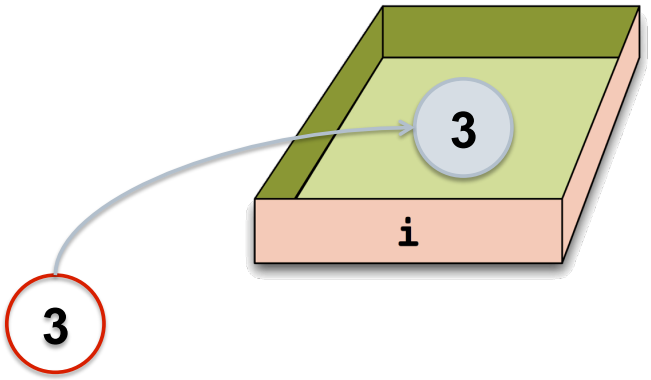
Évaluation d'une affectation

- Exemple
  - `i : integer;`
  - `...`
  - `i:=3;`



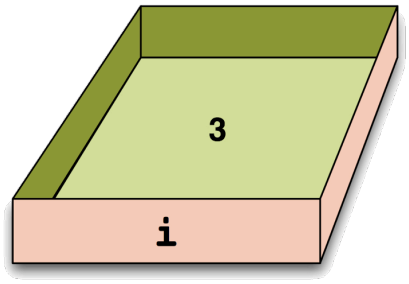
Évaluation d'une affectation

- Exemple
  - `i : integer;`
  - `...`
  - `i:=3;`



Évaluation d'une affectation

- Exemple
  - `i : integer;`
  - `...`
  - `i:=3;`



Variable : initialisation par lecture

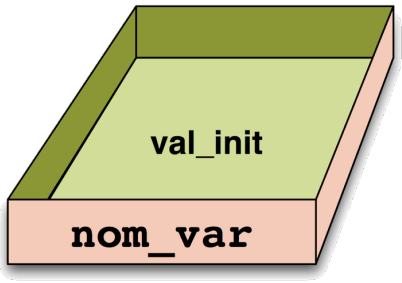
- Procédures de lecture du clavier
  - La variable prend la valeur saisie au clavier

en ada	dans un algorithme
`lire(nom_de_variable);`	litexte(nom_de_variable);

- Exemples :
  - **`lire(nom);`** **`--ada`**
    - l'utilisateur saisit **`"Dupont"`** au clavier
    - *`--la variable`* **`nom`** *`prend la valeur`* **`Dupont`**
  - **litexte(nom)** ; *{algo}*
    - l'utilisateur saisit "**Dupont**" au clavier
    - *{la variable* **nom** *prend la valeur* **Dupont***}*

Déclaration de variable initialisée

- Forme générale
  - nom_var : type := val_init ;



⚠ pas utilisée dans les algorithmes

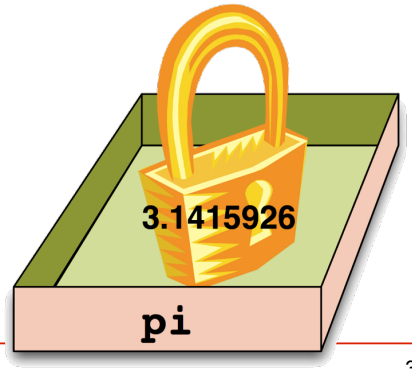
en ada
`nom : string(1..7) := "voiture" ;`
`compteur : integer := 1;`
`f : float := 3.0;`

Notion & déclaration de constante

- Associer un nom à une valeur qui ne sera pas modifiée
  - utilisation de la valeur en utilisant uniquement son nom.
- Exemple
  - la **valeur** du nombre π peut être nommée `pi` (les caractères autres que ceux de l'alphabet latin ne sont pas autorisés en Ada) en déclarant :

```
pi : constant float := 3.1415926;
```

  - le nom `pi` a été associé à la valeur `3.1415926` grâce au symbole `:=`
- Le mot-clé `constant` signifie que l'association du nom et de la valeur est définitive, donc constante dans le temps



Variable : utilisation de la valeur

- 2 méthodes
  - consultation
  - affichage à l'écran
- Rappel :

on peut utiliser la valeur d'une variable seulement si elle a été initialisée

Variable : affichage

- Procédure d'affichage à l'écran
  - la valeur de la variable est affichée à l'écran

en ada	dans un algorithme
`ecrire(nom_de_variable);`	écritexte(nom_de_variable);
`ecrire_ligne (nom_de_variable);`	

- Notes

`ecrire_ligne(var)` ne peut pas être utilisée si `var` est un caractère


```
with P_Esiut; use P_Esiut;
            -- Le programme de ce fichier va utiliser p_esiut

procedure E_S is
              -- Un programme est une procédure sans paramètre
    -- Déclarations et Initialisations des variables
    I   : Integer           := 3;
    F   : Float             := 3.0;
    C   : Character         := 'A';
    Nom : String (1 .. 20);
    S   : String (1 .. 27) := "une chaîne de 27 caractères";
begin  -- Début de l'algorithme
    Ecrire (2);    Ecrire_Ligne (I);
    Ecrire_Ligne (F);
    Ecrire ('b');    Ecrire (C);    A_La_Ligne;
    Ecrire_Ligne (S);
    Ecrire ("Quel est votre nom ? ");
    Lire (Nom);
    Ecrire ("Votre nom est : " & Nom);
end E_S; -- Fin de l'algorithme
```

??

Affichage : le symbole &

- En **ada**
  - Le symbole **&** permet la concaténation de deux chaînes de caractères uniquement
    - `Nom : String (1 .. 6) := "Dupont";`
    - `...`
    - `Ecrire ("Votre nom est : " & Nom);`
    - `écrit à l'écran Votre nom est : Dupont`
- En **algorithmique**
  - Le symbole **&** permet la concaténation de valeurs de tous les types de données
    - compteur := 6;
    - écritexte("Le compteur vaut : " & compteur)
    - écrit à l'écran Le compteur vaut : 6

Variable : consultation

- Une variable est **consultée** lorsque son nom figure en **partie droite d'affectation**
- Exemple

```
procedure exemple is
   i, j : integer;
begin
   i:=3;
   j:=2;
   j:=i+j;
   ecrire_ligne(j);
end exemple;
```

affectation qui initialise i à la valeur 3.
- partie gauche : un nom de variable
- partie droite : une valeur entière

affectation qui initialise j à la valeur 5.
- partie gauche : un nom de variable
- partie droite : une valeur entière

affectation qui modifie la valeur de j
- partie gauche : un nom de variable
- partie droite : une expression arithmétique

Expression

- Définitions
  - Une expression est une formule qui combine des ***valeurs***, des ***constantes*** ou des ***variables*** à l'aide d'***opérateurs***
  - Une expression a une valeur et donc un type : c'est la valeur de l'expression après son évaluation
- ≠ types d'expressions
  - Expression arithmétique
    - sur les entiers, sur les réels à virgule flottante
  - Expression de comparaison
  - Expression booléenne
    - sur les booléen

Expressions arithmétiques (entiers)

- Opérations sur les entiers

Opération	en ada	dans un algorithme
addition	`+`	+
soustraction	`-`	-
multiplication	`*`	* ou ×
division entière	`/`	div
reste de la division	`mod`	mod
puissance	`**`	
valeur absolue	`abs`	

- Exemples :

```
2 + max     abs(j)*3     2**3
```

Expressions arithmétiques (entiers)

- division entière **div**
  - définition
    - n div p = partie entière du quotient de la division de n par p ;
  - exemples
    - 7 div 2 = 3 ; 27 div 6 = 4
- modulo **mod**
  - définition
    - n mod p = reste de la division entière de n par p.
    - n = p * (n **div** p) + (n **mod** p)
  - exemples
    - 7 mod 2 = 1 ; 27 mod 6 = 3

Expressions arithmétiques (flottants)

- Opérations sur les flottants

Opération	en ada	dans un algorithme
addition	`+`	+
soustraction	`-`	-
multiplication	`*`	* ou ×
division	`/`	/ ou ÷
puissance	`**`	
valeur absolue	`abs`	

- Exemples :
  - `abs(1+A) + B` `-- (abs(1+a))+b`
  - `-4.0*A**2` `-- -(4.0*(a**2))`
  - `Y**(-3)` `-- parenthèses obligatoires, Y est réel`

Expression de comparaison

- Opérations de comparaison

Opération	en ada	dans un algorithme
égalité	`=`	=
non égalité	`/=`	≠
infériorité stricte (large)	`<` `(<=)`	< (≤)
supériorité stricte (large)	`>` `(>=)`	> (≥
inclusion	`in`	
non inclusion	`not in`	

⚠ Le résultat est de type booléen

- Exemples
  - pi = 3 -- vaut false (faux)
  - 2 ≤ 4 -- vaut true (vrai)

Évaluation d'une expression

- La valeur d'une `valeur` est la `valeur`
  - valeur(3) ➔ 3 (un entier)
  - valeur(3.4) ➔ 3.4 (un réel)
  - valeur(true) ➔ vrai (un booléen)
- La valeur d'une `variable` est la valeur que contient la `variable`
  - si la variable `i` contient `3`
    - alors valeur(`i`) ➔ `3`
  - si la variable `x` contient `3.4`
    - alors valeur(`x`) ➔ `3.4`
  - si la variable `test` contient `true`
    - alors valeur(`test`) ➔ `true`

Évaluation d'une expression

- La valeur d'une `expression` est le résultat du calcul faisant intervenir
  - la `valeur des variables`
  - la `valeur des valeurs`
  - combinées avec les `opérateurs` de l'`expression`
- Exemple

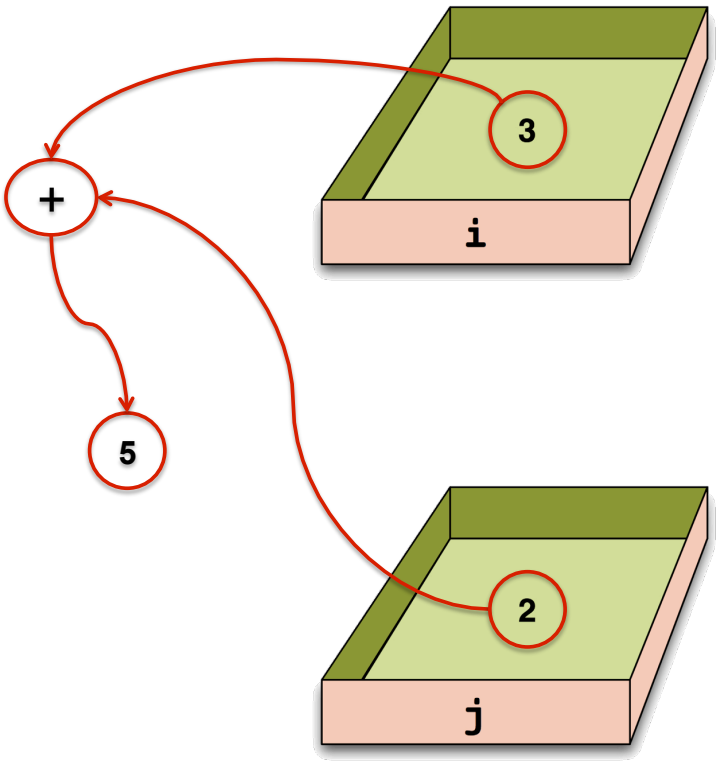
```
i:=3;
j:=2;
j:=i+j;
```

  - évaluation `i+j` et de `j:=i+j` ?
    - `i` vaut `3` ; `j` vaut `2`
    - `i+j` vaut `5`
    - la valeur `5` est rangée dans la variable `j`

Évaluation d'une expression

- Exemple

```
i:=3;
j:=2;
j:=i+j;
```



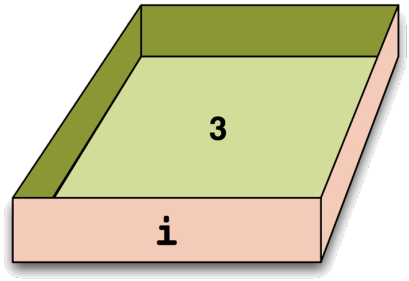
Évaluation d'une expression

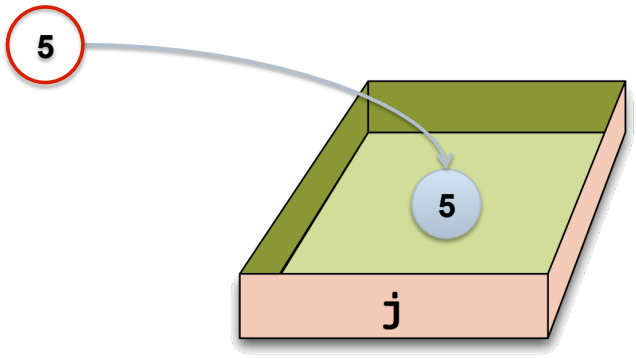
- Exemple

```
i:=3;

j:=2;

j:=i+j;
```





Définition d'un algorithme

- Elle comprend :
  - Un entête
    - type de l'algorithme (fonction, procédure)
    - nom de l'algorithme
    - liste de paramètres
  - Une spécification
    - Ce que fait l'algorithme :
      - précondition
      - postcondition
  - Un corps
    - Comment fait l'algorithme

Entête d'un algorithme

- Un type
  - **fonction** : retourne un résultat
    - Analogie avec une fonction mathématique
      - f(x) → y *{f appliquée à x retourne le résultat y}*
  - **procédure** : ne retourne pas de résultat au sens d'une fonction
    - p(x) *{p appliquée à x ne retourne pas de résultat}*
- Un nom
  - celui de la fonction ou de la procédure
- Une liste de paramètres
  - données sur lesquelles l'algorithme travaille

Entête : exemple

- Une procédure sans paramètre

```
procedure bonjour1 is
```

procédure bonjour1;

- Une procédure avec **paramètre**

```
procedure bonjour3(année: in integer) is
```

procédure bonjour3(d **année : entier**) ;

- Une fonction avec **paramètres** qui retourne une **chaîne de caractères**

```
function titre(sexe, marié:in  character)
        return string(1..12) is
```

fonction titre(d sexe, marié : car) : **chaîne** ;

Spécification d'un algorithme

- précondition
  - condition qui doit être vraie afin que l'algorithme produise l'effet attendu
- postcondition
  - description de l'effet de l'algorithme lorsque la précondition est vraie
- Notation

spécification {précondition} ➔ {postcondition}

Spécification : exemple (**algo**)

- **procédure** bonjour1 ;

  spécification { } ➔ {le message de bonjour à l'utilisateur a été affiché}

- **procédure** bonjour3(d an_cour : entier) ;

  spécification { } ➔ {le bonjour à l'utilisateur et son age ont été affichés}

- **fonction** titre(d sexe, marié : car) : chaîne ;

  spécification *{(sexe='M' ou sexe='F') et (marié='O' ou marié='N')}* ➔ *{résultat = titre correspondant à l'état-civil défini par sexe et marié }*

Pas de précondtion

Spécification : exemple (**ada**)

- **procedure** bonjour1 ;

  --spécification { } -> **{**le message de bonjour à l'utilisateur a été affiché**}**

- **procedure** bonjour3(an_cour: **in** integer) ;

  --spécification { } -> **{**le bonjour à l'utilisateur et son age ont été affichés**}**

Pas de précondtion

- **function** titre(sexe, marié: **in** character)
  return string;

  --spécification *{(sexe='M' ou sexe='F') et (marié='O' ou marié='N')} -> {résultat = titre correspondant à l'état-civil défini par sexe et marié }*

Corps de procédure

- **Ada**

  déclaration des variables

  begin

  liste d'instructions

  end nom_procedure;

- **Algorithme**

  déclaration des variables

  debproc

  liste d'instructions

  finproc;

Corps de fonction

- **Ada**

```
déclaration des
    variables
begin
   liste d'instructions
end nom_fonction;
```

- **Algorithme**

déclaration des variables
debfonc
    liste d'instructions
finfonc;

Programme `bonjour1` (**ada**)

```
with P_ESiut; use P_ESIut;

procedure bonjour1 is
--spécification { } ➔ {le message de
   bonjour à l'utilisateur a été affiché}

   nom : string(1..15);
begin
   ecrire("Comment vous appelez-vous ?");
   lire(nom);
   ecrire_ligne("Bonjour " & nom & " !");
end bonjour1;
```

Déclaration de la procédure principale

Déclaration de la variable

Initialisation de la variable

Consultation de la variable

Procédure `bonjour1` (**ada**)

- On a vu l'utilisation de la procédure `bonjour1` comme programme principal

- Comment faire pour que la procédure `bonjour1` ne soit pas le programme principal ?
  - Procédure locale à une procédure principale


```
with P_Esiut; use P_Esiut;

procedure principal is

   procedure bonjour1 is
   --spécification { } ➔ {le message de
    bonjour à l'utilisateur a été affiché}
     nom : string(1..15);
   begin
     ecrire("Comment vous appelez-vous ?");
     lire(nom);
     ecrire_ligne("Bonjour " & nom & " !");
   end bonjour1;
begin
   bonjour1;
end principal;
```

appel de la procédure

procédure principale

procédure locale à la procédure principale

Procédure `bonjour1` (**ada**)

- On a vu l'utilisation de la procédure `bonjour1` comme programme principal
- Comment faire pour que la procédure `bonjour1` ne soit pas le programme principal ?
  - Procédure locale à une procédure principale
- On verra comment inclure la procédure `bonjour1` dans une bibliothèque (librairie)

procédure **bonjour1** (**algorithme**)

procédure bonjour1 ;

Déclaration de la procédure

spécification *{ }* ➔ *{le message de bonjour à l'utilisateur a été affiché}*

nom : chaîne ;

Déclaration de la variable

debproc

écritexte ("Comment vous appelez-vous ?") ;

litexte (nom) ;

Initialisation de la variable

écritexte ("Bonjour " & nom & " !") ;

finproc ;

Consultation de la variable

procédure **bonjour1** (appel)

- Pour utiliser une procédure, on l'appelle
  - bonjour1 ; *{est une instruction qui fait partie du langage}*
- Trace
  - `Comment vous appelez-vous? **Dupont**`
  - `Bonjour Dupont !`

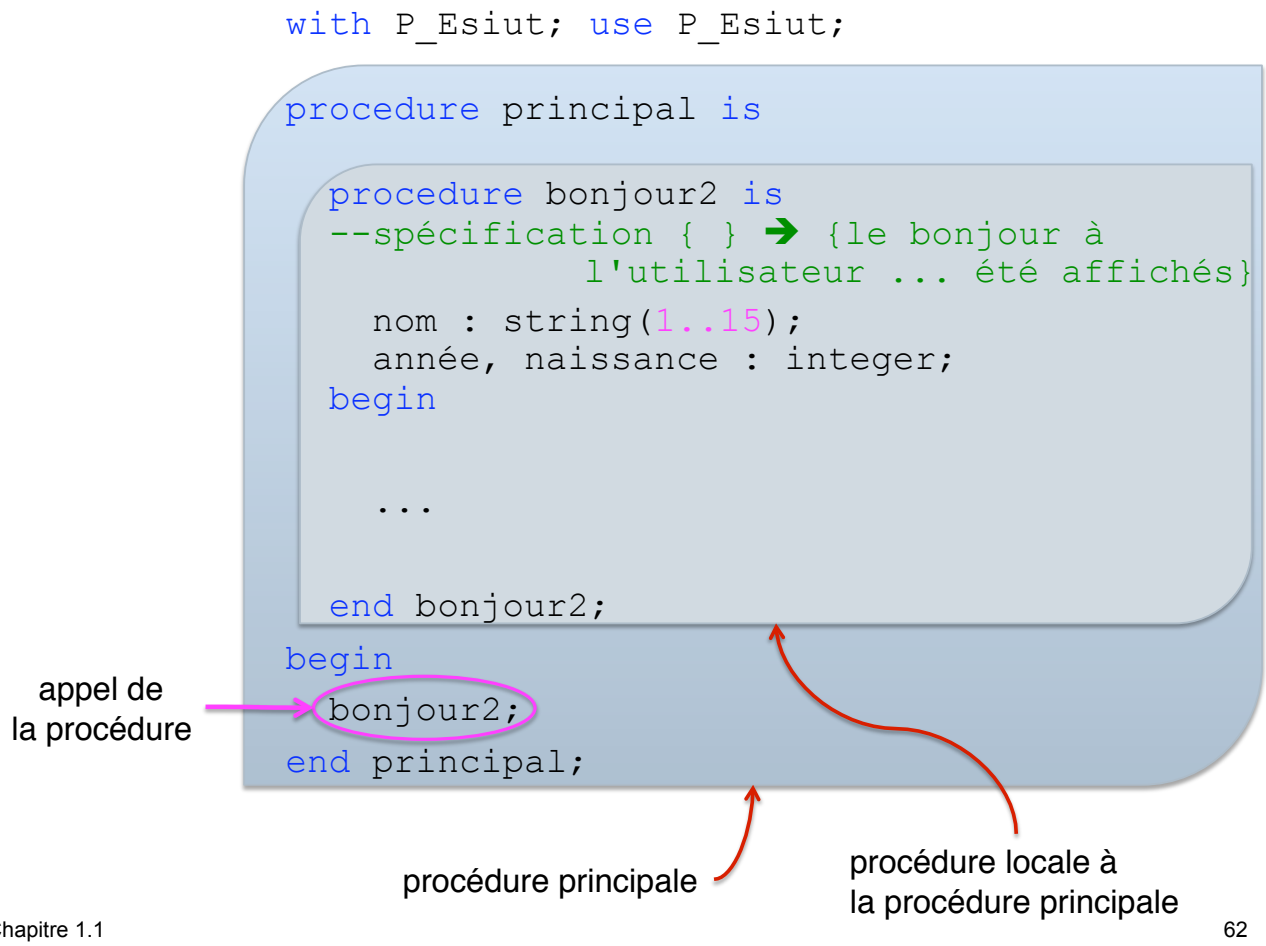
programme `bonjour2` (**ada**)

```
with P_ESiut; use P_ESiut;
procedure bonjour2 is
--spécification { } ➔ {le bonjour à l'utilisateur et son âge
    ont été affichés}
  nom : string(1..15);
  année, naissance : integer;
begin
  ecrire("En quelle année sommes nous?");
  lire(année) ;          --année prend une valeur
  a_la_ligne;
  ecrire("Comment vous appelez-vous?") ;
  lire(nom) ;            --nom prend une valeur
  a_la_ligne;
  ecrire("Votre année de naissance?");
  lire(naissance) ;      --naissance prend une valeur
  a_la_ligne;
  ecrire_ligne("Bonjour "& nom & " !") ;
  ecrire("Vous êtes âgé de ");
  ecrire(année - naissance);  --on ne peut pas utiliser &
  ecrire_ligne(" ans");
end bonjour2;
```

Procédure `bonjour2` (**ada**)

- On a vu l'utilisation de la procédure `bonjour2` comme programme principal

- Comment faire pour que la procédure `bonjour2` ne soit pas le programme principal ?
  - Procédure locale à une procédure principale



Procédure `bonjour2` (ada)

- On a vu l'utilisation de la procédure `bonjour2` comme programme principal
- Comment faire pour que la procédure `bonjour2` ne soit pas le programme principal ?
  - Procédure locale à une procédure principale
- On verra comment inclure la procédure `bonjour2` dans une bibliothèque (librairie)

procédure **bonjour2** (algorithme)

```
procédure bonjour2 ;
spécification { } ➔ {le bonjour à l'utilisateur et son âge ont été
    affichés}
    nom : chaîne ;
    année, naissance : entier ;
debproc
    écritexte ("En quelle année sommes nous?");
    litexte (année) ;                    {année prend une valeur}
    écritexte ("Comment vous appelez-vous?") ;
    litexte (nom) ;                              {nom prend une valeur}
    écritexte("Votre année de naissance?");
    litexte (naissance) ;                {naissance prend une valeur}
    écritexte ("Bonjour " & nom & " !") ;
    écritexte ("Vous êtes âgé de " & année - naissance & " ans") ;
finproc ;
```

consultation de la variable **nom**

consultation des variables **année** et **naissance** puis soustraction

procédure **bonjour2** (appel)

- Pour utiliser une procédure, on l'appelle
  - bonjour2 ; *{est une instruction qui fait partie du langage}*
- Trace
  - `En quelle année sommes nous? 2001`
  - `Comment vous appelez-vous? Dupont`
  - `Votre année de naissance? 1976`
  - `Bonjour Dupont !`
  - `Vous êtes âgé de 25 ans`

Nécessité d'initialiser les variables

procédure corr ;	a	b	c
a, b, c : entier ;	indéfini	indéfini	indéfini

debproc

a := 12 ; b := 5 ;			
c := a - b ;			
a := a + c ;			
b := a ;			

finproc ;

Nécessité d'initialiser les variables

procédure corr ;	a	b	c
a, b, c : entier ;	indéfini	indéfini	indéfini

debproc

a := 5 ;			
c := a + b ;			
d := c - 2 ;			

finproc ;

c ne peut pas être calculé car **b** n'a pas été initialisé !
d n'est même pas déclaré !

Notion de paramètre

- Un **paramètre** d'une fonction ou d'une procédure est …
- …une « **variable locale** » …
- … **visible** (utilisable) seulement à l'**intérieur** de la fonction ou de la procédure

Notion de paramètre

- Exemples
  - `procedure bonjour3(an_cour : in integer) is`
  - **procédure** bonjour3(d an_cour : entier) ;
    - La variable **an_cour** est utilisée par la procédure **bonjour3** et n'est visible qu'à l'intérieur de celle-ci
  - `function titre(sexe, marié : in character) return string is`
  - **fonction** titre(d sexe, marié : car) : chaîne ;
    - Les variables **sexe et marié** sont utilisées par la fonction **titre** et ne sont visibles qu'à l'intérieur de celle-ci

Notion de paramètre

- Paramètre formel
  - Les paramètres donnés dans l'entête sont appelés **paramètres formels**

Lors de la définition d'une fonction ou d'une procédure, on ne connaît pas leur valeur

procédure bonjour3 (ada)

```
procedure bonjour3(an_cour : in integer) is
--spécification {}➔{le bonjour à l'utilisateur
   et son âge ont été affichés}
  nom : string(1..15);
  naissance, âge : integer;
begin
  ecrire("Comment vous appelez-vous ? ") ;
  lire(nom) ;
  ecrire("Votre année de naissance ? ");
  lire(naissance);
  ecrire_ligne("Bonjour " & nom & " !") ;
  âge := an_cour - naissance ;
  ecrire("vous êtes âgé de ")
  ecrire(âge)
  ecrire_ligne(" ans") ;
end bonjour3;
```

paramètre formel

utilisation de la valeur du paramètre formel lors de l'appel

procédure bonjour3 (ada)

- La procédure bonjour3
  - … a un paramètre formel (**an_cour**)
  - … elle ne peut être une procédure principale
  - … elle sera donc appelée à travers un **paramètre effectif** : celui dont la valeur sera utilisée lors du calcul
    - âge := **an_cour** - naissance ;
- 2 appels possibles
  - Un paramètre effectif valeur immédiate
  - Un paramètre effectif variable initialisée

```
with P_Esiut; use P_Esiut;
procedure principal is
   procedure bonjour3(an_cour : in integer) is
      nom : string(1..15);
      naissance, âge : integer;
   begin
      ecrire("Comment vous appelez-vous ? ") ;
      lire(nom) ;
      ecrire("Votre année de naissance ? ");
      lire(naissance);
      ecrire_ligne("Bonjour " & nom & " !") ;
      âge := an_cour - naissance ;
      ecrire("vous êtes âgé de ")
      ecrire(âge)
      ecrire_ligne(" ans") ;
   end bonjour3;
begin
   bonjour3(2008)
end principal;
```

appel de la procédure

valeur immédiate


```
with P_Esiut; use P_Esiut;
procedure principal is
   procedure bonjour3(an_cour : in integer) is
      nom : string(1..15);
      naissance, âge : integer;
   begin
      ecrire("Comment vous appelez-vous ? ") ;
      lire(nom) ;
      ecrire("Votre année de naissance ? ");
      lire(naissance);
      ecrire_ligne("Bonjour " & nom & " !") ;
      âge := an_cour - naissance ;
      ecrire("vous êtes âgé de ")
      ecrire(âge)
      ecrire_ligne(" ans") ;
   end bonjour3;
   an : integer;
begin
   ecrire("en quelle année sommes-nous ?");
   lire(an);
   a_la_ligne;
   bonjour3(an);
end principal;
```

appel de la procédure

variable initialisée

procédure **bonjour3** (**algorithme**)

```
procédure bonjour3(d an_cour : entier) ;
spécification { } ➔ {le bonjour à l'utilisateur et son
    âge ont été affichés}
    nom : chaîne ; naissance, âge : entier ;
debproc
    écritexte ("Comment vous appelez-vous ?") ;
    litexte (nom) ;
    écritexte ("Votre année de naissance?");
    litexte (naissance);
    écritexte ("Bonjour " & nom & " !") ;
    âge := an_cour - naissance ;
    écritexte ("vous êtes âgé de " & âge & " ans") ;
finproc ;
```

paramètre formel

utilisation de la valeur du paramètre formel lors de l'appel

procédure **bonjour3** (paramètre effectif)

- Lors de l'appel, le paramètre est un paramètre **effectif** ; celui avec lequel la fonction ou la procédure s'**effectuera**

- Paramètre effectif **valeur**
  - bonjour3(**2005**);

- Paramètre effectif **variable**
  - Si la variable **an_courant** contient une valeur entière
  - bonjour3(**an_courant**);

Instruction d'itération

- On va souvent devoir faire les mêmes traitements plusieurs fois ➔ **instruction d'itération**

en ada	dans un algorithme

while *condition* **loop**
  instruction1;
  instruction2;
  …
end loop;
	tantque *condition* **faire**
	  instruction1 ;
	  instruction2 ;
	  …
	finfaire;

séquence d'instructions

- Tant que la *condition (expression booléenne)* est vraie, la **séquence d'instructions** est exécutée

Instruction d'itération spécifique Ada

- En Ada l'instruction **loop** peut s'écrire de deux manières différentes

séquence d'instructions de l'itération **exécutée au moins une fois**	séquence d'instructions de l'itération **éventuellement jamais exécutée**

loop
  instruction1;
  instruction2;
  …
  exit when *condition*
end loop;
	while *condition* **loop**
	  instruction1;
	  instruction2;
	  …
	end loop;

séquence d'instructions exécutée *jusqu'à ce que la condition devienne vrai* **Jamais dans un algorithme**	séquence d'instructions exécutée *tant que la condition est vraie*

procédure enfants2 (ada)

```
procedure enfants2 (an_cour : in integer) is
-- spécification {}➔{ les âges des enfants de l'utilisateur
   ont été affichés}
  nom : string(1..15);
  naissance, âge, nbenfants, i : integer;
begin
  ecrire("Combien d''enfants avez-vous ?");
  lire(nbenfants) ;
  i := 1 ;
  while i ≤ nbenfants loop
    ecrire("Nom d''un de vos enfants ?");
    lire(nom); a_la_ligne;
    ecrire("Son année de naissance ?");
    lire(naissance) ; a_la_ligne;
    âge := an_cour - naissance ;
    ecrire (nom & " est âgé de ");
    ecrire (âge); ecrire_ligne(" ans");
    i := i + 1 ;
  end loop;
end enfants2;
```

nbenfants fois

procédure enfants2 (ada)

- La procédure enfants2
  - … a un paramètre formel (**an_cour**)
  - … elle ne peut être une procédure principale
  - … elle sera donc appelée à travers un **paramètre effectif** : celui dont la valeur sera utilisée lors du calcul
    - âge := **an_cour** - naissance ;
- 2 appels possibles
  - Un paramètre effectif valeur immédiate
  - Un paramètre effectif variable initialisée

```
with P_Esiut; use P_Esiut;
procedure principal is
  procedure enfants2(an_cour : in integer) is
    nom : string(1..15);
    naissance, âge, nbenfants, i : integer;
  begin
    ecrire("Combien d''enfants avez-vous ?");
    lire(nbenfants) ;
    i := 1 ;
    while i ≤ nbenfants loop
      ...
      i := i + 1 ;
    end loop;
  end enfants2;
begin
  enfants2(2008)
end principal;
```

appel de la procédure

valeur immédiate


```
with P_Esiut; use P_Esiut;
procedure principal is
  procedure enfants2(an_cour : in integer) is
    nom : string(1..15);
    naissance, âge, nbenfants, i : integer;
  begin
    ecrire("Combien d''enfants avez-vous ?");
    lire(nbenfants) ;
    i := 1 ;
    while i ≤ nbenfants loop
      …
      i := i + 1 ;
    end loop;
  end enfants2;
  an : integer;
begin
  ecrire("en quelle année sommes-nous ?");
  lire(an);
  a_la_ligne;
  enfants2(an)
end principal;
```

appel de la procédure

variable initialisée

procédure **enfants2** (algorithme)

```
procédure enfants2 (d an_cour : entier);
spécification { } ➔ { les âges des enfants de l'utilisateur ont été affichés}
      nom : chaîne ;
      naissance, âge, nbenfants, i : entier ;
debproc
      écritexte ("Combien d"enfants avez-vous ?");
      litexte (nbenfants) ;
      i := 1 ;
      tantque i ≤ nbenfants faire
            écritexte ("Nom d'un de vos enfants ?");
            litexte (nom) ;
            écritexte("Son année de naissance ?");
            litexte (naissance) ;
            âge := an_cour - naissance ;
            écritexte (nom & " est âgé de " & âge & " ans") ;
            i := i + 1 ;
      finfaire ;
finproc ;
```

nbenfants fois

procédure **enfants2** (appel et trace)

- Appel
  - enfants2(**2005**);
- Trace
  - `Combien d'enfants avez-vous ? 2`
  - `Nom d'un de vos enfants ? Pierre`
  - `Son année de naissance ? 1992`
  - `Pierre est âgé de 13 ans`
  - `Nom d'un de vos enfants ? Paul`
  - `Son année de naissance ? 1996`
  - `Paul est âgé de 9 ans`

Instruction conditionnelle

- L'application de traitements peut dépendre de certaines conditions
  ➔ **instruction conditionnelle**
- Idée
  - **si** une condition (exp bool) X est vraie **alors**
    - exécuter des traitements X
  - **sinon**
    - exécuter des traitements Y

Instruction conditionnelle

- Forme générale

en ada	dans un algorithme

if *condition*$_1$ **then**
  seq_instructions$_1$;
{**elseif** *condition*$_{2..n}$ **then**
  seq_instructions$_{2..n}$;}
[**else**
  seq_instructions$_{n+1}$]
end if;
	si condition$_1$ **alors**
	  séquence d'instructions$_1$
	{sinonsi condition$_{2..n}$ **alors**
	  séquence d'instructions$_{2..n}$**}**
	[sinon
	  séquence d'instructions$_{n+1}$**]**
	finsi ;

- Notes
  - **{}** indique que la section **elseif** peut se répéter 0, 1 ou plusieurs fois
  - **[]** indique de la partie **else** est optionnelle

Instruction conditionnelle

Formes simples

en ada	dans un algorithme

`if` *condition*$_1$ `then`
  seq_instructions$_1$;
`else --si` *condition*$_1$ `fause`
  seq_instructions$_2$;
`end if;`
	si condition$_1$ **alors**
	  séquence d'instructions$_1$;
	sinon {si condition$_1$ fausse}
	  séquence d'instructions$_2$;
	finsi ;

`if` *condition*$_1$ `then`
  seq_instructions$_1$;
`end if;`
	si condition$_1$ **alors**
	  séquence d'instructions$_1$;
	finsi ;

si la condition est fausse, on ne fait rien !

Instruction conditionnelle

Formes simples : exemples

en ada	dans un algorithme

`if` *A > B* `then`
  max:=A;
`else`
  max:=B;
`end if;`
	si A > B **alors**
	  max := A ;
	sinon
	  max := B ;
	finsi ;

rechercher le maximum parmi deux valeurs	

`if` *X /= 0* `then`
  Y:=1/X;
`end if;`
	si X ≠ 0 **alors**
	  Y := 1/X ;
	finsi ;

éviter la division par 0	

Expressions booléennes

- Dans ce qui suit nous allons voir des expressions booléennes qui ne font pas seulement intervenir des opérateurs de comparaison

procédure **enfants3**

- On veut afficher l'âge des enfants sans demander le nombre d'enfants au préalable comme dans **enfants2**

- On va demander à l'utilisateur de répondre '*' à la question sur le prénom de l'enfant lorsqu'il n'a plus d'enfant

procédure **enfants3** (ada)

```
procedure enfants3 (an_cour : in integer) is
--spécification {} ➔ {les âges des enfants de l'utilisateur
    ont été affichés}
    nom : string(1..15); stop : boolean;
    âge, naissance : integer;

begin
    stop := false;
    while not stop loop

    end loop;
end enfants3;
```

??

procédure **enfants3** (algorithme)

procédure enfants3 (d an_cour : entier);
spécification *{ }* ➔ *{les âges des enfants de l'utilisateur ont été affichés}*
    nom : chaîne ; stop : booléen ;
    âge, naissance : entier ;

debproc
    stop := faux ;
    tantque non stop **faire**

??

    finfaire ;
finproc ;

Négation (`not` ; non ou ¬)

A	`not` A ; (non A, ¬A)
true (vrai)	`false` (faux)
false (faux)	`true` (vrai)

- (A = faux) *équivalent à* (non A)
- (A = vrai) *équivalent à* A
- (non (non A)) = (non non A) *équivalent à* A

procédure **`enfants3`** (**`ada`** suite)

```
procedure enfants3 (an_cour : in integer) is
--spécification {} ➔ {les âges des enfants de l'utilisateur
    ont été affichés}
    nom : string(1..15); stop : boolean;
    âge, naissance : integer;
begin
    stop := false;
    while not stop loop
       ecrire("Nom d''un de vos enfants ?");
       lire(nom) ;
       if nom /= "*" then
          ecrire("Son année de naissance ? "); lire(naissance);
          âge := an_cour - naissance;
          ecrire(nom & " est âgé de "); ecrire(âge);
          ecrire(" ans"); a_la_ligne;
       else -- nom = "*"
          stop := vrai;
       end if;
    end loop;
end enfants3;
```

tantque stop = *faux*

procédure **enfants3** (algorithme suite)

```
procédure enfants3 (d an_cour : entier);
spécification { } ➔ {les âges des enfants de l'utilisateur ont été affichés}
      nom : chaîne ; stop : booléen ;       âge, naissance : entier ;
debproc
      stop := faux ;
      tantque non stop faire
            écritexte ("Nom d"un de vos enfants ?") ;
            litexte (nom) ;
            si nom ≠ '*' alors
                  écritexte("Son année de naissance?") ;
                  litexte(naissance) ;
                  âge := an_cour - naissance ;
                  écritexte (nom & " est âgé de " & âge & " ans") ;
            sinon
                  stop :=vrai ;
            finsi ;
      finfaire ;
finproc ;
```

tantque stop = *faux*

procédure **enfants4**

- On veut afficher l'âge des enfants sans demander le nombre d'enfants au préalable comme dans **enfants3**

- Mais on ne veux pas traiter plus de 10 enfants pour chaque utilisateur !

procédure **enfants4** (ada)

```
procedure enfants4 (an_cour : in integer) is
--spécification {} ➔ {les âges des enfants de l'utilisateur
    ont été affichés}
  nom : string(1..15); naissance, âge, i : integer;
  stop : boolean;

begin
  stop := false; i := 0 ;
  while not stop and (i < 10) loop
  end loop;
end enfants4;
```

??

procédure **enfants4** (algorithme)

procédure enfants4 (d an_cour : entier);
spécification *{ }* ➔ *{les âges des enfants de l'utilisateur ont été affichés}*
nom : chaîne ; naissance, âge, i : entier ;
stop : booléen ;

debproc
stop := faux ; ***i := 0*** ;
tantque non stop **et** (***i < 10***) **faire**

??

finfaire ;
finproc ;

`and` ; et (intersection) & `or` ; ou (union)

- Comme la négation les opérateurs **et** & **ou** sont des opérateurs booléens

A	B	A et B
vrai	vrai	vrai
vrai	faux	faux
faux	vrai	faux
faux	faux	faux

A	B	A ou B
vrai	vrai	vrai
vrai	faux	vrai
faux	vrai	vrai
faux	faux	faux

Attention

- le **ou logique** est vrai si au moins un parmi A et B est vrai
- ce n'est pas le "fromage **ou** dessert" du restaurant qui est un **ou exclusif**
  - i.e. soit fromage, soit dessert, mais pas les deux !

`and` ; et (intersection) & `or` ; ou (union)

- Note sur l'évaluation du **et** & du **ou**
  - Dans un programme lorsque vous écrivez une expression du genre …
    - `A and B`
    - `A or B`
  - … l'ordre n'est pas important !!
  - En effet …
    - `A and B = B and A`
    - `A or B = B or A`
  - … les opérateurs and & or sont commutatifs !
    - comme l'addition : 7+5 = 5+7
    - ou la multiplication : 7×5 = 5×7

`and` ; et (intersection) & `or` ; ou (union)

Lorsque l'ordre est important,
il faudrait pouvoir le signaler…

- … dans le programme, afin
  - que le lecteur du programme en soit conscient, et
  - d'empêcher le compilateur de faire des optimisations
- … dans l'algorithme, afin
  - que le lecteur en soit conscient

`and` ; et (intersection) & `or` ; ou (union)

- De l'importance de l'ordre d'évaluation :
  - On veut exécuter des traitements traitements1 si l'inverse d'un réel est supérieur strictement à 2 et des traitements2 sinon
  - On ne peut pas écrire …

en ada	dans un algorithme

`if 1/X>2 then`
`  traitements1;`
`else`
`  traitements2;`
`end if;`
	si 1/X>2 **alors**
	  traitements1;
	sinon
	  traitements2;
	finsi;

  - … **il faut éviter la division par 0**

`and` ; et (intersection) & `or` ; ou (union)

- De l'importance de l'ordre d'évaluation :
  - Est-il suffisant d'écrire ?

en ada	dans un algorithme

`if x/=0 and 1/X>2 then`
`  traitements1;`
`else`
`  traitements2;`
`end if;`
	si X≠0 **et** 1/X>2 **alors**
	  traitements1;
	sinon
	  traitements2;
	finsi;

  - **Non !!!!!!**

  - il faut s'assurer que la première condition (X≠0) est évaluée avant la seconde (1/X>2), et qu'en plus …
  - … si la première condition (X≠0) est fausse, alors la seconde condition (1/X>2) n'est pas évaluée

`and` ; et (intersection) & `or` ; ou (union)

- Observation sur l'évaluation du **et** & du **ou**

A	B	A **et** B
vrai	vrai	vrai
vrai	faux	faux
faux	vrai	faux
faux	faux	faux

- et
  - si A est vrai alors
    - A et B = B
  - si A est faux alors
    - A et B = A
    - **B n'est pas examinée**

A	B	A **ou** B
vrai	vrai	vrai
vrai	faux	vrai
faux	vrai	vrai
faux	faux	faux

- ou
  - si A est vrai alors
    - A et B = A
    - **B n'est pas examinée**
  - si A est faux alors
    - A ou B = B

`and` ; et (intersection) & `or` ; ou (union)

- Pour le cas qui nous intéresse :
  - `if x/=0 and 1/X>2 then`
  - **si** X≠0 **et** 1/X>2 **alors**
- On peut noter
  - qu'il ne faut pas évaluer 1/X lorsque X=0
  - que si X≠0 la valeur de la condition est la valeur de 1/X>2
- On introduit alors l'opérateur suivant :

en ada	dans un algorithme
`and then`	et alors

`and then` & et alors

- L'évaluation de l'expression **E** …
  - `A and then B`
  - A **et alors** B
- … se passe comme suit
  - Évaluation de A
    - si A=vrai alors
      - Évaluation de **E** = Évaluation de B
    - si A=faux
      - B n'est pas examinée et Évaluation de **E** = faux

and then & et alors

- L'évaluation de l'expression E …

```
A and then B  =  if A then
                    B ;
                 else
                    false ;
                 end if ;
```

```
A et alors B  =  si A alors
                    B;
                 sinon
                    faux;
                 finsi;
```


and then & et alors

- De l'importance de l'ordre d'évaluation :
  - On veut exécuter des traitements traitements1 si l'inverse d'un réel est supérieur strictement à 2 et des traitements2 sinon
  - On **doit** écrire

en ada	dans un algorithme

`if x/=0 and then 1/X>2 then`
`  traitements1;`
`else`
`  traitements2;`
`end if;`
	si X≠0 **et alors** 1/X>2 **alors**
	  traitements1;
	sinon
	  traitements2;
	finsi;

ce n'est pas de l'optimisation, c'est obligatoire !!

`or else` & ou sinon

- Il se passe la même chose pour le **ou**
- Pour imposer l'ordre d'évaluation on introduit les opérateurs
  - `or else`
  - ou sinon

`or else` & ou sinon

- L'évaluation de l'expression **E** …
  - `A or else B`
  - A **ou sinon** B
- … se passe comme suit
  - Évaluation de A
    - si A=faux alors
      - Évaluation de **E** = Évaluation de B
    - si A=vrai
      - B n'est pas examinée et Évaluation de **E** = vrai

ce n'est pas de l'optimisation, c'est obligatoire !!

`or else` & ou sinon

- L'évaluation de l'expression E …

```
A or else B  =  if A then
                    true ;
                else
                    B ;
                end if ;
```

A ou sinon B = si A alors
vrai;
sinon
B;
finsi;

Négation d'expression booléenne

- Loi de Morgan
  - `not(A and B) = not A or not B`
  - **non**(A **et** B) = **non** A **ou non** B
  - `not (A or B) = not A and not B`
  - **non**(A **ou** B) = **non** A **et non** B
  - `not(A and then B) = not A or else not B`
  - **non**(A **et alors** B) = **non** A **ou sinon non** B
  - `not(A or else B) = not A and then not B`
  - **non**(A **ou sinon** B) = **non** A **et alors non** B

procédure **enfants4** (**ada** suite)

```
procedure enfants4 (an_cour : in integer) is
--spécification {} ➔ {les âges des enfants de l'utilisateur ont
    été affichés}
  nom : string(1..15); naissance, âge, i : integer;
  stop : boolean;
begin
  stop := false; i := 0;
  while not stop and (i < 10) loop
    ecrire("Nom d''un de vos enfants ?");
    lire(nom);
    if nom/="*" then
       ecrire("Son année de naissance ? "); lire(naissance);
       âge := an_cour - naissance ;
       ecrire(nom & " est âgé de "); ecrire(âge);
       ecrire(" ans"); a_la_ligne;
       i := i + 1;
    else
       stop := true;
    end if;
  end loop;
end enfants4;
```

tantque (stop = *faux* et i<10)

procédure **enfants4** (**algorithme** suite)

procédure enfants4 (d an_cour : entier);
spécification { } ➔ *{les âges des enfants de l'utilisateur ont été affichés}*
nom : chaîne ; naissance, âge, i : entier ; stop : booléen ;
debproc
stop := faux ; **i := 0** ;
tantque non stop **et** (**i < 10**) **faire**
écritexte ("Nom d'un de vos enfants ?") ;
litexte (nom) ;
si nom ≠ "*" alors
écritexte("Son année de naissance ? ") ;
litexte (naissance) ;
âge := an_cour - naissance ;
écritexte (nom & " est âgé de " & âge & " ans") ;
i := i + 1 ;
sinon
stop := vrai ;
finsi ;
finfaire ;
finproc ;

tantque (stop = *faux* et i<10)

Assertions

- Fournir au lecteur des informations sur l'algorithme sous forme d'expressions logiques ou de conditions toujours vraies
- Notation
  - écrites en italiques et entre accolades **{…}**
  - **non** *se note* ¬
  - **et** *se note* **,** *ou* ∧
  - **ou** *se note* ∨
- Exemple simple
  - *{a=5 , b<6}*

Assertions

- Exemple complexe
  - *{ (a=5 , b=6) ∨ (c<7) }*

    Expression1 *Expression2*

  - Vrai dans trois cas :
    - a=5 , b=6 , c<7
      - *Expression1* ***et*** *Expression2*
    - a=5 , b=6 , c≥7
      - *Expression1* ***et*** *¬Expression2*
    - (a≠5 ∨ b≠6) , (c<7)
      - *¬Expression1* ***et*** *Expression2*

procédure **enfants4** (algo & assertions)

```
procédure enfants4 (d an_cour : entier);
spécification { } ➔ {les âges des enfants de l'utilisateur ont été affichés}
      nom : chaîne ; naissance, âge, i : entier ; stop : booléen ;
debproc
      stop := faux ; i := 0 ;
      tantque non stop et (i < 10) faire
            écritexte ("Nom d'un de vos enfants ?") ;
            litexte (nom) ;
            si nom ≠ "*" alors
                  écritexte("Son année de naissance ?") ;
                  litexte (naissance) ;
                  âge := an_cour - naissance ;
                  écritexte (nom & " est âgé de " & âge & " ans") ;
                  i := i + 1 ;
            sinon   { nom='*' }
                  stop := vrai ;
            finsi ;
      finfaire ;   { ???}
finproc ;
```

Où mettre les assertions ?

Assertion de sortie d'un tantque

tantque non stop **et** (i < 10) **faire**

- Condition d'itération
  - ¬stop , (i < 10) *peut être notée comme une assertion!*
- Après le **finfaire** c'est la **négation** de la condition d'itération qui est vérifiée !
  - ¬ (¬stop , (i < 10))
  - ➔ ¬¬stop v ¬(i < 10) *Morgan*
  - ➔ stop v (i ≥ 10)
  - mais i > 10 est impossible !

La première fois que (i<10) est faux c'est lorsque (i=10)

Assertion de sortie d'un tantque

tantque non stop **et** (i < 10) **faire**

- Condition d'itération
  - ¬stop , (i < 10) *peut être notée comme une assertion!*
- Après le **finfaire** c'est la **négation** de la condition d'itération qui est vérifiée !
  - ¬ (¬stop , (i < 10))
  - ➔ ¬¬stop ∨ ¬(i < 10) *Morgan*
  - ➔ stop ∨ (i ≥ 10)
  - mais i > 10 est impossible !
  - ➔ stop ∨ (i =10)

procédure **enfants4** (algo & assertions)

```
procédure enfants4 (d an_cour : entier);
spécification { } ➔ {les âges des enfants de l'utilisateur ont été affichés}
      nom : chaîne ; naissance, âge, i : entier ; stop : booléen ;
debproc
      stop := faux ; i := 0 ;
      tantque non stop et (i < 10) faire
            écritexte ("Nom d'un de vos enfants ?") ;
            litexte (nom) ;
            si nom ≠ "*" alors
                  écritexte("Son année de naissance ?") ;
                  litexte (naissance) ;
                  âge := an_cour - naissance ;
                  écritexte (nom & " est âgé de " & âge & " ans") ;
                  i := i + 1 ;
            sinon    { nom='*' }
                  stop := vrai ;
            finsi ;
      finfaire ;    { stop ∨ (i=10) }
finproc ;
```

procédure **enfants4** (**ada** & assertions)

```
procedure enfants4 (an_cour : in integer) is
--spécification {} ➔ {les âges des enfants de l'utilisateur ont
    été affichés}
  nom : string(1..15); naissance, âge, i : integer;
  stop : boolean;
begin
  stop := false; i := 0;
  while not stop and (i < 10) loop
    ecrire("Nom d''un de vos enfants ?");
    lire(nom);
    if nom/="*" then
       ecrire("Son année de naissance ? "); lire(naissance);
       âge := an_cour - naissance ;
       ecrire(nom & " est âgé de "); ecrire(âge);
       ecrire(" ans"); a_la_ligne;
       i := i + 1;
    else  --{nom="*"}
       stop := true;
    end if;
  end loop;  --{stop v (i = 10)}
end enfants4;
```


procédure **bonjour4** (type caractère)

- On veut écrire une procédure qui puisse dire bonjour en utilisant la civilité de l'utilisateur
  - Monsieur
  - Madame
  - Mademoiselle
- en fonction de son sexe et son état marital
  - Monsieur, pour un homme
  - Madame, pour une femme mariée
  - Mademoiselle, pour une femme célibataire

procédure **bonjour4** (**ada**, caractère)

```
procedure bonjour4 is
--spécification {}➔{ le message de bonjour personnalisé a été
    affiché}
  nom : string(1..15) ; sexe, marié : character ;
begin
  ecrire("Comment vous appelez-vous ?");
  lire(nom);
  ecrire("Quel est votre sexe (M ou F) ?") ;
  lire(sexe) ;
  if sexe = 'M' then    --{sexe='M'}
    ecrire("Bonjour, Monsieur " & nom & " !");
  else                  --{sexe='F'}
    ecrire("Etes-vous mariée (O ou N) ?") ;
    lire(marié) ;
    if marié = 'O' then    --{sexe='F', marié='O'}
      ecrire("Bonjour, Madame " & nom & " !") ;
    else                   --{sexe='F', marié='N'}
      ecrire("Bonjour, Mademoiselle " & nom & " !") ;
    end if ;
  end if ;
end bonjour4 ;
```


procédure **bonjour4** (algo, caractère)

procédure bonjour4 ;
spécification { } ➔ { *le message de bonjour personnalisé a été affiché*}
    nom : chaîne ; **sexe**, **marié** : **car** ;
debproc
    écritexte ("Comment vous appelez-vous ?") ;
    litexte (nom) ;
    écritexte ("Quel est votre sexe (M ou F) ?") ;
    litexte (**sexe**) ;
    si **sexe = 'M'** **alors** *{sexe='M'}*
        écritexte ("Bonjour, Monsieur " & nom & " !") ;
    sinon *{sexe='F'}*
        écritexte("Etes-vous mariée (O ou N) ?") ;
        litexte (**marié**) ;
        si **marié = 'O'** **alors** *{sexe='F',marié='O'}*
            écritexte ("Bonjour, Madame " & nom & " !") ;
        sinon *{sexe='F', marié='N'}*
            écritexte("Bonjour, Mademoiselle " & nom & " !") ;
        finsi ;
    finsi ;
finproc ;

Comment ferait-on en demandant sexe et marié à tous les utilisateurs ?

procédure **bonjour5** (algorithme)

procédure bonjour5 ;
spécification *{ }* ➔ *{ le message de bonjour personnalisé a été affiché}*
    nom : chaîne ; sexe, marié : car ;
debproc
    écritexte ("Comment vous appelez-vous ?") ;
    litexte (nom) ;
    écritexte ("Quel est votre sexe (M ou F) ?") ;
    litexte (sexe) ;
    écritexte ("Etes-vous marié(e) (O ou N) ?") ;
    litexte (marié) ;

finproc ;

Instruction conditionnelle (suite)

- Lorsqu'on a le choix entre plusieurs conditions on peut utiliser une forme plus générale de l'instruction conditionnelle

si condition$_1$ **alors**
    suite d'instructions **1**
sinonsi condition$_2$ **alors**
    suite d'instructions **2**
•••
sinonsi condition$_n$ **alors**
    suite d'instructions **n**
sinon
    suite d'instructions **n+1**
finsi ;

Si condition$_1$ **est vraie alors**
effectuer suite d'instructions 1 ;
sinon si condition$_2$ **est vraie alors**
effectuer suite d'instruction 2
…

Facultatif

procédure **bonjour5** (algorithme)

```
procédure bonjour5 ;
spécification { } ➔ { le message de bonjour personnalisé a été affiché}
      nom : chaîne ; sexe, marié : car ;
debproc
      écritexte ("Comment vous appelez-vous ?") ;
      litexte (nom) ;
      écritexte ("Quel est votre sexe (M ou F) ?") ;
      litexte (sexe) ;
      écritexte ("Etes-vous marié(e) (O ou N) ?") ;
      litexte (marié) ;
      si sexe = 'M' alors {sexe='M'}
      |    écritexte ("Bonjour, Monsieur " & nom & " !") ;
      sinonsi marié='O' alors {sexe='F', marié='O'}
      |    écritexte ("Bonjour, Madame ' & nom & " !") ;
      sinon {sexe='F', marié='N'}
      |    écritexte("Bonjour, Mademoiselle" & nom & " !") ;
      finsi ;
finproc ;
```


procédure **bonjour5** (ada)

```
procedure bonjour5 is
--spécification { } ➔ { le message de bonjour
    personnalisé a été affiché}
  nom : string(1..15); sexe, marié : character;
begin
  ecrire("Comment vous appelez-vous ?");
  lire(nom);
  ecrire("Quel est votre sexe (M ou F) ?");
  lire(sexe);
  ecrire("Etes-vous marié(e) (O ou N) ?");
  lire(marié);
  if sexe = 'M' then        --{sexe='M'}
    ecrire("Bonjour, Monsieur " & nom & " !");
  elseif marié='O' alors    --{sexe='F', marié='O'}
    ecrire("Bonjour, Madame " & nom & " !");
  else                      --{sexe='F', marié='N'}
    ecrire("Bonjour, Mademoiselle" & nom & " !");
  end if;
end bonjour5;
```

Première fonction : **titre**

- On veut écrire une fonction à deux paramètres formels de type caractère …
  - le sexe
  - l'état marital

- … qui retourne (délivre) comme résultat une chaîne de caractères de civilité :
  - Monsieur
  - Madame
  - Mademoiselle

Première fonction : `titre` (ada)

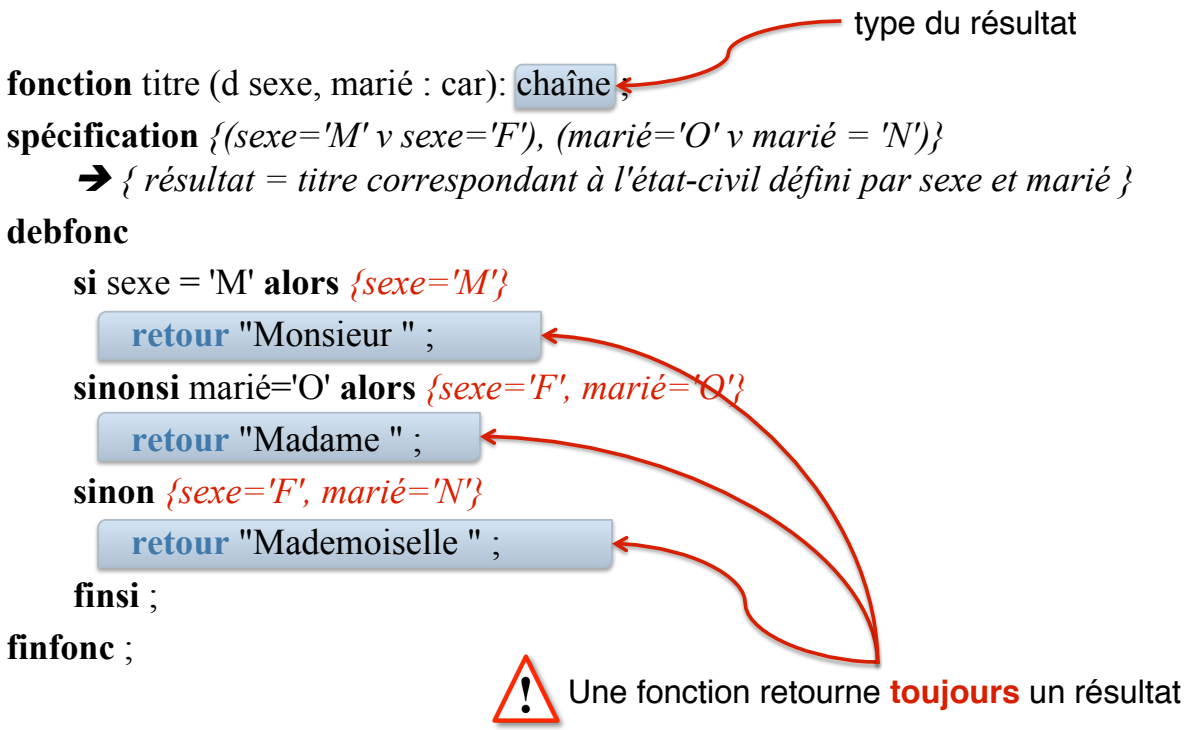
type du résultat

```
function titre (sexe, marié : in character)
                                  return string is
--spécification {(sexe='M' v sexe='F'), (marié='O' v
   marié = 'N')}➔{résultat = titre correspondant à
   l'état-civil défini par sexe et marié}
begin
  if sexe = 'M' then          --{sexe='M'}
    return("Monsieur    ");
  elseif marié='O' then       --{sexe='F', marié='O'}
    return("Madame      ");
  else                        --{sexe='F', marié='N'}
    return("Mademoiselle");
  end if;
end titre;
```

Une fonction retourne **toujours** un résultat

Toutes les chaînes retournées par la fonction doivent avoir la même longueur

Première fonction : **titre** (algorithme)



L'instruction `return`, retour

- … peut retourner une valeur de n'importe quel type
  - Exemples
    - retour 3 ;
    - retour "Monsieur" ;
    - retour sexe ;
      - si sexe est une variable, retour de sa valeur
- … interrompt l'exécution de la fonction

Fonction incorrecte (exemples)

```
fonction majeur (d âge : entier) : booléen ;
debfonc
    si âge < 18 alors
        retour faux ;
    finsi ;
    { si age ≥ 18 rien n'est prévu !!! }
finfonc ;
```

```
fonction majeur (d âge : entier) : booléen ;
debfonc
    retour vrai ; { exécution interrompue ici }
    si âge < 18 alors
        retour faux ;
    finsi ;
finfonc ;
```


Fonction correcte (exemples)

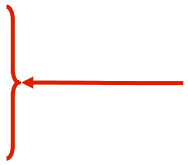
```
fonction majeur1 (d âge : entier) : booléen ;
debfonc
    si âge < 18 alors
        retour faux ;
    finsi ;
    { age ≥ 18 }
    retour vrai ;
finfonc ;
```

```
fonction majeur2 (d âge : entier) : booléen ;
debfonc
    si âge < 18 alors
        retour faux ;
    sinon { age ≥ 18 }
        retour vrai ;
    finsi ;
finfonc ;
```

Fonction correcte (exemples)

```
fonction majeur3 (d âge : entier) : booléen ;
debfonc
      si âge ≥ 18 alors
            retour vrai ;
      sinon { age < 18 }
            retour faux ;
      finsi ;
finfonc ;
```

```
fonction majeur4 (d âge : entier) : booléen ;
debfonc
      retour âge ≥ 18 ;
finfonc ;
```



Note : une fonction qui retourne un booléen est un prédicat

Appel de la fonction `titre` (`ada`)

- La fonction `titre`
  - … a 2 paramètres formels (**`sexe, marié`**)
  - … retourne toujours un résultat (**`string`**)
  - … sera appelée avec 2 **paramètres effectifs** dont les valeurs seront utilisées lors du calcul
- Lors de l'appel il faudra faire quelque chose du résultats
- 1 appel utile
  - 2 paramètres effectifs variable initialisée

```
with P_Esiut; use P_Esiut;
procedure principal is
  function titre (sexe, marié : in character)
                                        return string is
  --spécification {(sexe='M' v sexe='F'), (marié='O' v
    marié = 'N')}➔{résultat = titre correspondant à
    l'état-civil défini par sexe et marié}
  begin
    …
  end titre;
  nom, civilité : string(1..15); s, m : character;
begin
  ecrire("Comment vous appelez-vous ?");
  lire(nom);
  ecrire("Quel est votre sexe (M ou F) ?");
  lire(s);
  ecrire("Etes-vous marié(e) (O ou N) ?");
  lire(m);
  civilité := titre(s, m);
  ecrire_ligne("Bonjour " & civilité & " " & nom);
end principal;
```

appel de la fonction `titre` avec deux paramètres effectifs


```
with P_Esiut; use P_Esiut;
procedure principal is
  function titre (sexe, marié : in character)
                                        return string is
  --spécification {(sexe='M' v sexe='F'), (marié='O' v
    marié = 'N')}➔{résultat = titre correspondant à
    l'état-civil défini par sexe et marié}
  begin
    …
  end titre;
  nom, civilité : string(1..15); s, m : character;
begin
  ecrire("Comment vous appelez-vous ?");
  lire(nom);
  ecrire("Quel est votre sexe (M ou F) ?");
  lire(s);
  ecrire("Etes-vous marié(e) (O ou N) ?");
  lire(m);
  civilité := titre(s, m);
  ecrire_ligne("Bonjour " & civilité & " " & nom);
end principal;
```

le résultat de la fonction **`titre`** est rangée dans la variable **`civilité`**

Appel de la fonction **titre** (algorithme)

procédure bonjour6 ;
spécification { } ➔ *{le message de bonjour personnalisé a été affiché}*
    nom : chaîne ; s, m : car ;
debproc
    écritexte ("Comment vous appelez-vous ?") ;
    litexte (nom) ;
    écritexte ("Quel est votre sexe (M ou F) ?") ;
    litexte (**s**) ;
    écritexte ("Etes-vous marié (O ou N) ?") ;
    litexte (**m**) ;
    écritexte ("Bonjour, " & **titre(s, m)** & nom & " !") ;
finproc ;

Appel de la fonction **titre** avec les paramètres effectifs **s** et **m**.
La fonction retourne une chaîne de caractères.

Types de paramètre

- On a vu

ada	algorithme

`function majeur4(âge : in integer) return boolean is`
`begin`
`  return(age >= 18) ;`
`end majeur4;`
	fonction majeur4 (**d** âge : entier)
	: booléen ;
	debfonc
	retour age ≥ 18 ;
	finfonc ;

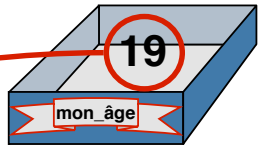
?????????

Type de paramètre

- Dans la fonction majeur,

  âge est un **paramètre donnée**

  - Lors du passage du paramètre effectif, on passe **la valeur** de celui-ci
  - Exemple
    - mon_âge := 19 ;
    - maj := majeur4(mon_âge) ;
      - âge vaut 19 dans majeur4



Paramètre donnée

- Il a obligatoirement une valeur initiale

  ⚠ **sa valeur ne peut qu'être consultée !**

  ➔ Il ne peut pas figurer en partie gauche d'affectation

- Il se comporte comme une **constante**

Question ?

- On voudrait pouvoir regrouper les questions posées à l'utilisateur et fournir 3 résultats (nom, sexe & état-civil)
  - Impossible avec une fonction !
    - Rappel : une fonction ne produit qu'un résultat
- Le seul autre type d'algorithme dont on dispose est la procédure
  - Il faudrait qu'une procédure puisse rendre un (ou plusieurs) résultat(s)
    - **N'est-ce pas contradictoire ????**

Un autre type de paramètres

- Il faudrait que le paramètre se comporte comme une « variable à laquelle on peut donner une valeur »
  - La valeur doit être visible à l'extérieur de la procédure
- Idée :
  - ne pas transmettre la valeur (le contenu), mais transmettre la boîte (la variable) définie par l'appelant
- **Paramètre résultat**

procédure **interrogatoire** (algorithme)

```
procédure interrogatoire ( r nom : chaîne ; r sexe, marié : car) ;
debproc
    écritexte ("Comment vous appelez-vous ?") ;
    litexte (nom) ;
    écritexte ("Quel est votre sexe (M ou F) ?") ;
    litexte (sexe) ;
    écritexte ("Etes-vous marié (O ou N) ?") ;
    litexte (marié) ;
finproc ;
```


procédure **bonjour7** (algorithme)

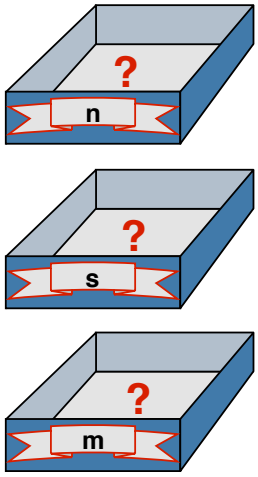
```
procédure bonjour7 ;
spécification {} ➔ {}
    n : chaîne ; s, m : car
debproc
    interrogatoire(n, s, m) ;
    écritexte("Bonjour, " & titre(s, m) & n & "!") ;
finproc ;
```

```
procédure interrogatoire ( r nom : chaîne ; r sexe, marié : car) ;
debproc
    écritexte ("Comment vous appelez-vous ?") ;
    litexte (nom) ;
    écritexte ("Quel est votre sexe (M ou F) ?") ;
    litexte (sexe) ;
    écritexte ("Etes-vous marié(e) (O ou N) ?") ;
    litexte (marié) ;
finproc ;
```

procédure **bonjour7** (trace)

procédure bonjour7 ;
spécification {} ➔ {}
➔ n : chaîne ; s, m : car
debproc

finproc ;



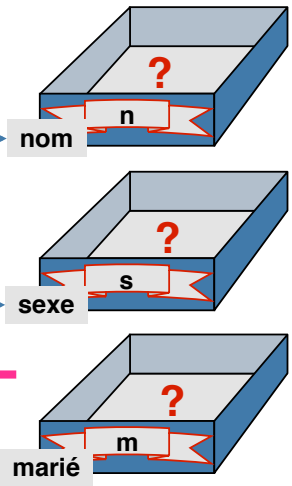
procédure interrogatoire (**r** nom : chaîne ; **r** sexe, marié : car) ;
debproc

finproc ;

procédure **bonjour7** (trace)

procédure bonjour7 ;
spécification {} ➔ {}
n : chaîne ; s, m : car
debproc
➔ **interrogatoire(n, s, m) ;**

finproc ;



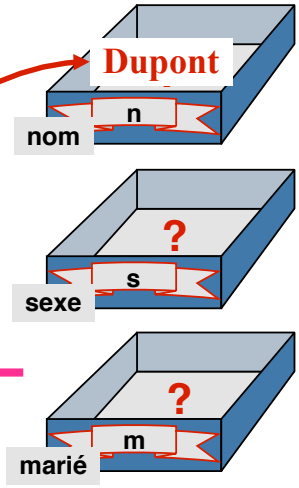
procédure interrogatoire (**r** nom : chaîne ; **r** sexe, marié : car) ;
debproc

finproc ;

procédure **bonjour7** (trace)

procédure bonjour7 ;
spécification {} ➔ {}
    n : chaîne ; s, m : car
debproc
    interrogatoire(n, s, m) ;

finproc ;



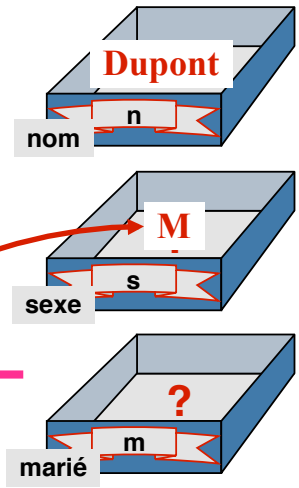
procédure interrogatoire (r nom : chaîne ; r sexe, marié : car) ;
debproc
    écritexte ("Comment vous appelez-vous ?") ;
➔ **litexte** (nom) ; `Dupont`

finproc ;

procédure **bonjour7** (trace)

procédure bonjour7 ;
spécification {} ➔ {}
    n : chaîne ; s, m : car
debproc
    interrogatoire(n, s, m) ;

finproc ;



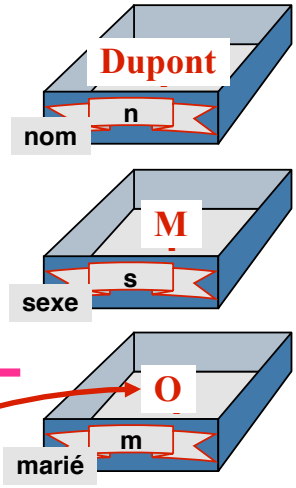
procédure interrogatoire (r nom : chaîne ; r sexe, marié : car) ;
debproc
    écritexte ("Comment vous appelez-vous ?") ;
    litexte (nom) ;
    écritexte ("Quel est votre sexe (M ou F) ?") ;
➔ **litexte** (sexe) ; `M`

finproc ;

procédure **bonjour7** (trace)

```
procédure bonjour7 ;
spécification {} ➔ {}
    n : chaîne ; s, m : car
debproc
    interrogatoire(n, s, m) ;

finproc ;
```

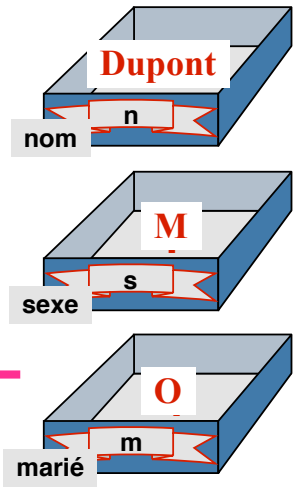



```
procédure interrogatoire ( r nom : chaîne ; r sexe, marié : car) ;
debproc
    écritexte ("Comment vous appelez-vous ?") ;
    litexte (nom) ;
    écritexte ("Quel est votre sexe (M ou F) ?") ;
    litexte (sexe) ;
    écritexte ("Etes-vous marié(e) (O ou N) ?") ;
➔   litexte (marié) ;      O
finproc ;
```


procédure **bonjour7** (trace)

```
procédure bonjour7 ;
spécification {} ➔ {}
    n : chaîne ; s, m : car
debproc
    interrogatoire(n, s, m) ;

finproc ;
```

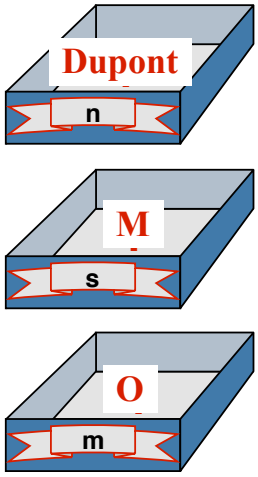



```
procédure interrogatoire ( r nom : chaîne ; r sexe, marié : car) ;
debproc
    écritexte ("Comment vous appelez-vous ?") ;
    litexte (nom) ;
    écritexte ("Quel est votre sexe (M ou F) ?") ;
    litexte (sexe) ;
    écritexte ("Etes-vous marié(e) (O ou N) ?") ;
    litexte (marié) ;
➔ finproc ;
```

procédure **bonjour7** (trace)

```
procédure bonjour7 ;
spécification {} ➔ {}
    n : chaîne ; s, m : car
debproc
    interrogatoire(n, s, m) ;

finproc ;
```

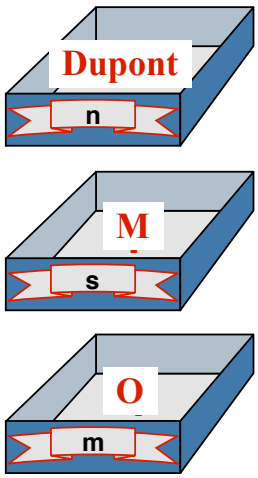



```
procédure interrogatoire ( r nom : chaîne ; r sexe, marié : car) ;
debproc
    écritexte ("Comment vous appelez-vous ?") ;
    litexte (nom) ;
    écritexte ("Quel est votre sexe (M ou F) ?") ;
    litexte (sexe) ;
    écritexte ("Etes-vous marié(e) (O ou N) ?") ;
    litexte (marié) ;
➔ finproc ;
```


procédure **bonjour7** (trace)

```
procédure bonjour7 ;
spécification {} ➔ {}
    n : chaîne ; s, m : car
debproc
    interrogatoire(n, s, m) ;
➔   écritexte("Bonjour," & titre(s, m) & n & "!") ;
finproc ;
```

```
Bonjour, Monsieur Dupont !
```




```
procédure interrogatoire ( r nom : chaîne ; r sexe, marié : car) ;
debproc
    écritexte ("Comment vous appelez-vous ?") ;
    litexte (nom) ;
    écritexte ("Quel est votre sexe (M ou F) ?") ;
    litexte (sexe) ;
    écritexte ("Etes-vous marié(e) (O ou N) ?") ;
    litexte (marié) ;
finproc ;
```

procédure **interrogatoire** (ada)

```
procedure interrogatoire (
        nom : out string ;
        sexe, marié : out character) is
begin
  ecrire("Comment vous appelez-vous ?");
  lire(nom);
  ecrire ("Quel est votre sexe (M ou F) ?");
  lire(sexe);
  ecrire('Etes-vous marié(e) (O ou N) ?");
  lire(marié);
end interrogatoire;
```


procédure **bonjour7** (ada)

```
procedure bonjour7 is
    n : string(1..15); s, m : character;
begin
    interrogatoire(n, s, m);
    ecrire("Bonjour, " & titre(s, m) & n & " !");
end bonjour7;
```

- La procédure bonjour7 peut être une procédure principale ou une procédure locale à une procédure principale.

```
with P_Esiut; use P_Esiut;
procedure bonjour7 is
  procedure interrogatoire (
        nom : out string ;
        sexe, marié : out character) is
  begin
    ecrire("Comment vous appelez-vous ?");
    lire(nom);
    ecrire ("Quel est votre sexe (M ou F) ?");
    lire(sexe);
    ecrire('Etes-vous marié(e) (O ou N) ?");
    lire(marié);
  end interrogatoire;
  n : string(1..15); s, m : character;
begin
  interrogatoire(n, s, m);
  ecrire("Bonjour, " & titre(s, m) & n & " !");
end bonjour7;
```



```
with P_Esiut; use P_Esiut;
procedure principal is
  procedure interrogatoire (
        nom : out string ;
        sexe, marié : out character) is
  begin
    ecrire("Comment vous appelez-vous ?");
    lire(nom);
    ecrire ("Quel est votre sexe (M ou F) ?");
    lire(sexe);
    ecrire('Etes-vous marié(e) (O ou N) ?");
    lire(marié);
  end interrogatoire;
  procedure bonjour7 is
    n : string(1..15); s, m : character;
  begin
    interrogatoire(n, s, m);
    ecrire("Bonjour, " & titre(s, m) & n & " !");
  end bonjour7;
begin
  bonjour7
end principal;
```

Paramètre résultat

- On considère qu'il n'a pas de valeur initiale

  ⚠ **sa valeur ne peut pas être consultée !**

  ➔ Il ne peut pas figurer en partie droite d'affectation, ni dans un test

- Il se comporte comme une « variable à laquelle on doit donner une valeur » dans la procédure

⚠ **Il doit recevoir une valeur dans tous les cas !**

Paramètre donnée-résultat

⚠ Il se comporte comme une **variable ordinaire**

- Il a une valeur lors de l'appel
  - i.e. le paramètre effectif a une valeur
- La procédure peut le consulter
  - Comme le paramètre donnée
- La procédure peut modifier sa valeur
  - Comme le paramètre résultat

procédure **permut** (ada)

```
procedure permut (a , b : in out integer) is
 --spécification {a=x , b=y} ➔ {a=y , b=x}
   c : integer;
begin                -- a = x , b = y
   c := a ;          -- c = x
   a := b ;          -- a = y
   b := c ;          -- b = x
                     -- a = y , b = x
end permut;
```

- Appel de `permut` :
  - `n1:=25 ; n2:=17;`
  - `permut(n1 , n2);`

procédure **permut** (algorithme)

procédure permut (**dr** a , b : entier) ;
spécification { $a = x , b = y$ } ➔ { $a = y , b = x$ }
    c : entier ;
debproc { $a = x , b = y$ }
    c := a ; { $c = x$ }
    a := b ; { $a = y$ }
    b := c ; { $b = x$ }
  { $a = y , b = x$ }
finproc ;

- Appel de permut :
  - n1 := 25 ; n2 := 17 ;
  - permut (n1 , n2);

Bilan sur les paramètres

- **Fonction** : seulement des paramètres **d**
- **Procédure** : des paramètres **d**, **r**, **dr**

	d `in` donnée	**r** `out` résultat	**dr** `in out` donnée-résutat
valeur initiale	**doit être définie**	**quelconque**	**doit être définie**
valeur finale	**identique à la valeur initiale**	**définie (affectée dans l'algo.)**	**définie (affectée dans l'algo.)**
paramètre effectif	**expression ou variable initialisée**	**variable**	**variable initialisée**
contrainte	**jamais modifié**	**jamais consulté**	**aucune**