L3 SID APU Algorithmique, Programmation, Unix

Thomas Pellegrini, équipe SAMoVA, IRIT,
thomas.pellegrini@irit.fr

IRIT - UPS

2016-2017

Présentation de l'UE

début : 15 septembre, fin : 18 octobre

- 6 cours de 2h
- 3 TD de 2h
- 6 TP de 2h, 2 groupes d'étudiants

Modalités de Contrôle des Connaissances

- Examen : Jeudi 3 novembre, 7h45

Syllabus

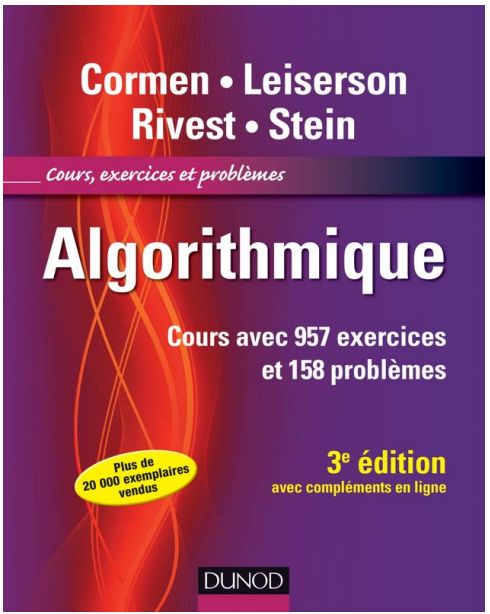
Objectifs

- Connaître et manipuler les systèmes UNIX (Linux)
- Connaître les principales structures de données pour stocker des données
- Initiation à l'algorithmique et à la programmation impérative pour résoudre un problème
- Intégrer l'équation : programmes = algorithmes + structures de données

Syllabus

Bibliographie

- Cormen, Leiserson, Rivest, Stein, Algorithmique, 3e édition, DUNOD, 2010



Algorithme, algorithmique

Définition : algorithme

Un algorithme est la description, pour un exécutant donné, d'une méthode de résolution d'un problème en termes d'une suite ordonnée d'actions fournissant le résultat cherché

Exemple 1 : trouver son chemin

— Pourriez-vous m'indiquer le chemin de la gare, s'il vous plait ?
— Oui, bien sûr : vous allez tout droit jusqu'au prochain carrefour, vous prenez à gauche au carrefour et ensuite la troisième à droite, et vous verrez la gare juste en face de vous.
— Merci !

Tâche à réaliser ? Instructions ? Données manipulées ?

Exemple 2 : recette de cuisine

- Battre les oeufs en omelette
- Mélanger la farine avec le sel puis faire un puit au centre. Y ajouter les oeufs et mélanger à l'aide d'un fouet
- Ajouter le lait petit à petit en mélangeant constamment pour éviter la formation de grumeaux. Fouetter énergiquement
- Si la pâte présente des grumeaux, la mixer ou la passer à travers une passoire fine
- Laisser reposer environ 15 minutes

Tâche à réaliser ? Instructions ? Données manipulées ?

Exemple 3 : plus grand commun diviseur PGCD

Algorithme d'Euclide (3e siècle avant J.C.) : calculer le pgcd de a et b ?

1. Diviser a par b et noter le reste r
2. b devient a et r devient a
3. Répéter 1) et 2) tant que r est plus grand que zéro
4. Le PGCD est le dernier reste r non nul

Tâche à réaliser ? Instructions ? Données manipulées ?

Exemple 4 : distance d'édition — *edit distance*

- Applications : reconnaissance de caractères, correction automatique, séquençage génomique, etc.
- Distance d'édition : nombre minimal d'opérations requises pour transformer une chaîne de caractères en une autre
- Dist(n e i g e, p i g e) = 2 : une suppression ('n') et une substitution ('e' $\rightarrow$ 'p')
- Trois opérations possibles : suppression, substitution, insertion
- Algorithme inventé il y a plus de 40 ans : algorithme de Wagner–Fischer
- Un travail récent du MIT démontre qu'il n'est pas possible d'améliorer cet algorithme qui est quadratique en temps : Backurs & Indyk, 2014
- Pour en savoir plus, très bonne description : blog Matlab

Remarques

- La notion d'algorithme est très ancienne et a précédé celle d'ordinateur
- Définition d'un ≪ programme ≫ : codage d'un algorithme dans un langage de programmation compréhensible et exécutable par un ordinateur
- Un algorithme est indépendant du langage de programmation utilisé

  $\Rightarrow$ nous allons définir un langage algorithmique indépendant du langage de programmation qui servira ensuite à coder les algos

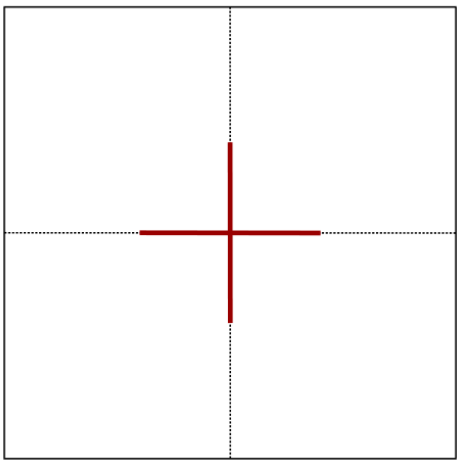
Citer quelques noms de langages de programmation ?

Méthode : comment procéder ?

1. Formuler clairement le problème
2. Exprimer les entrées et sorties du problème
3. Dégager les grandes actions / fonctionnalités
4. Concevoir l'algorithme
5. Traduire dans un langage de programmation

Exemple : tracer une croix avec un traceur

Tâche : tracer une croix où chaque branche a une longueur de 20 unités



- Méthode : écrire un premier algorithme très simple puis de nouvelles versions plus raffinées
- On procède par « **raffinages successifs** »

Exemple : tracer une croix avec un traceur

Algorithme niveau 1

1: Tracer la branche nord ;
2: Tracer la branche est ;
3: Tracer la branche sud ;
4: Tracer la branche ouest ;

Exemple : tracer une croix avec un traceur

Algorithme niveau 1

1: Tracer la branche nord ;
2: Tracer la branche est ;
3: Tracer la branche sud ;
4: Tracer la branche ouest ;

Algorithme niveau 2

1: Amener le stylet au centre sans tracer ;
2: Tracer 20 traits d'une unité vers le Nord ;
3: Amener le stylet au centre sans tracer ;
4: Tracer 20 traits d'une unité vers l'Est ;
5: Amener le stylet au centre sans tracer ;
6: Tracer 20 traits d'une unité vers le Sud ;
7: Amener le stylet au centre sans tracer ;
8: Tracer 20 traits d'une unité vers l'Ouest ;

Exemple : tracer une croix avec un traceur

Le traceur peut être commandé avec les actions suivantes :

LEVER : rompt le contact entre le stylet et la surface de travail.
BAISSER : établit le contact entre le stylet et la surface de travail.
CENTRER : amène le stylet au centre de la surface de travail.
MONTER : déplace le stylet d'un pas vers le Nord.
DESCENDRE : déplace le stylet d'un pas vers le Sud.
AVANCER : déplace le stylet d'un pas vers l'Est.
RECULER : déplace le stylet d'un pas vers l'Ouest.
RAZ : remet le compteur Cptr à zéro.
INCR : augmente de 1 la valeur du compteur Cptr.

Exemple : tracer une croix avec un traceur

Algorithme niveau 3

```
1: {Tracer la branche Nord}
2: LEVER ;
3: CENTRER ;
4: BAISSER ;
5: RAZ ;
6: tant que (Cptr < 20) faire
7:     MONTER ;
8:     INCR ;
9: fin tant que
10: {Tracer la branche Est}
11: {à vous !}
```

Exemple : tracer une croix avec un traceur

Algorithme niveau 3 (suite)

```
 1: {Tracer la branche Nord}
 2: ... ;
 3: {Tracer la branche Est}
 4: LEVER ;
 5: CENTRER ;
 6: BAISSER ;
 7: RAZ ;
 8: tant que (Cptr < 20) faire
 9:     AVANCER ;
10:     INCR ;
11: fin tant que
```

Même chose pour les deux branches manquantes !

Et maintenant ?

- Structure d'un algorithme
- Les variables
- Les structures de contrôle
- Les entrées / sorties
- Les opérations arithmétiques simples

Structure d'un algorithme

- Un algorithme est composé d'une suite d'actions **élémentaires** ou **composées**

Définition : action élémentaire ou instruction

- Action non-décomposable
- S'écrit à l'aide d'un verbe à l'infinitif suivi de compléments (≪ pseudo-français ≫)
- Types d'action :
  - opérations d'entrée/sortie
  - gestion de la mémoire
  - opérations arithmétiques
  - appel à d'autres fonctionnalités
- Exemples :
  - ≪ diviser a par b ≫ ou plus simplement ≪ a / b ≫
  - ≪ lire une valeur entrée au clavier ≫

Structure d'un algorithme

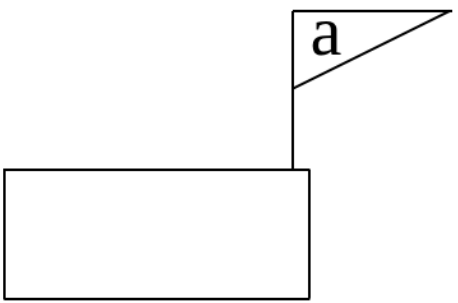
- Un algorithme est composé d'une suite d'actions **élémentaires** ou **composées**

Définition : action composée

Les actions composées sont des combinaisons de structures de contrôle et d'actions élémentaires

Les variables

- L'algorithme ne manipule pas directement les valeurs des données
- Il manipule des « objets abstraits » appelés variables qui prendront les valeurs des données seulement au moment de l'exécution
- Une variable est associée à une zone de la mémoire de la machine appelé « case mémoire » ou « mot mémoire »



Les variables

- La variable est repérée par un nom, ou identificateur
- Elle contient une valeur qui peut varier au cours de l'exécution de l'algorithme, d'où le nom de variable
- Au départ la valeur est indéfinie
- Toutes les valeurs qu'elle peut prendre sont d'un même type, c'est à dire sont de même nature et peuvent subir les mêmes opérations de base

Les variables

- Quand une variable est déclarée dans un algorithme (ici « a »), la machine lui associe une adresse mémoire
- Dans l'algorithme, on ne manipule que le nom des variables, pas l'adresse mémoire
- L'algorithme décrit des traitements sur la variable

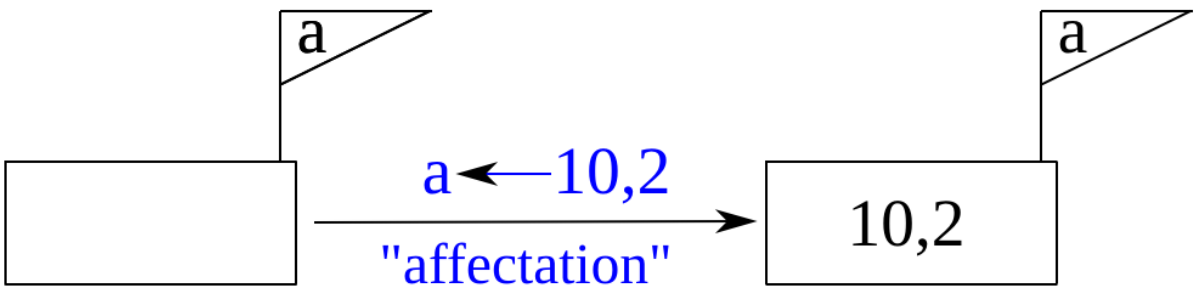
Noms des variables

- Le nom d'une variable (aussi appelé ≪identificateur≫) est un nom symbolique construit à l'aide :
  - des lettres majuscules,
  - des lettres minuscules non accentuées,
  - des chiffres,
  - du caractère ≪ souligné ≫ (*underscore) : _*.
  - *Exemples :* ≪ *a* ≫, ≪ *diametre* ≫, ≪ *nom_de_famille* ≫
  - ≪ *nom_de_famille* ≫*mieux que* ≪ *nomdefamille* ≫
  - *ATTENTION :* ≪ *nom de famille* ≫ *incorrect, pas d'espaces !*

Affectation à une variable

variable $\leftarrow$ valeur ;

$a \leftarrow 10,2$



"on affecte la valeur 10,2 à la variable a"

"a reçoit la valeur 10,2"

Les expressions

Une expression est composée d'opérandes (constantes, variables) et d'opérateurs
Selon la machine utilisée, on peut avoir :

- Les Opérateurs arithmétiques : +, -, *, /
- Fonctions mathématiques élémentaires
  - Mod : modulo
  - Abs : valeur absolu
  - Cos : cosinus
  - Sin : sinus
  - Tan : tangente
  - Sqrt : racine carrée

Affecter une expression à une variable

$$\text{variable} \leftarrow \text{expression}\,;$$
$$a \leftarrow 7 * 3 + 2;$$
$$a \leftarrow Sqrt(7) * 3 + 2;$$
$$b \leftarrow a/2, 1$$

- L'expression est tout d'abord évaluée
- Puis la valeur obtenue est ≪affectée≫ à la variable, c'est-à-dire placée dans la zone mémoire associée à la variable

Les structures de contrôle

Définition : structure de contrôle

- Ordonnancement d'actions élémentaires
- 3 structures possibles :
  1. la séquence
  2. la sélection
  3. la répétition

1. la séquence

Définition

Un enchaînement inconditionnel d'actions élémentaires

```
action 1 ;
action 2 ;
...
action n ;
```

2. la sélection ou test

Définition

Un choix entre deux actions ou séquences d'actions en fonction de la valeur d'une condition

```
SI condition ALORS
    action(s) 1 ;
SINON
    action(s) 2 ;
FIN SI ;
```

2. la sélection ou test

Définition

Un choix entre deux actions ou séquences d'actions en fonction de la valeur d'une condition

```
SI condition ALORS
    action(s) 1 ;
SINON
    action(s) 2 ;
FIN SI ;
```

- *condition* est une variable ou une expression booléenne (on dit aussi un booléen)
- *condition* peut prendre deux valeurs : VRAI ou FAUX
- La ou les *action(s) 1* ne sont effectuées que si *condition* est évaluée à VRAI

2. la sélection ou test (2)

Exemple 1

```
SI il pleut ALORS
    étudier le cours d'algo ;
SINON
    aller chasser des Pokémons ;
FIN SI ;
```

Exemple 2

```
SI a = 0 ALORS
    afficher "a est nul" ;
FIN SI ;
```

3. la répétition

Définition

Une même action ou séquence est exécutée tant qu'une condition reste vérifiée

```
TANT QUE condition FAIRE
    action 1 ;
    action 2 ;
    ... ;
FIN TANT QUE ;
```

3. la répétition (2)

Exemple

```
TANT QUE cpt > 0 FAIRE
    distance <- distance + 10 ;
    cpt <- cpt - 1 ;
FIN TANT QUE ;
```

Attention à ne pas oublier de modifier la valeur de *cpt* dans la boucle

3. la répétition (3)

La boucle POUR

- Alternative à TANT QUE
- Une même action ou séquence est exécutée sans condition un nombre de fois fixé

```
POUR i DE val1 À val2 FAIRE
    action 1 ;
    action 2 ;
    ... ;
FIN POUR ;
```

3. la répétition (4)

La boucle POUR

```
POUR cpt DE 0 À 9 FAIRE
    distance <- distance + 10 ;
FIN POUR ;
```

Donner deux différences fondamentales entre POUR et TANT QUE ?
Quand utiliser POUR plutôt que TANT QUE ?

3. la répétition (5)

POUR

- le nombre d'itérations est connu à l'avance
- le ≪ pas d'incrémentation ≫ est fixe
- la sortie de la boucle est liée à l'indice de la boucle (un compteur) et pas à une condition à définir vous-même

TANT QUE

- Dans tous les autres cas

Règles d'écriture

- Mots « réservés » :

  `SI, TANT QUE, FAIRE...`

- Penser à décaler (on dit « indenter ») les instructions dans les blocs

Entrées d'un algorithme

- entrées : une ou plusieurs variables utilisées par un algorithme

Exemple 1 :

Algorithme TRI-TABLEAU(A)

1: ...

Exemple 2 : demander à l'utilisateur de votre algorithme de taper au clavier une valeur numérique pour une variable *a* :

```
LIRE(a) ;
```

Sorties d'un algorithme

- sorties : données affichées à l'écran par l'algorithme

```
a <- 10,3
ECRIRE(a) ;
```

Que va afficher ce petit exemple ?

Premier algorithme

Afficher tous les nombres de 1 à N, avec N un nombre entier positif saisi par l'utilisateur

Premier algorithme

Afficher tous les nombres de 1 à N, avec N un nombre entier positif saisi par l'utilisateur

Algorithme AFFICHER_1_N niveau 1

1: Saisir la valeur de N ;
2: Afficher toutes les valeurs de 1 à N ;

Premier algorithme

Afficher tous les nombres de 1 à N, avec N un nombre entier positif saisi par l'utilisateur

Algorithme AFFICHER_1_N niveau 1

```
1: Saisir la valeur de N ;
2: Afficher toutes les valeurs de 1 à N ;
```

Algorithme AFFICHER_1_N niveau 2

```
1: Saisir la valeur de N ;
2: Initialiser la variable cpt à 1
3: TANT QUE cpt n'a pas atteint la valeur de N FAIRE
4:     Afficher cpt ;
5:     Augmenter la valeur de cpt de 1 (incrémenter cpt de 1) ;
6: FIN TANT QUE
```

Premier algorithme

Afficher tous les nombres de 1 à N, avec N un nombre entier positif saisi par l'utilisateur

Algorithme AFFICHER_1_N niveau 3

```
1: LIRE(N) ;
2: cpt ← 1
3: TANT QUE cpt < N FAIRE
4:     ECRIRE(cpt) ;
5:     cpt ← cpt + 1
6: FIN TANT QUE
```

Premier algorithme

Et avec POUR ce serait pas mieux pour cet exemple ?

Premier algorithme

Et avec POUR ce serait pas mieux pour cet exemple ?

Algorithme AFFICHER_1_N niveau 3

```
1: LIRE(N) ;
2: POUR cpt DE 1 À N FAIRE
3:     ECRIRE(cpt) ;
4: FIN POUR
```

Et en python ? (attention *spoiler* !)

Avec une boucle TANT QUE :

```
N = int(input('Entrer un entier positif : '))
cpt = 1
while cpt < N+1:
    print(cpt)
    cpt = cpt + 1
```

Avec une boucle POUR :

```
N = int(input('Entrer un entier positif : '))
for cpt in range(1,N+1):
    print(cpt)
```

Consignes pour le succès !

- Toujours assigner une variable utilisée dans une condition **avant** l'évaluation de cette condition
- Toujours être sûr que la condition dans un TANT QUE va s'arrêter un jour : on parle de « condition d'arrêt »

Types de variables

A chaque variable est associé un ≪ type ≫ qui peut être :

1. un type simple : *Entier, Réel, Caractère, Booléen*
2. un type structuré ou composé : un tableau, une liste, une pile, une file, une liste doublement chaînée, un arbre, etc.

Types de variables

A chaque variable est associé un ≪ type ≫ qui peut être :

1. un type simple : *Entier, Réel, Caractère, Booléen*
2. un type structuré ou composé : un tableau, une liste, une pile, une file, une liste doublement chaînée, un arbre, etc.

Le type définit deux choses :

1. Les valeurs possibles de la variable, par exemple un nombre réel ne peut pas être assigné à une variable de type *Entier*
2. Les opérations réalisables sur la variable en question : addition, division, etc.

Types de variables

En algorithmique et dans certains langages de programmation (Java, C, C++, etc.), il est impératif de déclarer explicitement le type d'une variable.

- En algo, la syntaxe est :

```
nomVariable : Type ;
nomVariable <- valeur ;
```

- Ou bien on peut faire la déclaration et l'affection en une seule ligne :

```
nomVariable : Type <- valeur ;
```

- Si la valeur d'une variable ne doit plus être modifiée après sa déclaration, on la déclare comme **constante** :

```
nomVariable : constante Type <- valeur ;
```

Types de variables

Exemples

Tableaux

- Contrairement à un objet simple comme un nombre, le tableau est une structure de données qui peut contenir une liste de valeurs
- Les éléments d'un tableau peuvent être des objets simples, ou des objets structurés
- Les ≪chaînes de caractères≫ sont des tableaux de caractères
- Les constantes chaînes de caractères sont des suites de caractères entre guillemets
  Exemple : ”ceci est une chaîne”

Tableaux

Déclaration en algo :

```
Type_Elements nomTableau[Nb_Elements] ;
```

Exemples

```
N : constante Entier <-  6 ;
Entier T[N] ;
```

Si l'on veut déclarer un tableau appelé *Tab* contenant des réels et demander la taille du tableau (nombre d'éléments) à l'utilisateur, appelée *Nb* pour changer de *N* :

```
Lire(Nb) ;
Réel Tab[Nb] ;
```

Dans les deux exemples, on obtient un tableau de *N* ou *Nb* cases mémoires auxquelles aucune valeur n'a encore été assignée / affectée

Tableaux

Syntaxe pour accéder à un élément du tableau :

```
nomTableau[Expression] ;
```

Exemples

```
N : constante Entier <-  6 ;
Entier T[N] ;
```

Attention, les indices commencent à 0 et pas 1 !
Accès au premier élément du tableau : T[0]
Accès au deuxième élément du tableau : T[1]
...
Accès au dernier élément du tableau : T[5] ou plus généralement T[N-1]

Tableaux : affectation

```
N : constante Entier <-  6 ;
Entier T[N] ;
```

- À la « main » :

```
T[0] <- 12 ;
T[1] <- 1 ;
T[2] <- -3 ;
T[3] <- 3 ;
T[4] <- 2 ;
T[5] <- 9 ;
```

- Avec une boucle POUR ou TANT QUE si la même valeur pour tous les éléments du tableau, par exemple 0 :

```
POUR indice de 0 À N-1 FAIRE
    T[indice] <- 0
```

Tableaux : affectation

Si l'on veut demander les valeurs de chaque élément à l'utilisateur :

```
indice : Entier <- 0
TANT QUE indice < N FAIRE
    Lire(T[indice])
    indice <- indice + 1
```

Tableaux

On peut définir des tableaux à dimension > 1 :

- 2 dimensions : une « matrice »
- à partir de 3 dimensions : un « tenseur »

Déclaration

```
dim1 <- constante Entier 6
dim2 <- constante Entier 10
...
dimN <- constante Entier 100
Entier T[dim1][dim2] ... [dimN]
```

Accès

```
T[Expr1][Expr2] ... [ExprN]
```

Fonctions

Question : comment réutiliser un algo existant plusieurs fois sans avoir à le recopier à chaque fois ?
Réponse : en définissant une fonction !

Définition : une fonction est une suite ordonnée d'instructions qui peut éventuellement retourner une valeur
En algo :

```
FONCTION nomFonction (liste de paramètres) : type
Variables nomVariables : type
DEBUT
    Instruction(s)
    RETOURNER Expression
FIN
```

Fonctions

Exemple : Définir une fonction qui renvoie le plus grand de deux nombres différents

```
FONCTION max (X: réel, Y:réel) : réel
{Retourne le plus grand des deux nombres X et Y}
DEBUT
    SI  X > Y ALORS
        RETOURNER X ;
    SINON
        RETOURNER Y ;
    FIN SI ;
FIN
```

Fonctions

Comment appeler la fonction dans notre algorithme principal ?

Syntaxe de l'appel : une variable appelée résultat dans cet exemple « reçoit » la valeur retournée par la fonction :

```
résultat <- nomFonction (liste de paramètres)
```

Exemple

```
A : Réel <- 10,2
B : Réel <- 11,3
plus_grand : Réel
plus_grand <- max(A, B)
ECRIRE("Le plus grand nombre vaut : ", plus_grand)
```

Les variables A et X (de la fonction max) sont-elles les mêmes ? Puis-je demander à afficher X dans l'algo principal ?

Fonctions : remarques

1. Une fonction peut ne rien renvoyer (certains parlent de **procédure** plutôt que de fonction dans ce cas)

```
FONCTION max (X: réel, Y:réel)
{Affiche le plus grand des deux nombres X et Y}
DEBUT
    SI  X > Y ALORS
        ECRIRE(X) ;
    SINON
        ECRIRE(Y) ;
    FIN SI ;
FIN
```

Dans ce cas, la syntaxe de l'appel est :

```
nomFonction(liste de paramètres)
```

Fonctions : remarques

2. Il faut souvent définir des « pré-conditions » sur les paramètres d'entrée

```
FONCTION factorielle (n: Entier) : Entier
{Affiche la factorielle de n}
DEBUT
    VERIFIER Entier : n
    VERIFIER n >= 0
    {calcul de factorielle}
    ...
FIN
```

Fonctions : remarques

3. Il faut donner des exemples d'appel de votre fonction dans les commentaires (dans l'entête)

```
FONCTION factorielle (n: Entier) : Entier
{Affiche la factorielle de n}
{factorielle(1)=1}
{factorielle(2)=2}
{factorielle(3)=6}
DEBUT
    ...
FIN
```

Fonctions : remarques

4. Il faut tester votre fonction sur de nombreux exemples, y compris et surtout des cas extrêmes : on parle de ≪ jeu de tests ≫

Exemple

Que donne votre fonction factorielle pour n=0, n=-3, n=3,2 ou encore n='A' ?