Algorithmique et Programmation C
DUT Informatique

Mohamed Tahrichi

mtahrichi@gmail.com

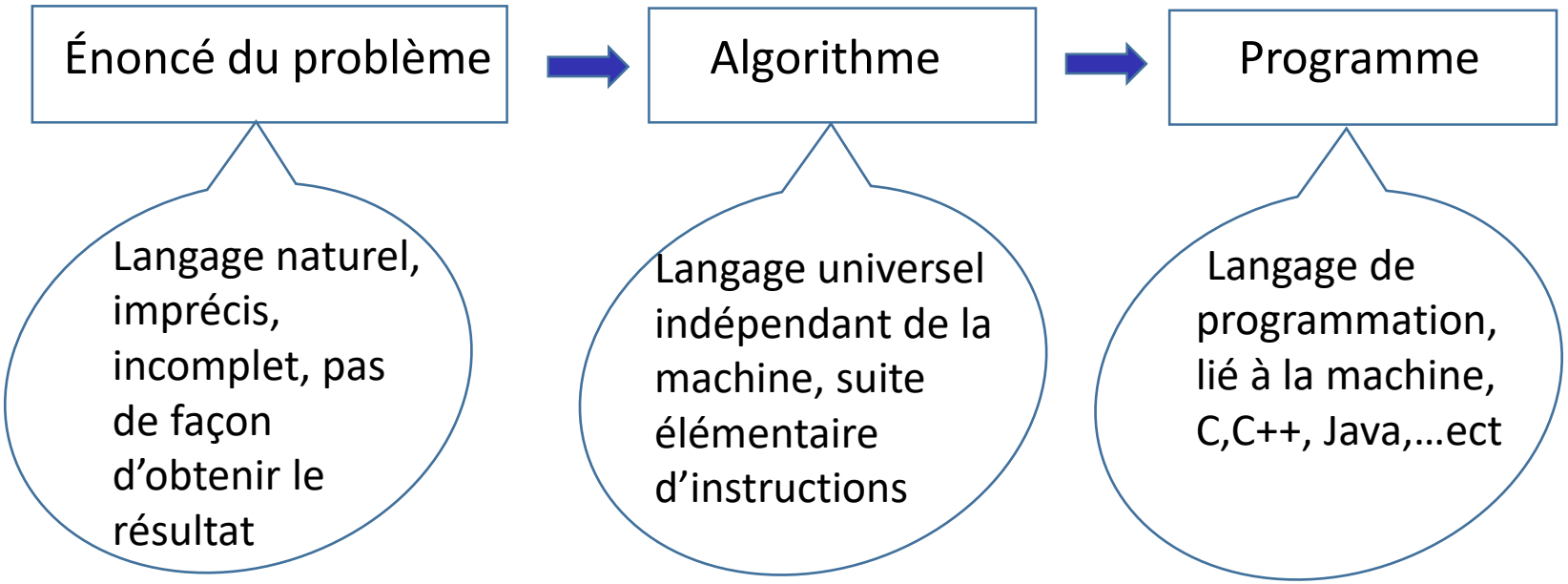
EST Oujda

Département d'informatique

2019-2020

Introduction

- On dit qu'on doit mettre en place un algorithme qui va être ensuite traduit en un programme exécutable par la machine.
- On a finalement l'enchaînement suivant :



Algorithme

Définition

- Un algorithme est une suite finie d'instructions qu'on applique dans un ordre bien déterminé à un nombre fini de données pour arriver à un résultat.
- Un algorithme ne dépend pas
  - du langage dans lequel il est implanté,
  - de la machine qui exécutera le programme correspondant.
- Un algorithme est écrit en utilisant un Langage de Description d'Algorithme (LDA).

Caractéristiques d'un algorithme

- L'algorithme est un moyen pour le programmeur de présenter son approche du problème à d'autres personnes. En effet, un algorithme est l'énoncé dans un langage bien défini d'une suite d'opérations permettant de répondre à un problème.
- Un algorithme doit donc être :
  - ✓Lisible : l'algorithme doit être compréhensible même par un non informaticien.
  - ✓De haut niveau : l'algorithme doit pouvoir être traduit en n'importe quel langage de programmation.

Caractéristiques d'un algorithme

- ✓Précis : chaque élément de l'algorithme ne doit pas porter une confusion.
- ✓Concis : un algorithme ne doit pas dépasser une page. Si ce n'est pas le cas, il faut décomposer le problème en plusieurs sous-problèmes.
- ✓Structuré : un algorithme doit être composé de différentes parties facilement identifiables.

Comment écrire un algorithme

- Pour écrire un algorithme on doit suivre les étapes suivantes :

1. Comprendre le problème
2. Identifier les données
3. Identifier les résultats souhaités
4. Déterminer les transformations à faire sur ces données pour obtenir les résultats.

Comment écrire un algorithme

- En informatique le schéma suivant est adopté pour la présentation de l'algorithme

Algorithme Nom de l'algorithme ;

Déclaration des constantes ;

Début

Déclaration des variables ;

Corps de l'algorithme
- Lecuture des données;
- Instruction à exécuter;
- Résultats à afficher;

Fin

Exemple

Exemple

Étapes de la réalisation d'un algorithme qui nous permet de calculer la surface d'un cercle

1. Problème : calculer la surface d'un cercle?
2. Données : π et R
3. Résultats souhaités : la surface du cercle
4. Traitement à faire sur ces données : $\pi \times R \times R$

Exemple

Exemple

Algortihme surface_cercle;
Constante π : réel ← 3,14;
Variable R : réel;
Variable S : réel;
Début
Lire(R);
S ← π*R*R;
Ecrire(S);
Fin

Éléments de base d'un algorithme

- Dans un algorithme les objets manipulés sont représentés par des symboles alors que les actions sont représentées par des symboles ou des verbes à l'infinitif.

- Dans l'exemple 1:
  - ✓ Les objets sont représentées par π , S et R
  - ✓ Les actions sont représentées par :

lire, écrire, * et ←

Les objets

- Chaque objet est défini par :

✓Un Identificateur qui le désigne et le distingue des autres objets.

✓Un type qui caractérise la nature des informations qui seront représentées par l'objet et les opérations qui seront autorisées sur cet objet.

✓Une valeur qui représente l'état actuel de l'objet.

Les objets

- Nous avons deux classe d'objets :

✓ Les constantes: une constante est un objet qui ne change pas le long de l'algorithme. Elle est déclarée par

Constante Identificateur: Type ← valeur ;

Exemple Constante π : réel ← 3,14 ;

✓Les variables: une variables est un objet qui représente une valeur numérique, une chaîne de caractères ou une fonction. Le contenu d'une variable peut être modifié par une action. Une variable est déclarée par :

Variable Identificateur : type ;

Exemple Variable R, S : réel ;

Les objets

- Les types : Le type permet de définir l'ensemble des valeurs susceptibles d'être prises par une variable ainsi que les opérations possibles sur cette variable. Les types de base sont :

  - ✓ Entier
  - ✓ Réel
  - ✓ Caractère (chaîne de caractères)
  - ✓ Booléen

Les opérateurs

- Les opérateurs permettent de produire un effet sur les objets de l'algorithme (variables ou constante) en effectuant des calculs, des vérification, des égalités et des inégalités. Ils sont divisés en trois classe :

a) Les opérateurs arithmétiques

Opérateur	Signification	Syntaxe
$+$	addition	$x + y$
-	soustraction	$x - y$
$*$	multiplication	$x * y$
$/$	division	x/y
mod	modulo	$x\ mod\ y$
div	Division Euclidienne	$x\ div\ y$

Exemple

Exemple

- ✓ (15 mod 4) = 3 :
  le reste de la division euclidienne de 15 par 4;
- ✓(10 / 2) = 5 :
  le quotient de la division euclidienne de 10 par 2;
- ✓ (11 / 2) = 5;
- ✓ 11 div 2=5.5;

Les opérateurs

b) Les opérateurs de comparaison

Opérateur	Signification	Syntaxe
=	égal	$x = y$
<>	différent	$x <> y$
<	inférieur	$x < y$
>	supérieur	$x > y$
<=	inférieur ou égal	$x <= y$
>=	supérieur ou égal	$x >= y$

c) Les opérateurs logiques

Opérateur	Signification	Syntaxe
et	intersection	x et y
ou	réunion	x ou y
non	complémentaire	non y

L'affectation

- Une instruction est une action élémentaire commandant à la machine un calcul, ou une communication avec un de ses périphériques d'entrés ou de sorties. Les instructions de base sont :
  - ✓ L'instruction d'affectation :

- L'affectation permet d'affecter une valeur à une variable.
- Symbolisée en algorithmique par «←», elle précise le sens de l'affectation.
- Syntaxe : Identificateur ← Expression

Expression peut être soit identificateur, constante, expression arithmétique ou expression logique

L'affectation

- **Sémantique :**

Une affectation peut être définie en deux étapes :

1. Évaluation de l'expression qui se trouve dans la partie droite de l'affectation
2. Copie la valeur de l'expression dans la variable.

Exemple

```
0  Algorithme    Calcul
1  Variables  A, B, C, D : entier
2  Debut
3        A ← 10
4        B ← 30
5        C ← A+B
6       D ← C*A
7  Fin
```

Instruction d'entrée & sortie

- L'instruction d'entrée ou de lecture permet à l'utilisateur de saisir des données au clavier pour qu'elles soient utilisées par le programme.

Syntaxe : Saisir(identificateur)

- L'instruction de sortie (d'écriture) permet d'afficher des informations à l'utilisateur à travers l'écran.

Syntaxe : Afficher(expression)

Expression peut être une valeur, un résultat, un message, le contenu d'une variable...

Exemple

Exemple

Algorithme PrixTTC

constantes TVA : réel ← 20.6;
variables prixHT, prixTTC : réels;

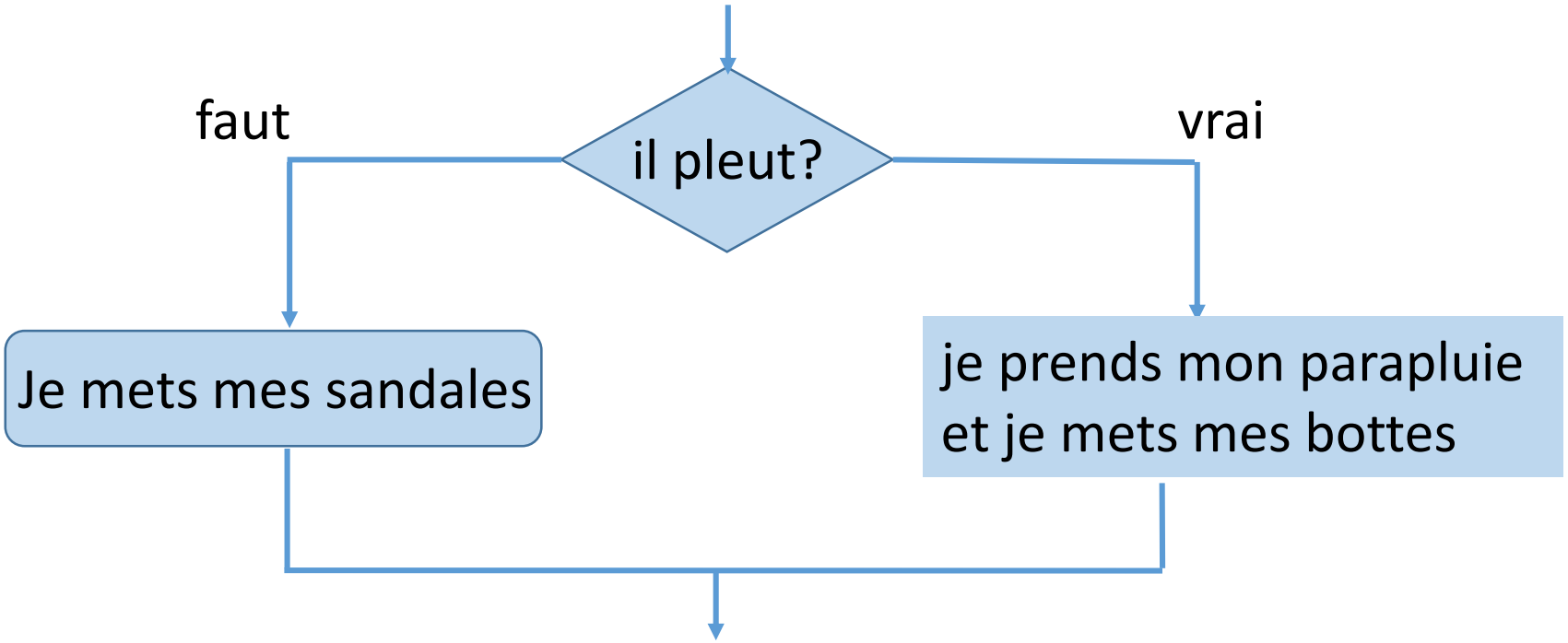
début

afficher("Donnez-moi le prix hors taxe :");
saisir(prixHT);
prixTTC ← prixHT * (1+TVA/100);
afficher(prixHT, "euros H.T. devient ", prixTTC, "euros T.T.C.");

Fin

Les conditionnelles

- Possibilité de choisir une séquence d'instructions selon une condition donnée.
- Exemple : « s'il pleut, je prends mon parapluie et je mets mes bottes, sinon je mets mes sandales. »



Si…alors…sinon

- Syntaxe :

```
SI condition
    ALORS
        liste d'instructions
    SINON
        liste d'instructions
FIN SI
```

- La *condition* est une expression booléenne {vrai,faux}.
- Si la condition est vraie, on exécute la branche *alors*.
- Si la condition est fausse, on exécute la branche *sinon*.

Exemple

Exemple

```
Algorithme   SimpleOuDouble

constante SEUIL : entier ← 10;
variable val : entier;
début
afficher("Donnez-moi un entier : ");
saisir(val);
si val < SEUIL
     alors afficher ("Voici son double :" , val × 2);
     sinon afficher ("Voici la valeur inchangée :" , val);
fsi

fin
```

Imbrication de conditionnelles

- Toute instruction algorithmique peut être placée dans une conditionnelle, donc également une conditionnelle !
- Cela permet de multiplier les choix possibles d'exécution.
- Syntaxe

si <*cond*1> **alors**

. . . . // cond1 vraie

sinon

si <*cond*2> **alors** // cond1 fausse

. . . // cond2 vraie

sinon

. . . // cond2 fausse

fsi

fsi

Selon

- Lorsque l'on veut comparer une seule variable à une énumération de valeurs connues à l'avance, on peut utiliser la structure selon.
- Syntaxe

selon <identificateur>
    (liste de) valeur(s) : instructions
    (liste de) valeur(s) : instructions
    …
    [autres: instructions]

- S'il y a plus de deux choix possibles, l'instruction selon permet une facilité d'écriture.

Exemple

Exemple

```
Algorithme  Selon;
Variable abr : chaine;
début
Lire(abr);
selon abr
  "M"       :  afficher( "Monsieur");
  "Mme " :  afficher("Madame");
   "Mlle"  :  afficher("Mademoiselle");
    autres : afficher("Monsieur, Madame");
Fselon
fin
```

Les boucles

- Possibilité de répéter une suite d'instructions selon une condition donnée.
- On dispose de trois structures de contrôle différentes :
  - pour : répète des instructions un nombre connu de fois ;
  - tant que : répète des instructions tant que la condition de la boucle est vraie ;
  - répéter : comme le tant que, mais on effectue au moins une fois les instructions de la boucle.

Structure pour

- Il est fréquent que le nombre de répétitions soit connu à l'avance, et que l'on ait besoin d'utiliser le numéro de l'itération afin d'effectuer des calculs ou des tests.
- Syntaxe:

```
POUR variable DE val initiale A val finale [par <pas>]
FAIRE
        liste d'instructions
FIN POUR
```

  - ✓La variable dont on donne le nom va prendre successivement toutes les valeurs entières entre valeur initiale et valeur finale.
  - ✓Pour chaque valeur prise par la variable, la liste des instructions est exécutée.
  - ✓La variable utilisée pour énumérer les itérations est appelée indice d'itération ou compteur.

La structure pour

- Implicitement, l'instruction pour:
  - initialise une variable de boucle (le compteur)
  - incrémente cette variable à chaque pas
  - vérifie que cette variable ne dépasse pas la borne supérieure

Exercice

Exercice

- En utilisant la boucle pour, écrire un algorithme qui calcule le factorielle d'un nombre n donné par l'utilisateur.

La structure tant que

- Dans beaucoup de cas, on souhaite répéter une instruction tant qu'une certaine condition est remplie, alors qu'il est a priori impossible de savoir à l'avance au bout de combien d'itérations cette condition cessera d'être satisfaite.
- Le mécanisme permettant cela est la boucle : Tant que.

Syntaxe :

```
TANT QUE condition FAIRE
        liste d'instructions
FIN TANT QUE
```

La structure tant que

Exemple

```
Algorithme FaitLeTotal
Constante STOP : entier ←-1;
variables val, totalValeurs : Entiers;
Début
totalValeurs←0;
afficher("Donnez une valeur,", STOP, " pour finir.");
saisir(val);
tant que val ≠STOP faire
    totalValeurs←totalValeurs+ val;
    afficher("Donnez une autre valeur,", STOP, " pour finir.");
    saisir(val);
Ftq
afficher("La somme des valeurs saisies est", totalValeurs);
fin
```

Comparaison pour et tant que

```
pour cpt ←1 à nbVal faire
    afficher("Donnez une valeur:");
    saisir(valeur);
    totalValeurs←totalValeurs+ valeur;
Fpour
```

...équivaut à :

```
cpt ←0
tant que cpt <nbVal faire
    afficher("Donnez une valeur :");
    saisir(valeur);
    totalValeurs←totalValeurs+ valeur;
    cpt ←cpt + 1;
ftq
```

Comparaison pour et tant que

- Syntaxe:

  Répéter

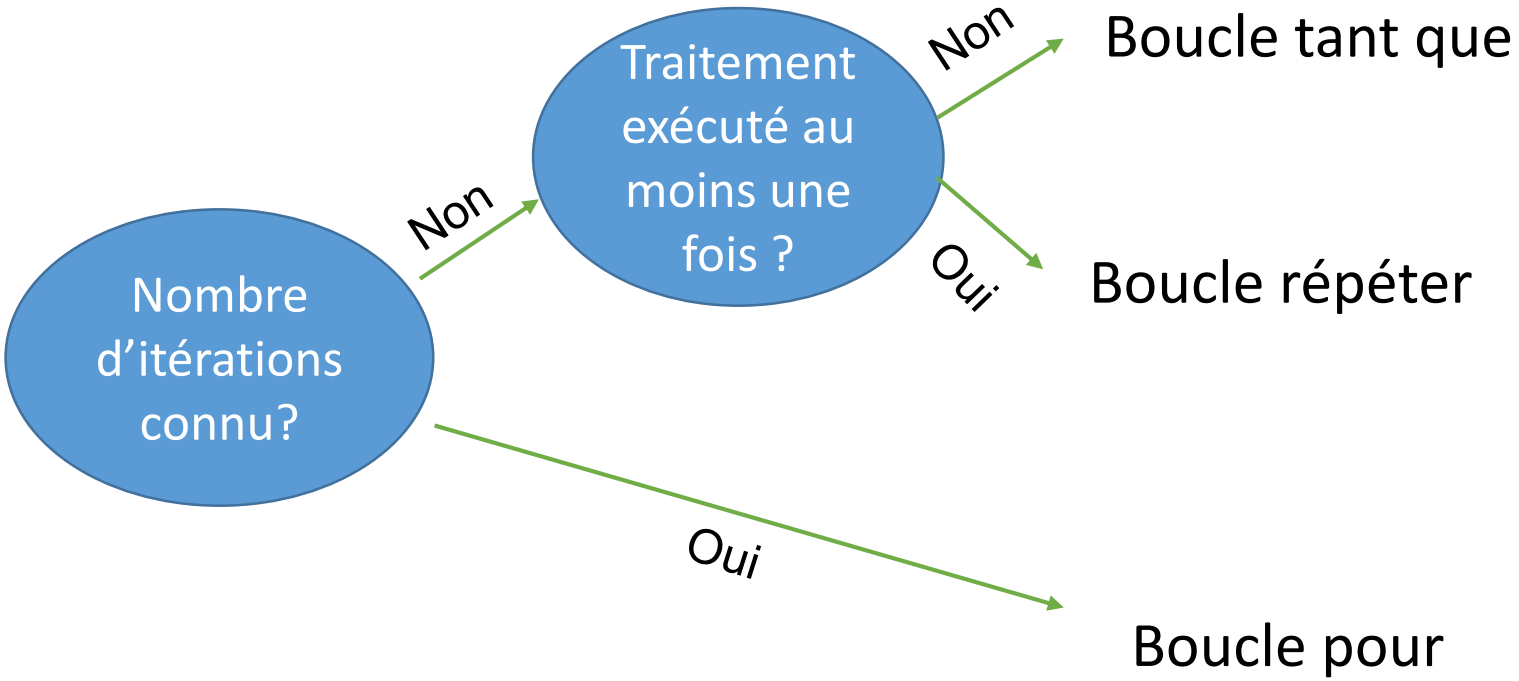
  (ré)affectation de la variable de condition

  Traitement {suite d'instructions}

  tant que <expression logique (vraie)>

- exécuter une suite d'instructions au moins une fois et la répéter tant qu'une condition est remplie

Choisir pour…tant que… répéter



Introduction

- Une fonction est une suite ordonnée d'instructions qui retourne une valeur (bloc d'instructions nommé et paramétré).
- Prennent en entrée des paramètres.
- Restituent à l'algorithme appelant un ou plusieurs résultats.
- Définies par :

En-tête

- ✓ Nom : identifiant de la fonction
- ✓ Liste des paramètres : informations extérieures à la fonction
- ✓ Résultat : valeur de retour de la fonction
- ✓ Description: en langage naturel du rôle de la fonction

Corps

- ✓ Algorithme de la fonction

En-tête d'une fonction

- Syntaxe

fonction nomFonction(mode *Par1*:type =,Par2:…) : **ret** typeRet

- nomFonction identifiant de la fonction
- mode à donner pour chaque paramètre
  - ✓**in :** paramètre *donnée*, non modifiable par la fonction ;
  - ✓**out :** paramètre *résultat*, modifié par la fonction, valeur initiale non utilisée ;
  - ✓**in-out :** paramètre *donnée/résultat*, modifié, valeur initiale utilisée.
- Par1 identifiant du paramètre dans la fonction
- type type du paramètre
- retour
  - ✓mot clef «ret» suivi du type de la valeur renvoyée par la fonction (typeRet) ;
  - ✓si pas de retour, on remplace ret par le mot clef «vide».

Corps d'une fonction

- Construit comme un algorithme classique :
  - ✓Déclaration des constantes
  - ✓Déclaration des variables
  - ✓Algorithme de nomFonction
- Les constantes/variables définies dans une fonction sont utilisables uniquement dans cette fonction et ne peuvent pas être utilisées en dehors de celle-ci.
- Elles sont détruites dès que l'on sort de la fonction.
- On parle de constantes/variables locales.

Déclaration/Définition/Retour

- Déclaration et définition
  - ✓La *déclaration* des fonctions est placée avant le début de l'algorithme avec la déclaration des variables
  - ✓La *définition* des fonctions est placée après l'algorithme principal
- Retour de fonction
  - ✓Lorsque la fonction a un *retour*, sa dernière instruction exécutée doit être

    retourner valeurDeRetour
  - ✓ Lorsque la fonction n'a pas de retour, elle peut arrêter son exécution avec l'instruction

    retourner
- Les fonctions sans retour sont appelées *procédures*.

Variables, paramètres et appel

- Variables locales et paramètres
  - ✓Les paramètres de la fonction *ne doivent pas* être redéclarer dans son corps.
  - ✓Ce sont déjà des variables *locales* à la fonction dans lesquelles on recopie les éléments reçus de l'algorithme appelant.
  - ✓Les fonctions peuvent ne pas avoir de paramètres.

- Paramètres formels et effectifs
  - ✓Une fonction est utilisée avec son nom, suivi de la liste des valeurs pour ses paramètres entre parenthèses.
  - ✓Les paramètres dans l'en-tête de la fonction sont les *paramètres formels*.
  - ✓Les paramètres utilisés lors de l'appel de la fonction par l'algorithme sont les *paramètres effectifs*.

En-tête d'une fonction

Algorithme exemple_fonction

fonction somme(**in** *a* : entier, **in** *b* : entier) : **ret** entier

Variables *x, y, z : entier*

Début

z ← somme(*x,y*);

écrire(*z);*

Fin

Variables effectifs

---

fonction somme(**in** *a* : entier, **in** *b* : entier) : **ret** entier

Variables *c :* entier

Début

c ← *a+b*

retourner *c*

Fin

Variables formels

Exercices

Exercice_1

- Ecrire un algorithme qui fait appel à une fonction pour dessiner trois carrés de coté 1,3 et 5.
- Généraliser l'algorithme précédent à fin de dessiner n carrés de 1,3,5,…,2n-1 avec n donné par l'utilisateur.

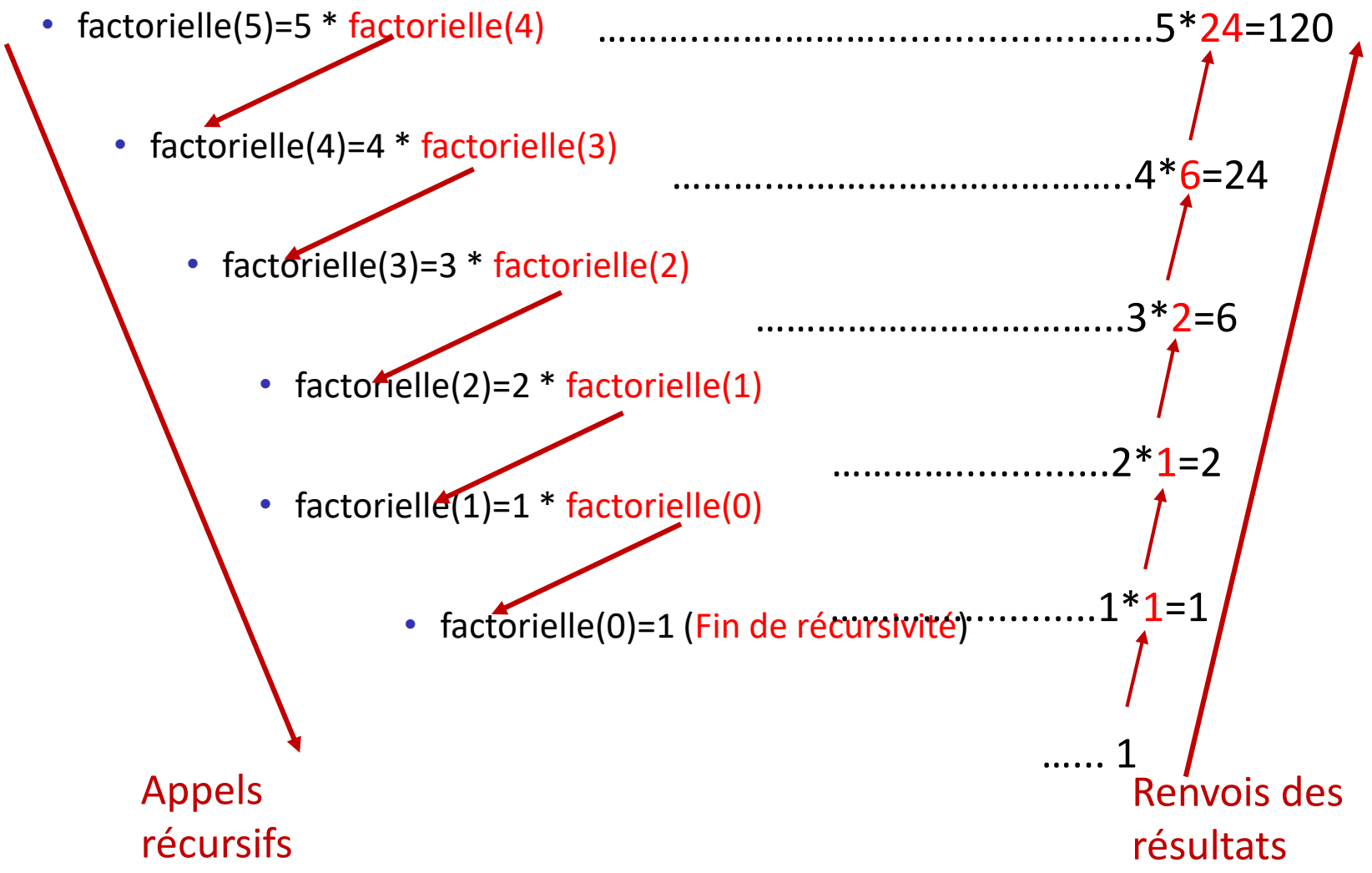
Exercice_2

- Ecrire une fonction qui convertit une durée exprimée en seconde en heures, minutes, secondes.
- Ecrire un algorithme qui fait appel à la fonction précédente

Fonctions récursives

- Une fonction récursive est une fonction qui peut faire appelle à elle-même.
- Pour définir une fonction récursive il faut :
  - ✓un (ou plusieurs) cas de base, dans laquelle la fonction ne fait plus appelle à elle-même. Sinon l'algorithme ne peut pas terminer.
  - ✓si on ne se situe pas dans un cas de base la fonction fait appel à elle-même (appel récursif).
  - ✓Chaque appel récursif doit en principe se «rapprocher» d'un cas de base, de façon à permettre la terminaison de l'algorithme.
- C'est comme pour définir une suite récurrente en mathématique

Exemple


factorielle(5)=5 * factorielle(4)
factorielle(4)=4 * factorielle(3)
factorielle(3)=3 * factorielle(2)
factorielle(2)=2 * factorielle(1)
factorielle(1)=1 * factorielle(0)
factorielle(0)=1 (Fin de récursivité)
5*24=120
4*6=24
3*2=6
2*1=2
1*1=1
1
Appels récursifs
Renvois des résultats

Exercice

Exercice

- Ecrire une fonction récursive qui calcul le nombre binomial:

$$\text{Binomial}(k,n) = \begin{cases} 0 & \text{si } k>n \\ 1 & \text{si } k=n \text{ ou } k=0 \\ \text{Binomial}(k-1,n-1)+\text{Binomial}(k,n-1) & \text{sinon} \end{cases}$$

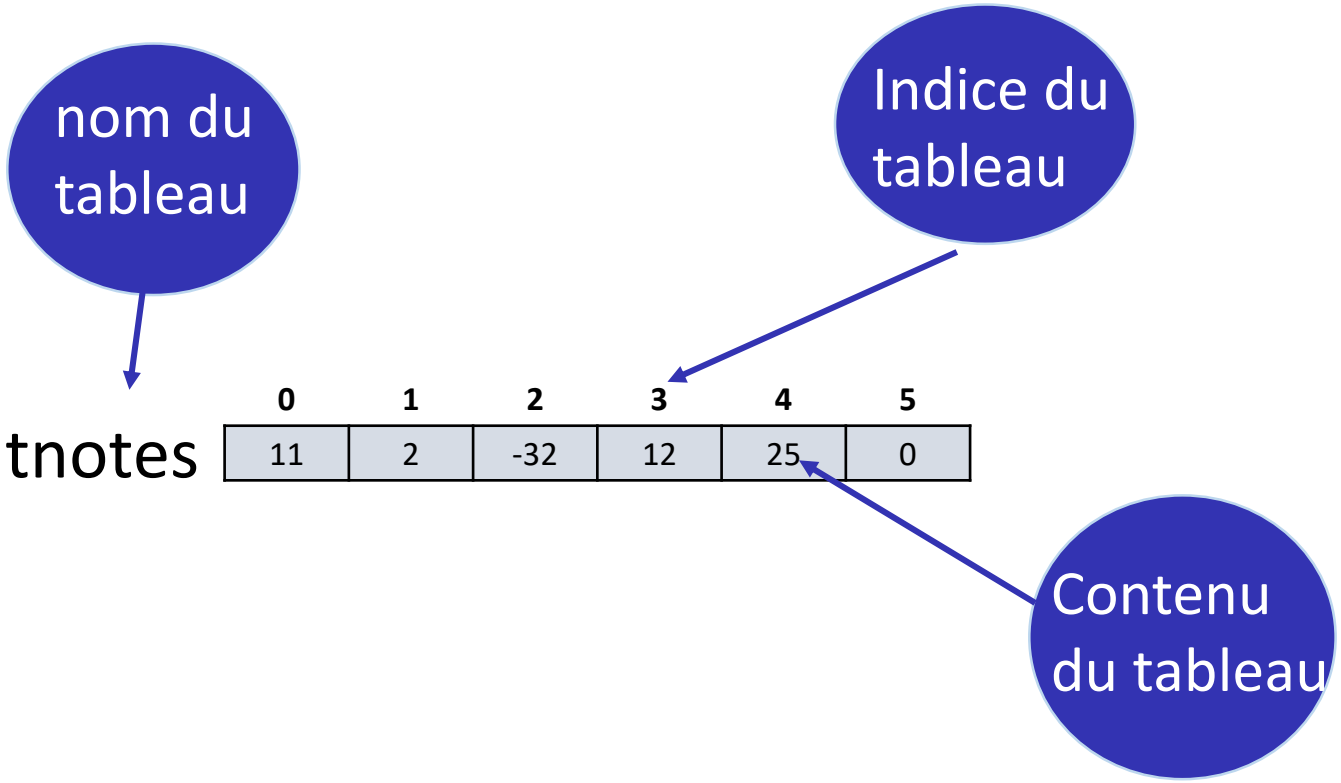
Introduction

- Structure de données qui permet de rassembler un ensemble de valeurs de même type sous un même nom en les différenciant par un indice.
- Exemple :

  tnotes | 11 | 2 | -32 | 12 | 2 | 25 | 0 | un tableau de 7 valeurs
- Déclaration

  variables tnotes [7] : tableau de réels;
- Chaque élément est repéré dans le tableau par un indice qui varie de 0 à *taille* -1
- On accède à la case 2 par *tNotes*[2]. Attention, c'est la 3ème case !

Introduction



- Nombre d'octets occupés : dépend du type des valeurs enregistrées

Initialisation

- Lorsqu'on connaît les valeurs initiales à placer dans un tableau on peut le faire en une seule instruction :

tableau ← {liste de valeurs}

- Le nombre d'éléments dans la liste doit correspondre au nombre d'éléments du tableau.

Exemple

```
Algorithme Exemple
Variables T[7]: tableau d'entiers
Début
T←{2,3,6,4,7,12,3}  // Initialisation du tableau
Fin
```

Lecture et écriture

- Pour initialiser un tableau avec des valeurs fournies par l'utilisateur, on est obligé de lire les valeurs une par une.
- Une boucle de lecture est donc nécessaire.
- L'affichage se fait aussi élément par élément.

Exemple

pour *i* **de** 0 **à** *nbElem*-1 **faire** // boucle de lecture...
Lire(*Tab*[*i*])
fpour
pour *i* **de** 0 **à** *nbElem*-1 **faire** // boucle d'écriture...
écrire *Tab*[*i*]
fpour

Tableaux et fonctions

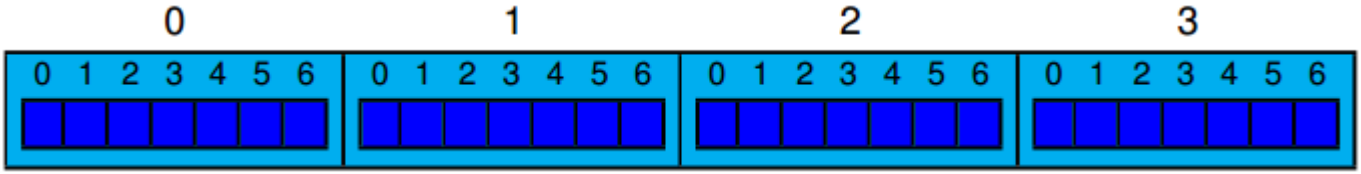
- On peut passer un tableau en paramètre d'une fonction.
- Il faut en général passer aussi la *taille* du tableau.

Exemple

fonction lirePrix(**out** *tPrix* : tableau de réels, **in** *taille* : entier) : **vide**
Variables *i :* entier
Début
pour *i* **de** 0 **à** *taille*-1 **faire**
lire(*tPrix*[*i*]) ;
fpour
Fin

Tableaux à plusieurs dimensions

- Représentation de matrices, de tables, d'images, . . .
- Un tableau contient des éléments de même type :
  - le type peut être quelconque, et donc aussi un tableau
- Un tableau à deux dimensions est donc un tableau de tableaux !
- Exemple : *table* (tableau [4] de tableaux [7] d'entiers) définit *table* comme un tableau de 4 cases contenant chacune un tableau de 7 entiers.
- Représentation vectorielle de *table* :



Tableaux à plusieurs dimensions

- Déclaration:

  variable T: tableau(4,7) d'entiers

- Accès à un élément : *nomDuTableau*[*indice*1,*indice*2], avec :
  - *indice*1 entre 0 et $dimension_1$-1
  - *indice*2 entre 0 et $dimension_2$-1
- Représentation matricielle de *table* :