Algorithmique et Programmation

Hervé HOCQUARD
hocquard@labri.fr

Université de Bordeaux

LaBRI

2019–2020

Semestre 1

Semestre 1

Plan

Introduction

- L'informatique met en jeu des ordinateurs qui fonctionnent selon des schémas préétablis.

Introduction

- L'informatique met en jeu des ordinateurs qui fonctionnent selon des schémas préétablis.
- Pour résoudre un problème donné à l'aide d'un ordinateur, il faut lui indiquer la *suite d'actions* à exécuter dans son schéma de fonctionnement.

Introduction

- L'informatique met en jeu des ordinateurs qui fonctionnent selon des schémas préétablis.
- Pour résoudre un problème donné à l'aide d'un ordinateur, il faut lui indiquer la *suite d'actions* à exécuter dans son schéma de fonctionnement.
- Cette suite d'actions est un *programme* qui est exprimé dans un langage de programmation plus ou moins évolué (code machine, Visual Basic, C, C++, Java, Caml, Prolog, ...).

Introduction

- L'informatique met en jeu des ordinateurs qui fonctionnent selon des schémas préétablis.
- Pour résoudre un problème donné à l'aide d'un ordinateur, il faut lui indiquer la *suite d'actions* à exécuter dans son schéma de fonctionnement.
- Cette suite d'actions est un *programme* qui est exprimé dans un langage de programmation plus ou moins évolué (code machine, Visual Basic, C, C++, Java, Caml, Prolog, ...).
- Pour écrire un programme (la suite d'actions), il faut donc d'abord savoir *comment faire* pour résoudre le problème.

Algorithme

- Un *algorithme* est l'expression de la résolution d'un problème de sorte que le résultat soit calculable par une machine.

Algorithme

- Un *algorithme* est l'expression de la résolution d'un problème de sorte que le résultat soit calculable par une machine.
- L'algorithme est exprimé dans un modèle théorique de machine universelle (von Neumann) qui ne dépend pas de la machine réelle sur laquelle on va l'utiliser.

Algorithme

- Un *algorithme* est l'expression de la résolution d'un problème de sorte que le résultat soit calculable par une machine.
- L'algorithme est exprimé dans un modèle théorique de machine universelle (von Neumann) qui ne dépend pas de la machine réelle sur laquelle on va l'utiliser.
- Il peut être écrit en langage naturel, mais pour être lisible par tous, on utilise un *langage algorithmique* plus restreint qui comporte tous les concepts de base de fonctionnement d'une machine.

Algorithme

- Un *algorithme* est l'expression de la résolution d'un problème de sorte que le résultat soit calculable par une machine.
- L'algorithme est exprimé dans un modèle théorique de machine universelle (von Neumann) qui ne dépend pas de la machine réelle sur laquelle on va l'utiliser.
- Il peut être écrit en langage naturel, mais pour être lisible par tous, on utilise un *langage algorithmique* plus restreint qui comporte tous les concepts de base de fonctionnement d'une machine.
- On a finalement l'enchaînement suivant :

$$\text{Énoncé du problème} \longrightarrow \underset{\text{(universel)}}{\text{Algorithme}} \longrightarrow \underset{\text{(lié à une machine)}}{\text{Programme}}$$

Problème et énoncé

- Un problème a un *énoncé* qui donne des informations sur le *résultat* attendu.

Problème et énoncé

- Un problème a un *énoncé* qui donne des informations sur le *résultat* attendu.
- L'énoncé peut être en langage naturel, imprécis, incomplet, et ne donne généralement pas la façon d'obtenir le résultat.

Problème et énoncé

- Un problème a un *énoncé* qui donne des informations sur le *résultat* attendu.
- L'énoncé peut être en langage naturel, imprécis, incomplet, et ne donne généralement pas la façon d'obtenir le résultat.
- Exemples :
  - calculer le PGCD de deux nombres ;
  - calculer la surface d'une pièce ;
  - ranger une liste de mots par ordre alphabétique ;
  - calculer x tel que x divise a et b et s'il existe y qui divise a et b alors $x \geq y$.

Problème et énoncé

- Un problème a un *énoncé* qui donne des informations sur le *résultat* attendu.
- L'énoncé peut être en langage naturel, imprécis, incomplet, et ne donne généralement pas la façon d'obtenir le résultat.
- Exemples :
  - calculer le PGCD de deux nombres ;
  - calculer la surface d'une pièce ;
  - ranger une liste de mots par ordre alphabétique ;
  - calculer x tel que x divise a et b et s'il existe y qui divise a et b alors $x \geq y$.
- Le dernier exemple décrit très précisément le résultat mais ne nous dit pas *comment le calculer* .

Résolution de problèmes

- Pour résoudre un problème, il faut donner *la suite d'actions élémentaires* à réaliser pour obtenir le résultat.

Résolution de problèmes

- Pour résoudre un problème, il faut donner *la suite d'actions élémentaires* à réaliser pour obtenir le résultat.
- Les actions élémentaires dont on dispose sont définies par le langage algorithmique utilisé.

Résolution de problèmes

- Pour résoudre un problème, il faut donner *la suite d'actions élémentaires* à réaliser pour obtenir le résultat.
- Les actions élémentaires dont on dispose sont définies par le langage algorithmique utilisé.
- Exemple : « **Construire une maison** »
  Niveler le terrain
  Placer les fondations
  Monter les murs
  Poser la toiture
  Installer l'électricité
  Installer les canalisations
  Ajouter les portes et fenêtres

Résolution de problèmes

- Pour résoudre un problème, il faut donner *la suite d'actions élémentaires* à réaliser pour obtenir le résultat.
- Les actions élémentaires dont on dispose sont définies par le langage algorithmique utilisé.
- Exemple : « **Construire une maison** »
  Niveler le terrain
  Placer les fondations
  Monter les murs
  Poser la toiture
  Installer l'électricité
  Installer les canalisations
  Ajouter les portes et fenêtres
- L'ordre des opérations a son importance, mais dans certains cas plusieurs ordres sont possibles.

Résolution de problèmes

- Pour résoudre un problème, il faut donner *la suite d'actions élémentaires* à réaliser pour obtenir le résultat.
- Les actions élémentaires dont on dispose sont définies par le langage algorithmique utilisé.
- Exemple : « **Construire une maison** »
  Niveler le terrain
  Placer les fondations
  Monter les murs
  Poser la toiture
  Installer l'électricité
  Installer les canalisations
  Ajouter les portes et fenêtres
- L'ordre des opérations a son importance, mais dans certains cas plusieurs ordres sont possibles.
- Parfois, il faut *décomposer* les actions trop complexes.

Résolution de problèmes

- Pour résoudre un problème, il faut donner *la suite d'actions élémentaires* à réaliser pour obtenir le résultat.
- Les actions élémentaires dont on dispose sont définies par le langage algorithmique utilisé.
- Exemple : « **Construire une maison** »
  Niveler le terrain
  Placer les fondations
  Monter les murs
  Poser la toiture
  Installer l'électricité
  Installer les canalisations
  Ajouter les portes et fenêtres

  « **Poser la toiture** »
  Poser la charpente
  Poser les chevrons
  Poser les liteaux
  Poser la sous-toiture
  Poser la couverture
- L'ordre des opérations a son importance, mais dans certains cas plusieurs ordres sont possibles.
- Parfois, il faut *décomposer* les actions trop complexes.

Un exemple

Exemple : algorithme d'Euclide pour calculer le PGCD de deux entiers.

Algorithme d'Euclide

Données : deux nombres entiers a et b

Résultat : PGCD de a et b

1. Ordonner les deux nombres tels que $a \geq b$.
2. Calculer leur différence $c \leftarrow a - b$.
3. Recommencer en remplaçant a par c ($a \leftarrow c$) jusqu'à ce que a devienne égal à b.
4. Le résultat est alors a ($= b$)

Exécution avec 174 et 72

Étape 1 : $a \leftarrow 174$ et $b \leftarrow 72$

Étape 2 : $c \leftarrow 174 - 72 = 102$

Étape 3 : $a \leftarrow c = 102 \neq b = 72$

Exécution avec 174 et 72

Étape 1 : $a \leftarrow 174$ et $b \leftarrow 72$

Étape 2 : $c \leftarrow 174 - 72 = 102$

Étape 3 : $a \leftarrow c = 102 \neq b = 72$

Étape 4 : $a \leftarrow 102$ et $b \leftarrow 72$

Étape 5 : $c \leftarrow 102 - 72 = 30$

Étape 6 : $a \leftarrow c = 30 \neq b = 72$

Exécution avec 174 et 72

Étape 1 : $a \leftarrow 174$ et $b \leftarrow 72$

Étape 2 : $c \leftarrow 174 - 72 = 102$

Étape 3 : $a \leftarrow c = 102 \neq b = 72$

Étape 4 : $a \leftarrow 102$ et $b \leftarrow 72$

Étape 5 : $c \leftarrow 102 - 72 = 30$

Étape 6 : $a \leftarrow c = 30 \neq b = 72$

Étape 7 : $a \leftarrow 72$ et $b \leftarrow 30$

Étape 8 : $c \leftarrow 72 - 30 = 42$

Étape 9 : $a \leftarrow c = 42 \neq b = 30$

Exécution avec 174 et 72

Étape 1 : $a \leftarrow 174$ et $b \leftarrow 72$

Étape 2 : $c \leftarrow 174 - 72 = 102$

Étape 3 : $a \leftarrow c = 102 \neq b = 72$

Étape 4 : $a \leftarrow 102$ et $b \leftarrow 72$

Étape 5 : $c \leftarrow 102 - 72 = 30$

Étape 6 : $a \leftarrow c = 30 \neq b = 72$

Étape 7 : $a \leftarrow 72$ et $b \leftarrow 30$

Étape 8 : $c \leftarrow 72 - 30 = 42$

Étape 9 : $a \leftarrow c = 42 \neq b = 30$

Étape 10 : $a \leftarrow 42$ et $b \leftarrow 30$

Étape 11 : $c \leftarrow 42 - 30 = 12$

Étape 12 : $a \leftarrow c = 12 \neq b = 30$

...

Exécution avec 174 et 72 (suite)

...

Étape 13 : $a \leftarrow 30$ et $b \leftarrow 12$

Étape 14 : $c \leftarrow 30 - 12 = 18$

Étape 15 : $a \leftarrow c = 18 \neq b = 12$

Exécution avec 174 et 72 (suite)

...

Étape 13 : $a \leftarrow 30$ et $b \leftarrow 12$

Étape 14 : $c \leftarrow 30 - 12 = 18$

Étape 15 : $a \leftarrow c = 18 \neq b = 12$

Étape 16 : $a \leftarrow 18$ et $b \leftarrow 12$

Étape 17 : $c \leftarrow 18 - 12 = 6$

Étape 18 : $a \leftarrow c = 6 \neq b = 12$

Exécution avec 174 et 72 (suite)

...

Étape 13 : $a \leftarrow 30$ et $b \leftarrow 12$

Étape 14 : $c \leftarrow 30 - 12 = 18$

Étape 15 : $a \leftarrow c = 18 \neq b = 12$

Étape 16 : $a \leftarrow 18$ et $b \leftarrow 12$

Étape 17 : $c \leftarrow 18 - 12 = 6$

Étape 18 : $a \leftarrow c = 6 \neq b = 12$

Étape 19 : $a \leftarrow 12$ et $b \leftarrow 6$

Étape 20 : $c \leftarrow 12 - 6 = 6$

Étape 21 : $a \leftarrow c = 6 = b = 6$

Exécution avec 174 et 72 (suite)

...

Étape 13 : $a \leftarrow 30$ et $b \leftarrow 12$

Étape 14 : $c \leftarrow 30 - 12 = 18$

Étape 15 : $a \leftarrow c = 18 \neq b = 12$

Étape 16 : $a \leftarrow 18$ et $b \leftarrow 12$

Étape 17 : $c \leftarrow 18 - 12 = 6$

Étape 18 : $a \leftarrow c = 6 \neq b = 12$

Étape 19 : $a \leftarrow 12$ et $b \leftarrow 6$

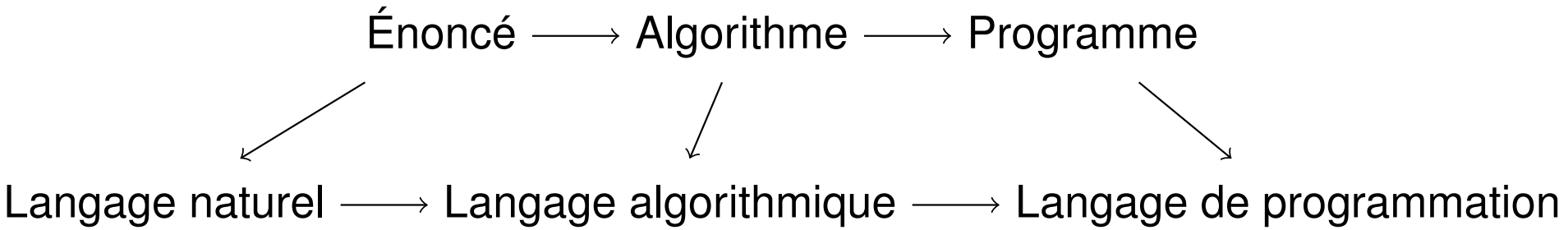
Étape 20 : $c \leftarrow 12 - 6 = 6$

Étape 21 : $a \leftarrow c = 6 = b = 6$

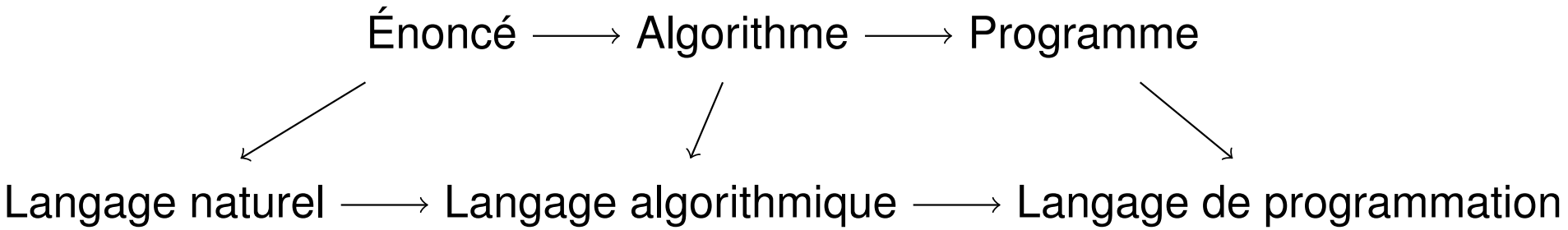
Arrêt car $a = b$

le résultat est donc 6

Langages

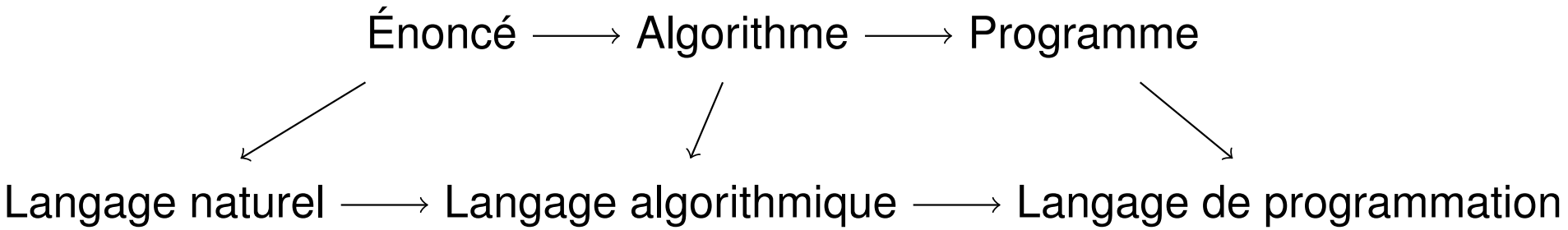

Énoncé ⟶ Algorithme ⟶ Programme
Langage naturel ⟶ Langage algorithmique ⟶ Langage de programmation

Langages



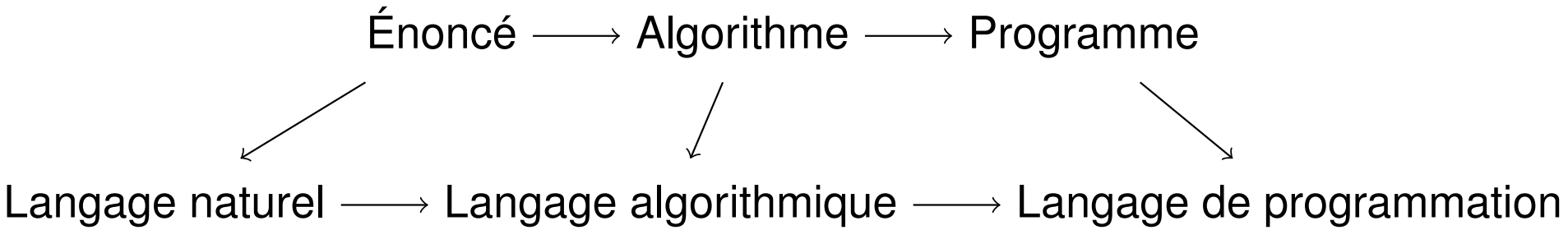
- Un langage est composé de deux éléments :

Langages



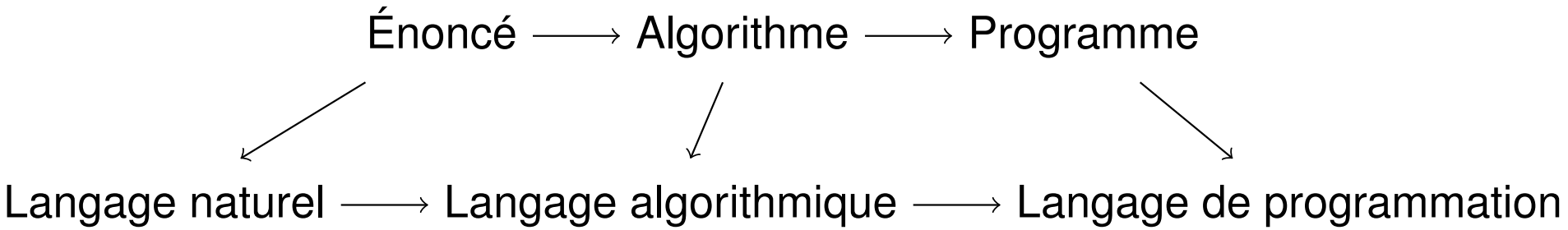
- Un langage est composé de deux éléments :
  - la *grammaire* qui fixe la syntaxe ;

Langages



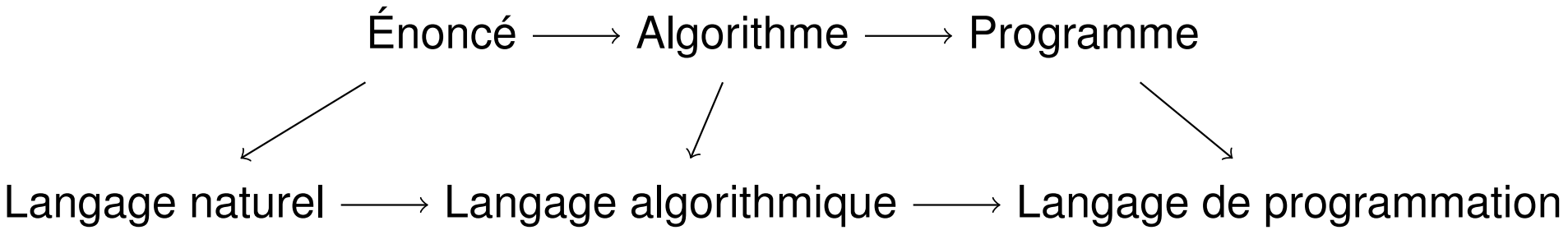
- Un langage est composé de deux éléments :
  - la *grammaire* qui fixe la syntaxe ;
  - la *sémantique* qui donne le sens.

Langages



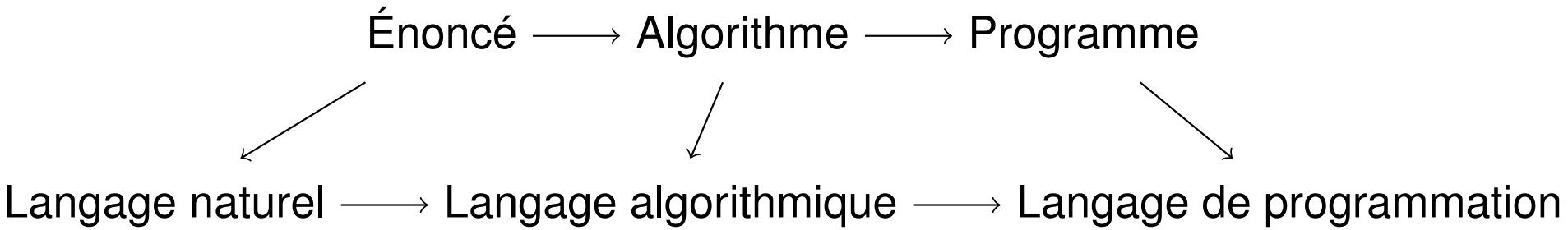
- Un langage est composé de deux éléments :
  - la *grammaire* qui fixe la syntaxe ;
  - la *sémantique* qui donne le sens.
- Le langage répond donc à deux questions :

Langages



- Un langage est composé de deux éléments :
  - la *grammaire* qui fixe la syntaxe ;
  - la *sémantique* qui donne le sens.
- Le langage répond donc à deux questions :
  - comment écrire les choses ?

Langages



- Un langage est composé de deux éléments :
  - la *grammaire* qui fixe la syntaxe ;
  - la *sémantique* qui donne le sens.
- Le langage répond donc à deux questions :
  - comment écrire les choses ?
  - que signifient les choses écrites ?

Types de langages

- Langages machines : code binaire, liés au processeur

Types de langages

- Langages machines : code binaire, liés au processeur
- Langages assembleurs : bas niveau, liés au processeur

Types de langages

- Langages machines : code binaire, liés au processeur
- Langages assembleurs : bas niveau, liés au processeur
- Langages évolués : haut niveau, indépendants de la machine
  $\Rightarrow$ compilateur ou interpréteur pour l'exécution

Types de langages

- Langages machines : code binaire, liés au processeur
- Langages assembleurs : bas niveau, liés au processeur
- Langages évolués : haut niveau, indépendants de la machine

  $\Rightarrow$ compilateur ou interpréteur pour l'exécution
  - *Langages impératifs (procéduraux)* : suite des instructions à réaliser dans l'ordre d'exécution : Visual Basic, C, Pascal, Fortran, Cobol, ...

Types de langages

- Langages machines : code binaire, liés au processeur
- Langages assembleurs : bas niveau, liés au processeur
- Langages évolués : haut niveau, indépendants de la machine

  $\Rightarrow$ compilateur ou interpréteur pour l'exécution
  - *Langages impératifs (procéduraux)* : suite des instructions à réaliser dans l'ordre d'exécution : Visual Basic, C, Pascal, Fortran, Cobol, ...
  - *Langages à objets* : modélisation par entités ayant des propriétés et des interactions possibles : C++, Java, ...

Types de langages

- Langages machines : code binaire, liés au processeur
- Langages assembleurs : bas niveau, liés au processeur
- Langages évolués : haut niveau, indépendants de la machine

  $\Rightarrow$ compilateur ou interpréteur pour l'exécution
  - *Langages impératifs (procéduraux)* : suite des instructions à réaliser dans l'ordre d'exécution : Visual Basic, C, Pascal, Fortran, Cobol, ...
  - *Langages à objets* : modélisation par entités ayant des propriétés et des interactions possibles : C++, Java, ...
  - *Langages déclaratifs* : règles de calcul et vérification de propriétés sans spécification d'ordre :
    - Fonctionnels : les règles sont exprimées par des fonctions : Lisp, Caml, ...
    - Logiques : les règles sont exprimées par des prédicats logiques : Prolog, ...

Plan

Étapes de la conception

- La conception d'un algorithme est la phase la plus difficile.

Étapes de la conception

- La conception d'un algorithme est la phase la plus difficile.
- On privilégie une méthodologie *hiérarchique* en *décomposant* l'algorithme en parties.

Étapes de la conception

- La conception d'un algorithme est la phase la plus difficile.
- On privilégie une méthodologie *hiérarchique* en *décomposant* l'algorithme en parties.
- Chaque partie est elle-même décomposée jusqu'à obtenir des instructions élémentaires (*raffinements successifs)*.

Étapes de la conception

- La conception d'un algorithme est la phase la plus difficile.
- On privilégie une méthodologie *hiérarchique* en *décomposant* l'algorithme en parties.
- Chaque partie est elle-même décomposée jusqu'à obtenir des instructions élémentaires (*raffinements successifs)*.
- Les différents éléments d'un algorithme sont :

Étapes de la conception

- La conception d'un algorithme est la phase la plus difficile.
- On privilégie une méthodologie *hiérarchique* en *décomposant* l'algorithme en parties.
- Chaque partie est elle-même décomposée jusqu'à obtenir des instructions élémentaires (*raffinements successifs)*.
- Les différents éléments d'un algorithme sont :

  Description des données ce qui doit être donné à l'algorithme

Étapes de la conception

- La conception d'un algorithme est la phase la plus difficile.
- On privilégie une méthodologie *hiérarchique* en *décomposant* l'algorithme en parties.
- Chaque partie est elle-même décomposée jusqu'à obtenir des instructions élémentaires (*raffinements successifs)*.
- Les différents éléments d'un algorithme sont :

  Description des données ce qui doit être donné à l'algorithme

  Description des résultats ce que doit produire l'algorithme

Étapes de la conception

- La conception d'un algorithme est la phase la plus difficile.
- On privilégie une méthodologie *hiérarchique* en *décomposant* l'algorithme en parties.
- Chaque partie est elle-même décomposée jusqu'à obtenir des instructions élémentaires (*raffinements successifs)*.
- Les différents éléments d'un algorithme sont :

  Description des données — ce qui doit être donné à l'algorithme

  Description des résultats — ce que doit produire l'algorithme

  Idée de l'algorithme — les grandes étapes des traitements et calculs

Étapes de la conception

- La conception d'un algorithme est la phase la plus difficile.
- On privilégie une méthodologie *hiérarchique* en *décomposant* l'algorithme en parties.
- Chaque partie est elle-même décomposée jusqu'à obtenir des instructions élémentaires (*raffinements successifs)*.
- Les différents éléments d'un algorithme sont :

Description des données	ce qui doit être donné à l'algorithme
Description des résultats	ce que doit produire l'algorithme
Idée de l'algorithme	les grandes étapes des traitements et calculs
Lexique des variables	liste des valeurs manipulées

Étapes de la conception

- La conception d'un algorithme est la phase la plus difficile.
- On privilégie une méthodologie *hiérarchique* en *décomposant* l'algorithme en parties.
- Chaque partie est elle-même décomposée jusqu'à obtenir des instructions élémentaires (*raffinements successifs)*.
- Les différents éléments d'un algorithme sont :

Description des données	ce qui doit être donné à l'algorithme
Description des résultats	ce que doit produire l'algorithme
Idée de l'algorithme	les grandes étapes des traitements et calculs
Lexique des variables	liste des valeurs manipulées
Algorithme	le détail des traitements et calculs

Étapes de la conception

- La conception d'un algorithme est la phase la plus difficile.
- On privilégie une méthodologie *hiérarchique* en *décomposant* l'algorithme en parties.
- Chaque partie est elle-même décomposée jusqu'à obtenir des instructions élémentaires (*raffinements successifs)*.
- Les différents éléments d'un algorithme sont :

Description des données	ce qui doit être donné à l'algorithme
Description des résultats	ce que doit produire l'algorithme
Idée de l'algorithme	les grandes étapes des traitements et calculs
Lexique des variables	liste des valeurs manipulées
Algorithme	le détail des traitements et calculs

- D'autres éléments peuvent s'ajouter (on les verra plus loin. . .)

Données et résultats

Description des données

- Spécifier précisément ce qui doit être donné à l'algorithme avant de l'utiliser.

Données et résultats

Description des données

- Spécifier précisément ce qui doit être donné à l'algorithme avant de l'utiliser.
- Synopsis :
  - Utilisation du mot clef « **Données** » suivi d'un texte en langage naturel décrivant les données.

Données et résultats

Description des données

- Spécifier précisément ce qui doit être donné à l'algorithme avant de l'utiliser.
- Synopsis :
  - Utilisation du mot clef « **Données** » suivi d'un texte en langage naturel décrivant les données.

Description des résultats

- Spécifier précisément ce que doit produire l'algorithme et le lien entre le résultat et les données fournies au départ.

Données et résultats

Description des données

- Spécifier précisément ce qui doit être donné à l'algorithme avant de l'utiliser.
- Synopsis :
  - Utilisation du mot clef « **Données** » suivi d'un texte en langage naturel décrivant les données.

Description des résultats

- Spécifier précisément ce que doit produire l'algorithme et le lien entre le résultat et les données fournies au départ.
- Synopsis :
  - Utilisation du mot clef « **Résultats** » suivi d'un texte en langage naturel décrivant le résultat.

Données et résultats

Description des données

- Spécifier précisément ce qui doit être donné à l'algorithme avant de l'utiliser.
- Synopsis :
  - Utilisation du mot clef « **Données** » suivi d'un texte en langage naturel décrivant les données.

Description des résultats

- Spécifier précisément ce que doit produire l'algorithme et le lien entre le résultat et les données fournies au départ.
- Synopsis :
  - Utilisation du mot clef « **Résultats** » suivi d'un texte en langage naturel décrivant le résultat.

Données et résultats

Description des données

- Spécifier précisément ce qui doit être donné à l'algorithme avant de l'utiliser.
- Synopsis :
  - Utilisation du mot clef « **Données** » suivi d'un texte en langage naturel décrivant les données.

Description des résultats

- Spécifier précisément ce que doit produire l'algorithme et le lien entre le résultat et les données fournies au départ.
- Synopsis :
  - Utilisation du mot clef « **Résultats** » suivi d'un texte en langage naturel décrivant le résultat.

Exemples

Problème 1 : trouver x tel que x divise a et b et s'il existe y qui divise a et b alors $x \geq y$

Données

Deux entiers strictement positifs : a et b.

Résultat

PGCD de a et b.

Exemples

Problème 1 : trouver x tel que x divise a et b et s'il existe y qui divise a et b alors $x \geq y$

Données

Deux entiers strictement positifs : a et b.

Résultat

PGCD de a et b.

Problème 2 : calculer la superficie d'un champ

Données

(aucune donnée)

Résultat

Demande les dimensions d'un champ à l'utilisateur, puis affiche sa superficie.

Idée de l'algorithme

- Description informelle en langage naturel des grandes étapes de l'algorithme (1ère décomposition).

Idée de l'algorithme

- Description informelle en langage naturel des grandes étapes de l'algorithme (1ère décomposition).
- Synopsis :
  - Mot clef « **Idée** » suivi de l'énumération des étapes.

Idée de l'algorithme

- Description informelle en langage naturel des grandes étapes de l'algorithme (1ère décomposition).
- Synopsis :
  - Mot clef « **Idée** » suivi de l'énumération des étapes.

Exemple : calcul de la superficie d'un champ

Idée

1. acquérir la longueur et la largeur du champ ;
2. calculer la superficie ;
3. afficher la superficie.

Idée de l'algorithme

- Description informelle en langage naturel des grandes étapes de l'algorithme (1ère décomposition).
- Synopsis :
  - Mot clef « **Idée** » suivi de l'énumération des étapes.

Exemple : calcul de la superficie d'un champ

Idée

1. acquérir la longueur et la largeur du champ ;
2. calculer la superficie ;
3. afficher la superficie.

- Ces trois étapes sont *déduites* du résultat demandé.

Idée de l'algorithme

- Description informelle en langage naturel des grandes étapes de l'algorithme (1ère décomposition).
- Synopsis :
  - Mot clef « **Idée** » suivi de l'énumération des étapes.

Exemple : calcul de la superficie d'un champ

Idée

1. acquérir la longueur et la largeur du champ ;
2. calculer la superficie ;
3. afficher la superficie.

- Ces trois étapes sont *déduites* du résultat demandé.
- Chaque étape de l'idée peut elle-même être décomposée si elle est trop complexe, et ainsi de suite (cf. exemple de la maison).

Plan

Lexique des variables

- Liste des valeurs manipulées par l'algorithme.

Lexique des variables

- Liste des valeurs manipulées par l'algorithme.
- Synopsis :
  - Mot clef « **Lexique des variables** » suivi de la liste détaillée des variables.

Lexique des variables

- Liste des valeurs manipulées par l'algorithme.
- Synopsis :
  - Mot clef « **Lexique des variables** » suivi de la liste détaillée des variables.
- Les variables permettent de nommer et mémoriser les valeurs manipulées par l'algorithme.

Lexique des variables

- Liste des valeurs manipulées par l'algorithme.
- Synopsis :
  - Mot clef « **Lexique des variables** » suivi de la liste détaillée des variables.
- Les variables permettent de nommer et mémoriser les valeurs manipulées par l'algorithme.
- Elles correspondent à des *emplacements en mémoire* réservés pour stocker ces valeurs.

Lexique des variables

- Liste des valeurs manipulées par l'algorithme.
- Synopsis :
  - Mot clef « **Lexique des variables** » suivi de la liste détaillée des variables.
- Les variables permettent de nommer et mémoriser les valeurs manipulées par l'algorithme.
- Elles correspondent à des *emplacements en mémoire* réservés pour stocker ces valeurs.
- Elles sont définies par :

Lexique des variables

- Liste des valeurs manipulées par l'algorithme.
- Synopsis :
  - Mot clef « **Lexique des variables** » suivi de la liste détaillée des variables.
- Les variables permettent de nommer et mémoriser les valeurs manipulées par l'algorithme.
- Elles correspondent à des *emplacements en mémoire* réservés pour stocker ces valeurs.
- Elles sont définies par :
  - *Un nom* : référence de la variable
    exemples : i, numéroProduit, ...

Lexique des variables

- Liste des valeurs manipulées par l'algorithme.
- Synopsis :
  - Mot clef « **Lexique des variables** » suivi de la liste détaillée des variables.
- Les variables permettent de nommer et mémoriser les valeurs manipulées par l'algorithme.
- Elles correspondent à des *emplacements en mémoire* réservés pour stocker ces valeurs.
- Elles sont définies par :
  - *Un nom* : référence de la variable
    exemples : i, numéroProduit, ...
  - *Un type* : nature de la valeur
    exemples : entier, réel, caractère, ...

Lexique des variables

- Les informations données pour chaque variable sont :

Lexique des variables

- Les informations données pour chaque variable sont :

  Nom doit être significatif

Lexique des variables

- Les informations données pour chaque variable sont :
  - Nom doit être significatif
  - Type indiqué entre parenthèses

Lexique des variables

- Les informations données pour chaque variable sont :

Nom	doit être significatif
Type	indiqué entre parenthèses
Description	texte bref en langage naturel donnant la signification de la variable dans l'algorithme

Lexique des variables

- Les informations données pour chaque variable sont :

Nom	doit être significatif
Type	indiqué entre parenthèses
Description	texte bref en langage naturel donnant la signification de la variable dans l'algorithme
Rôle	on distingue trois rôles possibles :

Lexique des variables

- Les informations données pour chaque variable sont :

  Nom doit être significatif

  Type indiqué entre parenthèses

  Description texte bref en langage naturel donnant la signification de la variable dans l'algorithme

  Rôle on distingue trois rôles possibles :

  - DONNÉE : la valeur de la variable est *donnée* à l'algorithme. Elle est renseignée *avant* l'exécution de l'algorithme.

Lexique des variables

- Les informations données pour chaque variable sont :

  Nom doit être significatif

  Type indiqué entre parenthèses

  Description texte bref en langage naturel donnant la signification de la variable dans l'algorithme

  Rôle on distingue trois rôles possibles :

  - DONNÉE : la valeur de la variable est *donnée* à l'algorithme. Elle est renseignée *avant* l'exécution de l'algorithme.
  - RÉSULTAT : la valeur de la variable est fournie en *résultat* par l'algorithme, généralement calculée par celui-ci. Elle peut être utilisée *après* l'exécution de l'algorithme.

Lexique des variables

- Les informations données pour chaque variable sont :

  Nom doit être significatif

  Type indiqué entre parenthèses

  Description texte bref en langage naturel donnant la signification de la variable dans l'algorithme

  Rôle on distingue trois rôles possibles :

  - DONNÉE : la valeur de la variable est *donnée* à l'algorithme. Elle est renseignée *avant* l'exécution de l'algorithme.
  - RÉSULTAT : la valeur de la variable est fournie en *résultat* par l'algorithme, généralement calculée par celui-ci. Elle peut être utilisée *après* l'exécution de l'algorithme.
  - INTERMÉDIAIRE : la valeur de la variable est calculée par l'algorithme et utilisée dans des calculs sans être fournie en résultat (résultats intermédiaires).

Exemples

Exemple 1 : PGCD

Lexique des variables

a	(entier)	Premier entier donné	DONNÉE
b	(entier)	Deuxième entier donné	DONNÉE
pgcd	(entier)	PGCD de *a* et *b*	RÉSULTAT
c	(entier)	Variable intermédiaire	INTERMÉDIAIRE

Exemples

Exemple 1 : PGCD

Lexique des variables

a	(entier)	Premier entier donné	DONNÉE
b	(entier)	Deuxième entier donné	DONNÉE
pgcd	(entier)	PGCD de *a* et *b*	RÉSULTAT
c	(entier)	Variable intermédiaire	INTERMÉDIAIRE

- Les deux entiers *a* et *b* sont des données du problème.

Exemples

Exemple 1 : PGCD

Lexique des variables

a	(entier)	Premier entier donné	DONNÉE
b	(entier)	Deuxième entier donné	DONNÉE
pgcd	(entier)	PGCD de *a* et *b*	RÉSULTAT
c	(entier)	Variable intermédiaire	INTERMÉDIAIRE

- Les deux entiers *a* et *b* sont des données du problème.
- La variable *pgcd* contient le résultat rendu par l'algorithme.

Exemples

Exemple 1 : PGCD

Lexique des variables

a	(entier)	Premier entier donné	DONNÉE
b	(entier)	Deuxième entier donné	DONNÉE
pgcd	(entier)	PGCD de *a* et *b*	RÉSULTAT
c	(entier)	Variable intermédiaire	INTERMÉDIAIRE

- Les deux entiers *a* et *b* sont des données du problème.
- La variable *pgcd* contient le résultat rendu par l'algorithme.
- La variable *c* est une variable intermédiaire dans cet algorithme.

Exemples

Exemple 2 : superficie d'un champ

Lexique des variables

longueur	(réel)	Longueur du champ	INTERMÉDIAIRE
largeur	(réel)	Largeur du champ	INTERMÉDIAIRE
surface	(réel)	Superficie du champ	INTERMÉDIAIRE

Exemples

Exemple 2 : superficie d'un champ

Lexique des variables

longueur	(réel)	Longueur du champ	INTERMÉDIAIRE
largeur	(réel)	Largeur du champ	INTERMÉDIAIRE
surface	(réel)	Superficie du champ	INTERMÉDIAIRE

- Il n'y a aucune donnée pour l'algorithme (les paramètres requis sont demandés à l'utilisateur).

Exemples

Exemple 2 : superficie d'un champ

Lexique des variables

longueur	(réel)	Longueur du champ	INTERMÉDIAIRE
largeur	(réel)	Largeur du champ	INTERMÉDIAIRE
surface	(réel)	Superficie du champ	INTERMÉDIAIRE

- Il n'y a aucune donnée pour l'algorithme (les paramètres requis sont demandés à l'utilisateur).
- L'algorithme ne rend aucun résultat (hormis un affichage).

Exemples

Exemple 2 : superficie d'un champ

Lexique des variables

longueur	(réel)	Longueur du champ	INTERMÉDIAIRE
largeur	(réel)	Largeur du champ	INTERMÉDIAIRE
surface	(réel)	Superficie du champ	INTERMÉDIAIRE

- Il n'y a aucune donnée pour l'algorithme (les paramètres requis sont demandés à l'utilisateur).
- L'algorithme ne rend aucun résultat (hormis un affichage).
- Toutes les variables ont le rôle « intermédiaire » dans cet algorithme.

Plan

Algorithme

- Détail, en langage algorithmique des traitements et calculs effectués par l'algorithme.

Algorithme

- Détail, en langage algorithmique des traitements et calculs effectués par l'algorithme.
- Synopsis :
  - Mot clef « **Algorithme** » suivi de la suite des actions élémentaires.

Algorithme

- Détail, en langage algorithmique des traitements et calculs effectués par l'algorithme.
- Synopsis :
  - Mot clef « **Algorithme** » suivi de la suite des actions élémentaires.

Exemple : superficie d'un champ

Algorithme

longueur $\leftarrow$ lire
largeur $\leftarrow$ lire
surface $\leftarrow$ *longueur* $\times$ *largeur*
écrire *surface*

Explications

- « lire » signifie que la valeur est donnée par l'utilisateur (ex. : au clavier).

Explications

- « lire » signifie que la valeur est donnée par l'utilisateur (ex. : au clavier).
- « écrire » ou « Affiche » affiche le texte et/ou les valeurs qui suivent à l'écran.

Explications

- « lire » signifie que la valeur est donnée par l'utilisateur (ex. : au clavier).
- « écrire » ou « Affiche » affiche le texte et/ou les valeurs qui suivent à l'écran.
- L'*affectation* est indiquée par « $\leftarrow$ » :
  - affecte la valeur résultant de l'évaluation de l'expression à droite de la flèche dans la variable placée à gauche ;
  - « *truc* $\leftarrow$ *machin* » peut se lire « la variable *truc* reçoit la valeur *machin* » ;
  - attention à la correspondance des types !
    - on ne peut affecter à une variable qu'une valeur de *même type* ;
  - cela permet de vérifier la cohérence de ce que l'on écrit.

Explications

- « lire » signifie que la valeur est donnée par l'utilisateur (ex. : au clavier).
- « écrire » ou « Affiche » affiche le texte et/ou les valeurs qui suivent à l'écran.
- L'*affectation* est indiquée par « $\leftarrow$ » :
  - affecte la valeur résultant de l'évaluation de l'expression à droite de la flèche dans la variable placée à gauche ;
  - « *truc* $\leftarrow$ *machin* » peut se lire « la variable *truc* reçoit la valeur *machin* » ;
  - attention à la correspondance des types !
    - on ne peut affecter à une variable qu'une valeur de *même type* ;
  - cela permet de vérifier la cohérence de ce que l'on écrit.
- Exemple : *maVariable* $\leftarrow$ 15
  - *maVariable* contient la valeur 15 après l'instruction ;
  - la valeur 15 sera prise en lieu et place de *maVariable* dans la suite de l'algorithme, jusqu'à sa prochaine modification.

Explications

- « lire » signifie que la valeur est donnée par l'utilisateur (ex. : au clavier).
- « écrire » ou « Affiche » affiche le texte et/ou les valeurs qui suivent à l'écran.
- L'*affectation* est indiquée par « $\leftarrow$ » :
  - affecte la valeur résultant de l'évaluation de l'expression à droite de la flèche dans la variable placée à gauche ;
  - « *truc* $\leftarrow$ *machin* » peut se lire « la variable *truc* reçoit la valeur *machin* » ;
  - attention à la correspondance des types !
    - on ne peut affecter à une variable qu'une valeur de *même type* ;
  - cela permet de vérifier la cohérence de ce que l'on écrit.
- Exemple : *maVariable* $\leftarrow$ 15
  - *maVariable* contient la valeur 15 après l'instruction ;
  - la valeur 15 sera prise en lieu et place de *maVariable* dans la suite de l'algorithme, jusqu'à sa prochaine modification.
- Il est interdit de chercher à utiliser la valeur d'une variable avant que celle-ci soit définie (variable non initialisée).

Déroulement de l'exemple

Algorithme

longueur ← lire
largeur ← lire
surface ← *longueur* × *largeur*
écrire *surface*

Variables

longueur ? *largeur* ? *surface* ?

Entrées

40
12

Sorties

Déroulement de l'exemple

Algorithme

longueur ← lire
largeur ← lire
surface ← *longueur* × *largeur*
écrire *surface*

Variables

longueur	*largeur*	*surface*
40	?	?

Entrées

40
12

Sorties

Déroulement de l'exemple

Algorithme

longueur ← lire
largeur ← lire
surface ← *longueur* × *largeur*
écrire *surface*

Variables

longueur	40	*largeur*	12	*surface*	?

Entrées

40
12

Sorties

Déroulement de l'exemple

Algorithme

longueur ← lire
largeur ← lire
surface ← *longueur* × *largeur*
écrire *surface*

Variables

longueur	40	*largeur*	12	*surface*	480

Entrées

```
40
12
```

Sorties

Déroulement de l'exemple

Algorithme

longueur ← lire
largeur ← lire
surface ← *longueur* × *largeur*
écrire *surface*

Variables

longueur	40	*largeur*	12	*surface*	480

Entrées

```
40
12
```

Sorties

```
480
```

Déroulement de l'exemple

Algorithme

longueur ← lire
largeur ← lire
surface ← *longueur* × *largeur*
écrire *surface*

Variables

longueur	40	*largeur*	12	*surface*	480

Entrées

```
40
12
```

Sorties

```
480
```

Exemple complet : superficie d'un champ

Données

(aucune donnée)

Résultat

Demande les dimensions d'un champ à l'utilisateur, puis affiche sa superficie.

Idée

1. acquérir la longueur et la largeur du champ ;
2. calculer la superficie ;
3. afficher la superficie.

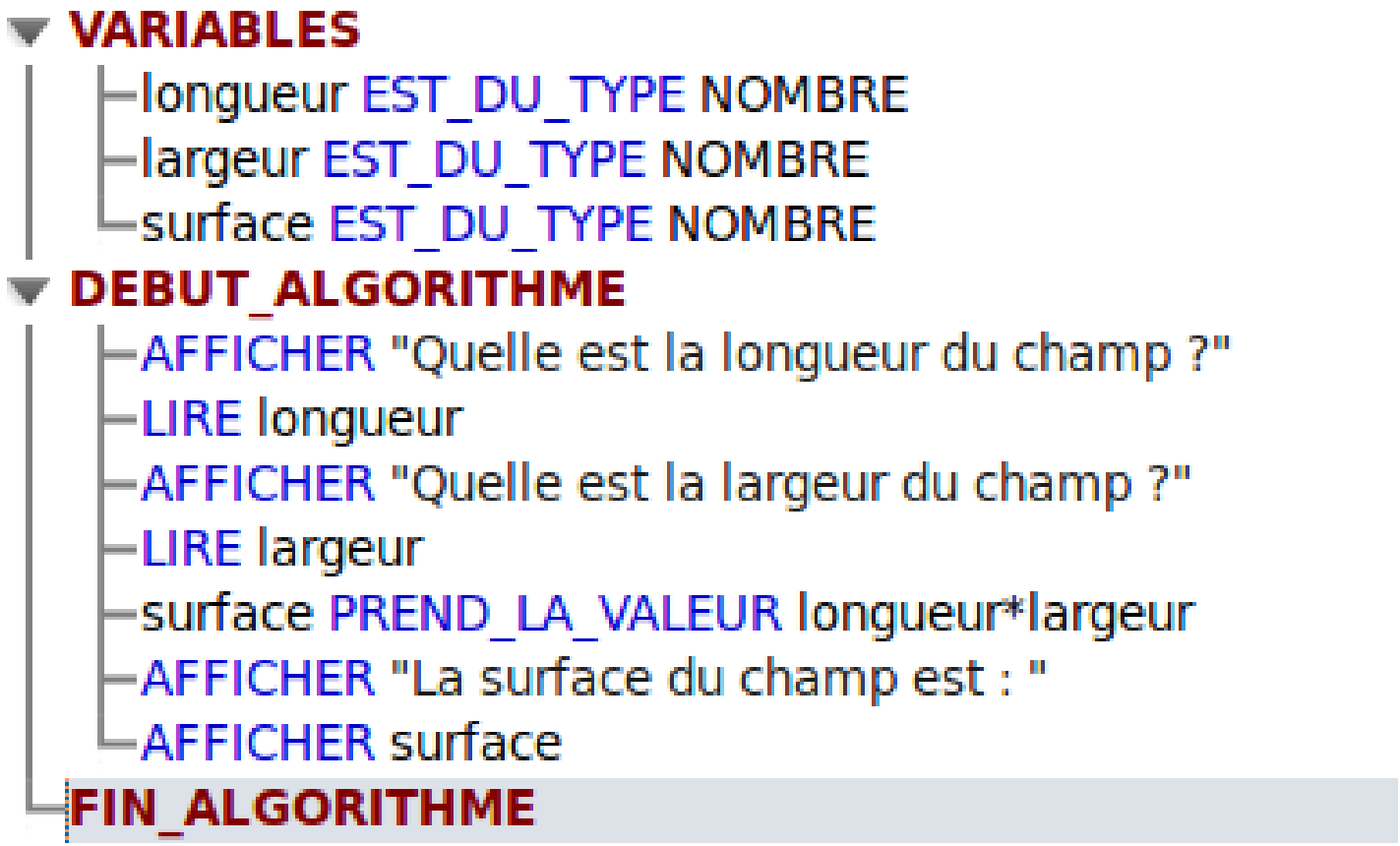
Lexique des variables

longueur	(réel)	Longueur du champ	INTERMÉDIAIRE
largeur	(réel)	Largeur du champ	INTERMÉDIAIRE
surface	(réel)	Superficie du champ	INTERMÉDIAIRE

Algorithme

longueur $\leftarrow$ lire
largeur $\leftarrow$ lire
surface $\leftarrow$ *longueur* $\times$ *largeur*
écrire *surface*

Exemple avec Algobox



Exemple en Visual Basic

Langage de programmation utilisé : Visual Basic (Alt+F11 depuis une feuille Excel par exemple...)

```
Sub champ()

    Dim longueur as Double
    Dim largeur as Double
    Dim surface as Double

      longueur=InputBox("Quelle est la longueur du champ ?")
      largeur=InputBox("Quelle est la largeur du champ ?")
      surface = longueur * largeur
      MsgBox("Le champ a une superficie de " & surface)

End Sub
```

Opérateurs

- Déjà vus : lire, écrire, $\leftarrow$ (affectation).
- Opérateurs arithmétiques : $+$, $-$, $\times$, $/$, mod (modulo ou reste de la division entière)

  $\Rightarrow$ notés comme en mathématiques (pas d'$*$!).

  Le type du résultat dépend du type des opérandes :
  - le résultat a toujours le type le plus complexe des opérandes
  - exemples :

    entier *op* entier $\Rightarrow$ entier
    entier *op* réel $\Rightarrow$ réel
    réel *op* entier $\Rightarrow$ réel
    réel *op* réel $\Rightarrow$ réel
  - en particulier :

    entier$/$entier $\Rightarrow$ entier (division entière !)

Opérateurs (suite)

- Opérateurs booléens : $\wedge$ (et), $\vee$ (ou), $\neg$ (non) dans $\{\text{vrai}, \text{faux}\}$
  - $A \wedge B$: vrai si A et B sont tous les deux vrai ;
  - $A \vee B$: vrai si A est vrai ou B est vrai ;
  - $\neg A$: inverse logique de A
- Opérateurs de comparaison : $\leq$, $\geq$, $<$, $>$, $=$, $\neq$
- Autres fonctions mathématiques courantes :

e^x	exponentielle
$\ln x$	logarithme
x^y	puissance
$\sqrt{x}$	racine carrée
$\lceil x \rceil$	partie entière supérieure
$\lfloor x \rfloor$	partie entière inférieure
$\lvert x \rvert$	valeur absolue

- Cf. TP pour la correspondance en VBA...

Plan

Lexique des constantes

- Valeurs utilisées mais non modifiées par l'algorithme, ni paramètre variable de celui-ci.
- Synopsis :
  - Mot clef « **Lexique des constantes** » suivi de la liste des constantes.
- Placé *avant* le lexique des variables.
- Les constantes sont définies par :

Nom	référence de la constante, habituellement tout en capitales
Type	nature de la valeur
Valeur	la valeur de la constante, connue *avant* l'exécution de l'algorithme et non modifiée par celui-ci
Description	un texte indiquant ce que représente la constante

Exemple

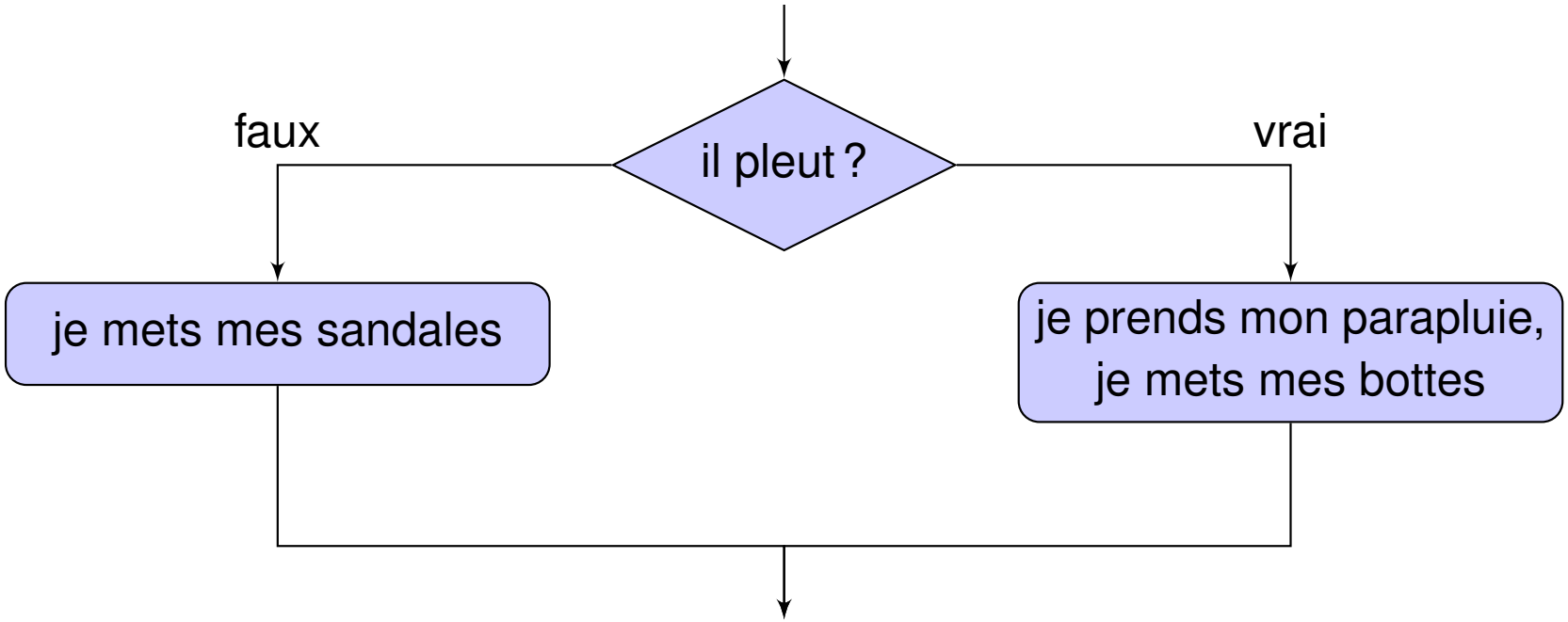
Lexique des constantes

TAUX_TVA (réel) = 19,6 Taux de TVA en vigueur

Plan

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. »



Les conditionnelles

- Synopsis :

si ⟨*condition*⟩ **alors**
| ⟨*instruction*⟩
| ⟨*instruction*⟩
| ...
sinon
| ⟨*instruction*⟩
| ⟨*instruction*⟩
| ...
fsi

si ⟨*condition*⟩ **alors**
| ⟨*instruction*⟩
| ⟨*instruction*⟩
| ...
fsi

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

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.
- Exemple :

Algorithme

si $\langle cond_1 \rangle$ **alors**

```
|  ...                  // cond1 vraie
```

sinon

```
|  si ⟨cond2⟩ alors                  // cond1 fausse
|  |  ...                //   cond2 vraie
|  sinon
|  |  ...                //   cond2 fausse
|  fsi
```

fsi

- L'ordre des implications est généralement important.

Conditionnelles de type cas

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

selon que ⟨*variable*⟩ **est**
  cas ⟨val_1⟩ **:**
    ⟨*instruction*⟩
    ...
  cas ⟨val_2⟩ **:**
    ⟨*instruction*⟩
    ...
  cas ⟨val_N⟩ **:**
    ⟨*instruction*⟩
    ...
  défaut :
    ⟨*instruction*⟩
    ...
fselon

- Les val_i sont des *constantes* toutes différentes, mais du même type que *variable*.
- Le cas *défaut* est optionnel.

Plan

Contrôles itératifs (boucles)

- Possibilité de *répéter* une suite d'instructions selon une condition donnée.
- Exemple : Euclide $\Rightarrow$ répéter... jusqu'à ce que $a = b$.
- On dispose de trois structures de contrôle différentes :

  tant que :
  - répète des instructions tant que la condition de la boucle est *vraie* ;
  - les instructions de la boucle peuvent *ne pas* être exécutées.

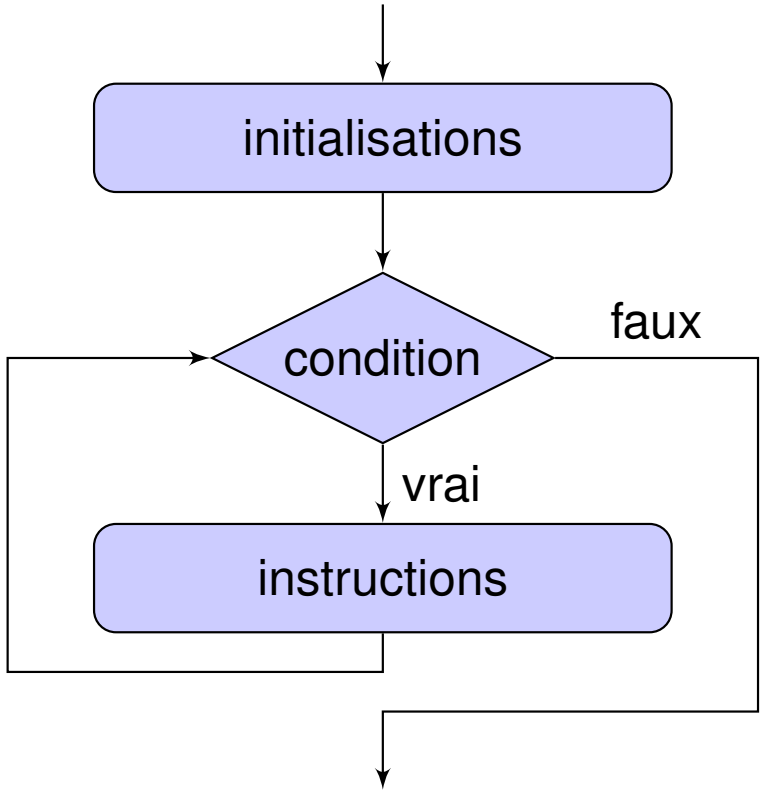
  pour :
  - répète des instructions un nombre *connu* de fois ;
  - la condition porte uniquement sur le nombre d'itérations à effectuer.

  répéter :
  - comme le *tant que*, mais on effectue *au moins une fois* les instructions de la boucle.

Structure *tant que*

- Synopsis :

⟨*initialisations*⟩
tant que ⟨*condition*⟩ **faire**
⟨*instruction*⟩
⟨*instruction*⟩
...
ftant



- Les *initialisations* spécifient les valeurs initiales des variables intervenant dans la *condition*.
- Il faut *au moins une* instruction dans la boucle susceptible de modifier la valeur de la condition.
- Les *instructions* de la boucle peuvent *ne pas* être exécutées.

Exemples

Exemple 1 : algorithme d'Euclide

Algorithme

```
// initialisation: a et b sont données
tant que a ≠ b faire                    // condition d'exécution de la boucle
    si a < b alors
        échanger(a, b)                  // échange des valeurs de a et b
    fsi
    c ← a − b
    // action modifiant la valeur d'au moins une variable
       intervenant dans la condition (a ← a − b)
    a ← c
ftant
```

- Utilisé pour effectuer un nombre inconnu (mais fini) d'itérations.
- Il faut s'assurer que la boucle *termine* et donc que la condition devient fausse à un moment donné !

Exemple de boucle infinie

Boucle infinie

Algorithme

$a \leftarrow 1$
tant que $a > 0$ **faire**
    écrire "Bonjour"
    $a \leftarrow a + 1$
ftant

- La condidtion $a > 0$ ne sera jamais fausse puisque $a \leftarrow a + 1$ et que a vaut 1 au début de l'algorithme.
- Dans ce cas "Bonjour" est affiché une infinité de fois...

Exemple 1 avec Algobox

ALGOBOX : PGCD_BIS

CODE DE L'ALGORITHME :

```
1     VARIABLES
2       a EST_DU_TYPE NOMBRE
3       b EST_DU_TYPE NOMBRE
4       temp EST_DU_TYPE NOMBRE
5     DEBUT_ALGORITHME
6       AFFICHER "Entrez la première valeur"
7       LIRE a
8       AFFICHER "Entrez la deuxième valeur"
9       LIRE b
10      TANT_QUE (a!=b) FAIRE
11        DEBUT_TANT_QUE
12        SI (a<b) ALORS
13          DEBUT_SI
14          temp PREND_LA_VALEUR a
15          a PREND_LA_VALEUR b
16          b PREND_LA_VALEUR temp
17          FIN_SI
18        temp PREND_LA_VALEUR a-b
19        a PREND_LA_VALEUR temp
20        FIN_TANT_QUE
21      AFFICHER "PGCD(a,b)="
22      AFFICHER a
23    FIN_ALGORITHME
```

RÉSULTATS :

```
***Algorithme lancé***
Entrez la première valeur
Entrer a : 174
Entrez la deuxième valeur
Entrer b : 72
PGCD(a,b)=6
***Algorithme terminé***
```

Généré par AlgoBox

Exemples

Exemple 2 : conversion de °F en °C

Algorithme

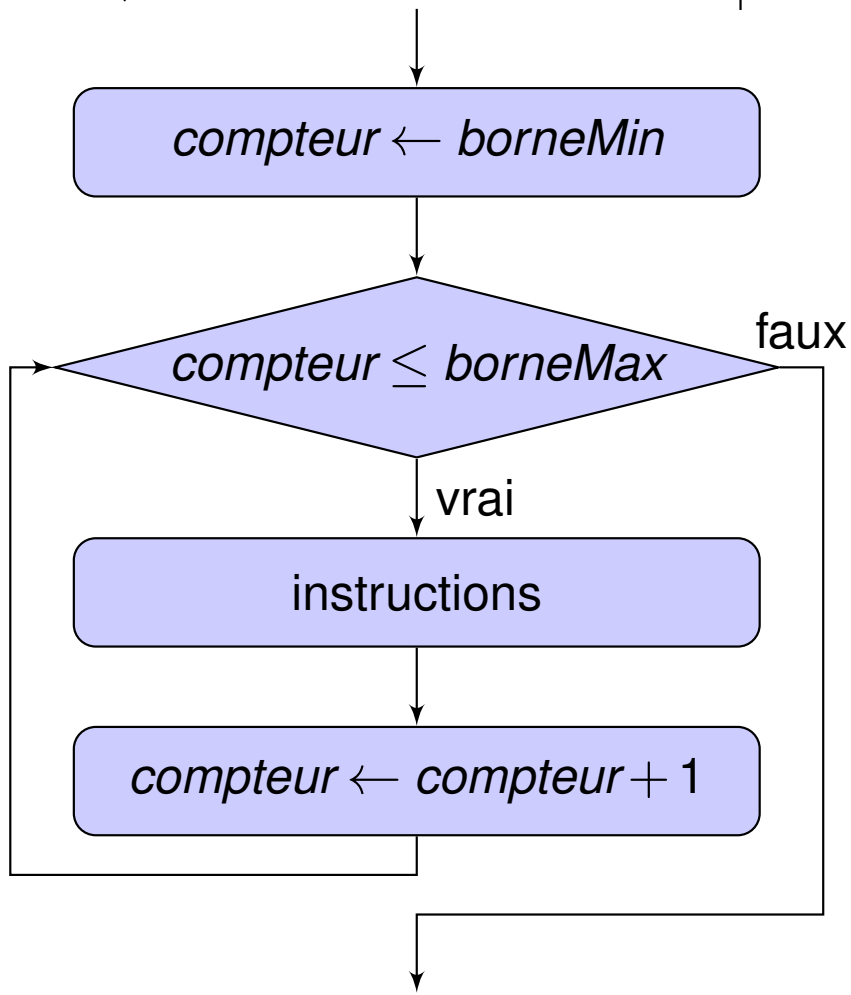
nbTemp ← lire
i ← 1 // initialisation de la boucle
tant que *i* ≤ *nbTemp* **faire** // condition d'exécution de la boucle
  fahr ← lire
  celsius ← (*fahr* − 32)/9
  écrire "La température ", *fahr*, " en °F est ", *celsius*, " en °C"
  // instruction de modification éventuelle de la condition
  i ← *i* + 1
ftant

- Utilisé dans ce cas pour effectuer un nombre *connu* d'itérations.
- Lorsque le nombre d'itérations est connu, on peut utiliser la structure *pour*...

Structure *pour*

- Synopsis :

pour ⟨*compteur*⟩ **de** ⟨*borneMin*⟩ **à** ⟨*borneMax*⟩ **faire**
⟨*instruction*⟩
⟨*instruction*⟩
...
fpour



- Lorsque la répétition ne porte que sur *le nombre* d'itérations et qu'il est connu *avant* de commencer la boucle, on peut utiliser une *écriture plus condensée* que le *tant que*, c'est la structure de contrôle *pour*.

Exemples

Exemple 1 : conversion de °F en °C

Algorithme

nbTemp ← lire
pour *i* **de** 1 **à** *nbTemp* **faire**
  fahr ← lire
  celsius ← (*fahr* − 32)/9
  écrire "`La température `", *fahr*, "` en °F est `", *celsius*, "` en °C`"
fpour

- Le *pour* cache l'initialisation et la mise à jour du compteur.
- Il permet aussi d'éviter des erreurs et des oublis.
- Il ne faut pas modifier le compteur dans la boucle.
- On peut utiliser la valeur du compteur dans la boucle.

Exemples

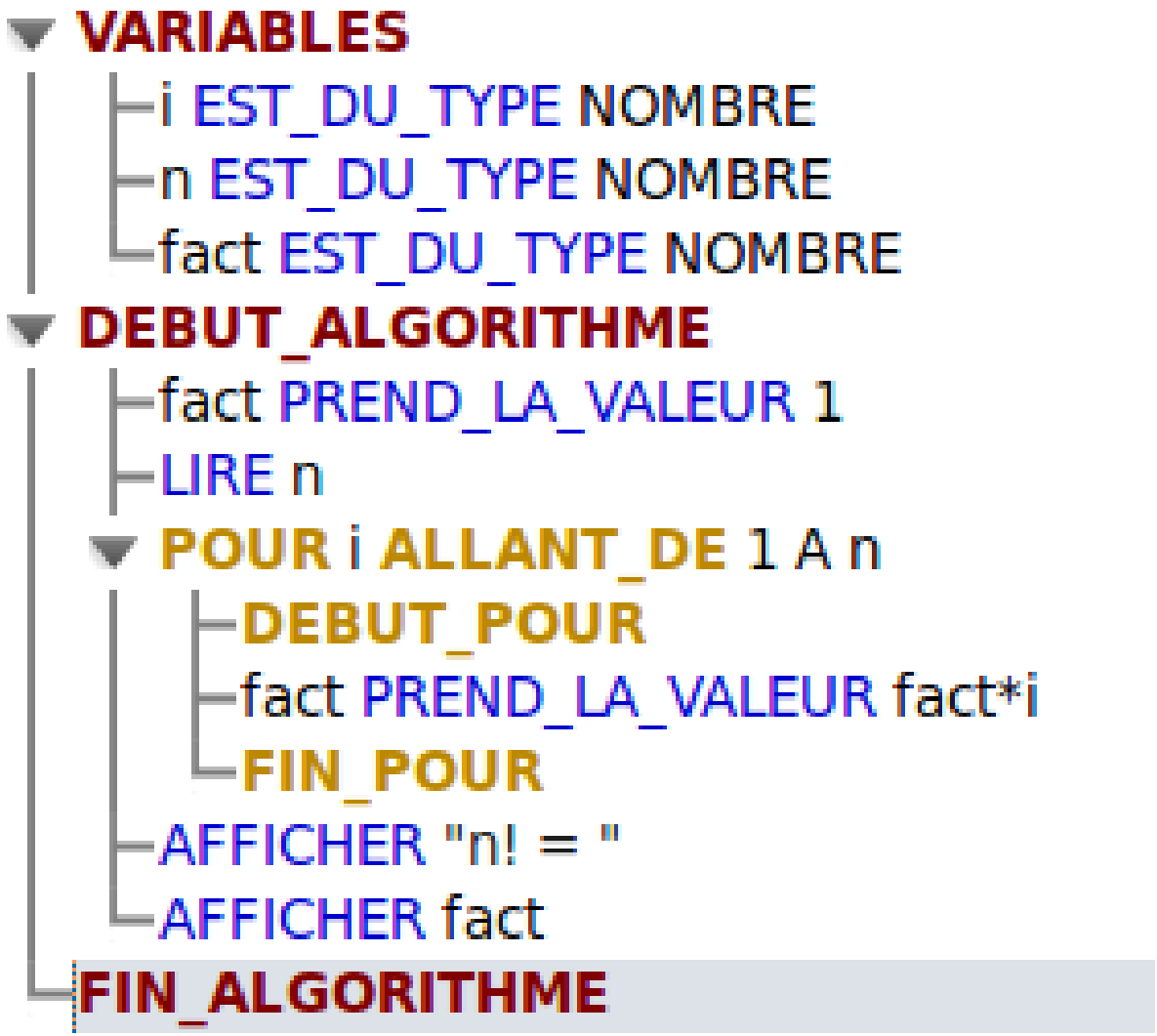
Exemple 2 : les factorielles

Algorithme

$n \leftarrow$ lire
$fact \leftarrow 1$
pour i **de** 1 **à** n **faire**
    $fact \leftarrow fact \times i$
    écrire $fact$
fpour

- Utilisation de la valeur de i dans la boucle.
- Initialisation du ou des élément(s) calculé(s) dans la boucle, ici la variable *fact*.
- Si on place l'affichage en dehors de la boucle (après le *fpour*), on affiche uniquement la dernière valeur calculée.

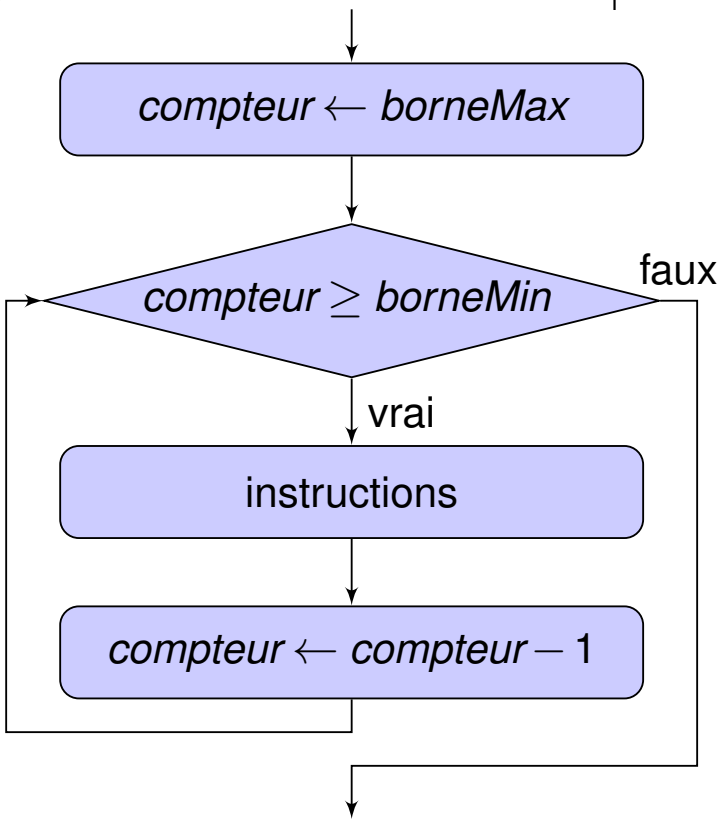
Exemple avec Algobox



Sens du pour

- Par défaut, une boucle *pour* va dans le sens *croissant*.
- On peut inverser le sens de parcours.
- Synopsis :

pour ⟨*compteur*⟩ **de** ⟨*borneMax*⟩ **à** ⟨*borneMin*⟩ **en descendant faire**
  ⟨*instruction*⟩
  ⟨*instruction*⟩
  ...
fpour

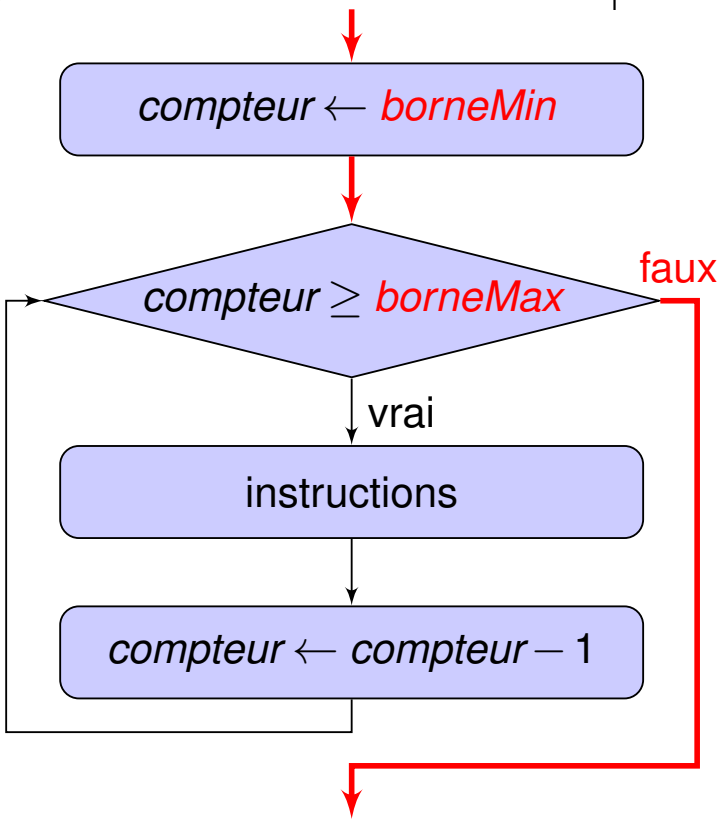


- On n'entre pas dans la boucle si ses bornes sont dans l'ordre inverse de son sens.

Sens du pour

- Par défaut, une boucle *pour* va dans le sens *croissant*.
- On peut inverser le sens de parcours.
- Synopsis :

pour ⟨*compteur*⟩ **de** ⟨*borneMax*⟩ **à** ⟨*borneMin*⟩ **en descendant faire**
  ⟨*instruction*⟩
  ⟨*instruction*⟩
  ...
fpour



- On n'entre pas dans la boucle si ses bornes sont dans l'ordre inverse de son sens.

Exemple (en descendant)

Exemple : compte à rebours

Données

/

Résultat

Affiche un compte à rebours pendant 10 secondes

Lexique des variables

i	(entier)	compteur	INTERMÉDIAIRE

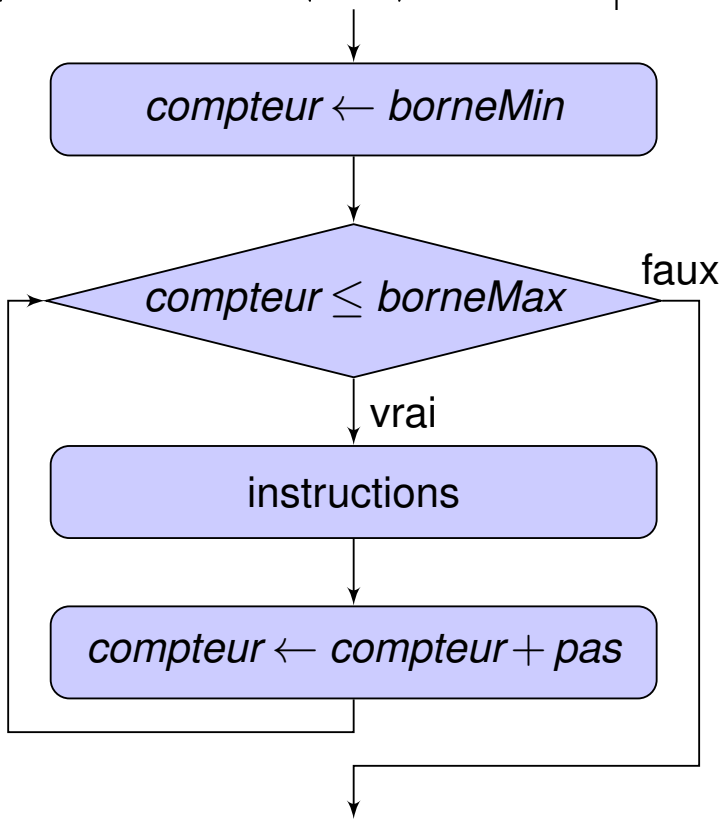
Algorithme

pour *i* **de** 10 **à** 1 **en descendant faire**
  écrire *i*
  attendre une seconde
fpour
écrire "BONNE ANNÉE !"

Incrément du pour

- On appelle « le *pas* » d'une boucle *pour* son incrément.
- Par défaut, le pas d'une boucle *pour* est de 1 ; mais on peut le changer.
- Synopsis :

pour ⟨*compteur*⟩ **de** ⟨*borneMin*⟩ **à** ⟨*borneMax*⟩ **par pas de** ⟨*pas*⟩ **faire**
  ⟨*instruction*⟩
  ⟨*instruction*⟩
  ...
fpour



- Le pas d'une boucle doit toujours être un nombre positif.
- Dans le cas d'une boucle *en descendant*, le *pas* correspond à son décrément.

Exemple (par pas de)

Exemple : énumérer les nombres pairs, puis impairs

Données

/

Résultat

Affiche les nombres pairs de 2 à 100, puis les nombres impairs de 99 à 1.

Lexique des variables

i (entier) compteur INTERMÉDIAIRE

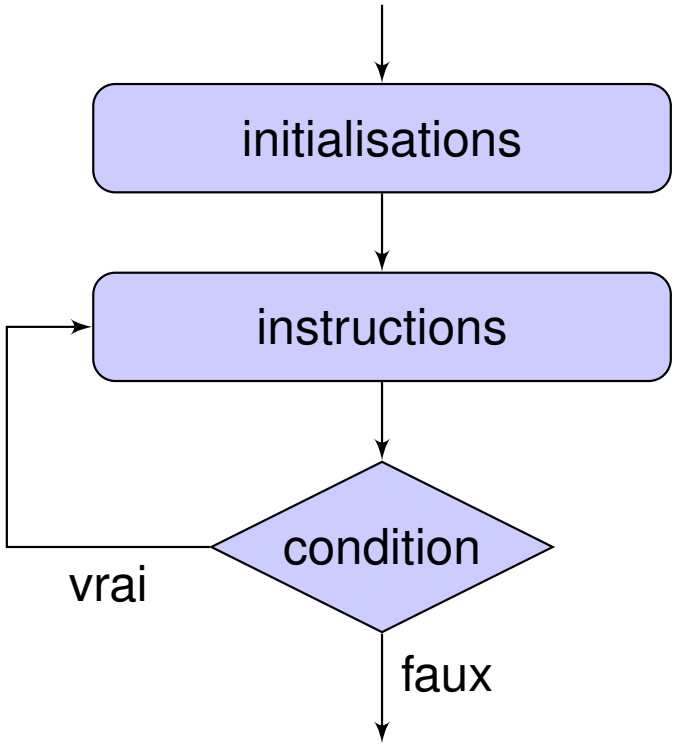
Algorithme

pour *i* **de** 2 **à** 100 **par pas de** 2 **faire**
  écrire *i*
fpour
pour *i* **de** 99 **à** 1 **en descendant par pas de** 2 **faire**
  écrire *i*
fpour

Structure *répéter*

- Similaire au *tant que*, mais exécutée au moins une fois.
- Synopsis :

⟨*initialisations*⟩
répéter
    ⟨*instruction*⟩
    ⟨*instruction*⟩
    ...
tant que ⟨*condition*⟩



- Pratique, par exemple, pour la vérification des lectures de données :
  - répéter la lecture tant que celle-ci n'est pas valide.

Exemple

Exemple : lire un nombre entre 1 et 100

Données

/

Résultat

Un nombre lu, compris entre 1 et 100.

Lexique des variables

nombre	(entier)	le nombre lu	RÉSULTAT

Algorithme

répéter
  nombre $\leftarrow$ lire
tant que $\neg(1 \leq nombre \wedge nombre \leq 100)$
// ici, $1 \leq nombre \leq 100$

Boucles imbriquées

- On peut placer n'importe quel type de boucle dans une autre.

Exemple

Algorithme

$n \leftarrow$ lire
pour i **de** 1 **à** n **faire**
  répéter
    $a \leftarrow$ lire
  tant que $a \leq 0$
  répéter
    $b \leftarrow$ lire
  tant que $b \leq 0$
  ...

(suite)

  ...
  tant que $a \neq b$ **faire**
    si $a < b$ **alors**
      échanger(a, b)
    fsi
    $a \leftarrow a - b$
  ftant
  écrire "`PGCD = `", a
fpour

Boucles imbriquées

- On peut placer n'importe quel type de boucle dans une autre.

Exemple

Algorithme

$n \leftarrow$ lire
pour i **de** 1 **à** n **faire**
  répéter
    $a \leftarrow$ lire
  tant que $a \leq 0$
  répéter
    $b \leftarrow$ lire
  tant que $b \leq 0$
  ...

(suite)

...
  tant que $a \neq b$ **faire**
    si $a < b$ **alors**
      échanger(a, b)
    fsi
    $a \leftarrow a - b$
  ftant
  écrire "PGCD = ", a
fpour

- Test de validité sur n non primordial.

Boucles imbriquées

- On peut placer n'importe quel type de boucle dans une autre.

Exemple

Algorithme

```
n ← lire
pour i de 1 à n faire
    répéter
        a ← lire
    tant que a ≤ 0
    répéter
        b ← lire
    tant que b ≤ 0
    ...
```

(suite)

```
    ...
    tant que a ≠ b faire
        si a < b alors
            échanger(a, b)
        fsi
        a ← a − b
    ftant
    écrire "PGCD = ", a
fpour
```

- On est sûr que $a > 0$ et $b > 0$.
- Primordial, sinon la boucle *tant que* qui suit est infinie !

Boucles imbriquées

- On peut placer n'importe quel type de boucle dans une autre.

Exemple

Algorithme

```
n ← lire
pour i de 1 à n faire
    répéter
        a ← lire
    tant que a ≤ 0
    répéter
        b ← lire
    tant que b ≤ 0
    ...
```

(suite)

```
    ...
    tant que a ≠ b faire
        si a < b alors
            échanger(a, b)
        fsi
        a ← a − b
    ftant
    écrire "PGCD = ", a
fpour
```

- On est sûr que $a > 0$ et $b > 0$.
- Primordial, sinon la boucle *tant que* qui suit est infinie !

Autre exemple

- Les boucles, lorsqu'elles sont imbriquées, permettent de démultiplier le nombre d'opérations effectuées.

Exemple :

Affiche toutes les multiplications de 1×1 à 100×100.

Algorithme

pour i **de** 1 **à** 100 **faire**
  pour j **de** 1 **à** 100 **faire**
    $m \leftarrow i \times j$
    écrire "␣", m ;
  fpour
  écrire "\n"
fpour

Sorties :

```
1 2 3 ... 100                (i = 1)
2 4 6 ... 200                (i = 2)
3 6 9 ... 300                (i = 3)
...
100 200 300 ... 10000        (i = 100)
```

- On effectue ici 10 000 fois les opérations du cœur des boucles !

Arrêt sur la fin des données

- Parfois, on souhaite lire des données tant que l'utilisateur en fournit.
- La lecture renvoie la constante spéciale EOF quand il n'y a plus de données.
- Cette valeur est placée dans la variable lue.

Exemple

Algorithme

$n \leftarrow$ lire
tant que $n \neq$ EOF **faire**
  $fact \leftarrow 1$
  pour i **de** 1 **à** n **faire**
    $fact \leftarrow fact \times i$
  fpour
  écrire $fact$
  $n \leftarrow$ lire
ftant

Exemple : triangle de Pascal

Énoncé

- On souhaite afficher les n premières lignes du triangle de Pascal, n étant donné.
- Par exemple, avec $n = 8$:

		numéro de colonne							
		0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	0	1							
	1	1	1						
	2	1	2	1					
numéro	*3*	1	3	3	1				
de ligne	*4*	1	4	6	4	1			
	5	1	5	10	10	5	1		
	6	1	6	15	20	15	6	1	
	7	1	7	21	35	35	21	7	1

Exemple : triangle de Pascal

Rappel de la démarche

1. comprendre l'énoncé ;
2. fixer les données et les résultats de l'algorithme (*ce qu'il doit faire*) ;
3. trouver une idée de l'algorithme (*comment le faire*) ;
4. spécifier précisément l'algorithme, avec son lexique des variables, etc. ;
5. (éventuellement) écrire un programme qui implémente l'algorithme.

Données et résultat

Données

Un nombre de lignes : n.

Résultat

Affiche les n premières lignes du triangle de Pascal.

Exemple : triangle de Pascal

Recherche d'idée

- L'affichage doit se faire ligne après ligne.
- La n-ième ligne contient n nombres.
- À la ligne n et à la colonne k, on doit trouver le nombre :

$$\binom{n}{k} = \frac{n!}{k! \times (n-k)!}$$

- On pourrait utiliser l'algorithme de calcul de factorielle, mais on ne le souhaite pas !
  En effet, 3 factorielles à calculer pour chaque nombre $\Rightarrow$ 3 boucles $\Rightarrow$ peut être coûteux (en temps de calcul).
- On peut remarquer que toutes les lignes commencent par 1 car, par définition, $\binom{n}{0} = 1$, et que...

Exemple : triangle de Pascal

Recherche d'idée (suite)

- ... et qu'on a la relation :

$$\begin{aligned}\binom{n}{k} &= \frac{n!}{k! \times (n-k)!} \\ &= \frac{n!}{k \times (k-1)! \times \frac{(n-k+1)!}{(n-k+1)}} \\ &= \frac{n-k+1}{k} \times \frac{n!}{(k-1)! \times (n-(k-1))!} \\ &= \frac{n-k+1}{k} \times \binom{n}{k-1}\end{aligned}$$

- On a donc une relation entre chaque nombre et son précédent sur la ligne.

Exemple : triangle de Pascal

Idée de l'algorithme

Idée

pour chaque ligne n° i, de 0 à $n-1$:

- le premier nombre de la ligne est 1 ;
- afficher ce nombre ;
- pour chaque colonne n° j, de 1 à i :
  - trouver le nouveau nombre, en multipliant le nombre précédent par $(i-j+1)/j$;
  - afficher ce nombre ;
- passer à la ligne suivante.

Exemple : triangle de Pascal

L'algorithme (enfin !)

Lexique des variables

n	(entier)	le nombre de lignes à afficher	DONNÉE

Exemple : triangle de Pascal

L'algorithme (enfin !)

Lexique des variables

n	(entier)	le nombre de lignes à afficher	DONNÉE
i	(entier)	numéro de ligne courant	INTERMÉDIAIRE
j	(entier)	numéro de colonne courant	INTERMÉDIAIRE
x	(entier)	le nombre courant dans le triangle	INTERMÉDIAIRE

Exemple : triangle de Pascal

L'algorithme (enfin !)

Lexique des variables

n	(entier)	le nombre de lignes à afficher	DONNÉE
i	(entier)	numéro de ligne courant	INTERMÉDIAIRE
j	(entier)	numéro de colonne courant	INTERMÉDIAIRE
x	(entier)	le nombre courant dans le triangle	INTERMÉDIAIRE

Exemple : triangle de Pascal

L'algorithme (enfin !)

Lexique des variables

n	(entier)	le nombre de lignes à afficher	DONNÉE
i	(entier)	numéro de ligne courant	INTERMÉDIAIRE
j	(entier)	numéro de colonne courant	INTERMÉDIAIRE
x	(entier)	le nombre courant dans le triangle	INTERMÉDIAIRE

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire "␣", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps

n	4
i	?
j	?
x	?

Sorties

–

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire "␣", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps

n	4
i	? 0
j	?
x	?

Sorties

–

Algorithme

```
pour i de 0 à n − 1 faire
    x ← 1
    écrire x
    pour j de 1 à i faire
        x ← (x × (i − j + 1))/j
        écrire " ", x
    fpour
    écrire "\n"
fpour
```

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps

n	4
i	? 0
j	?
x	? 1

Sorties

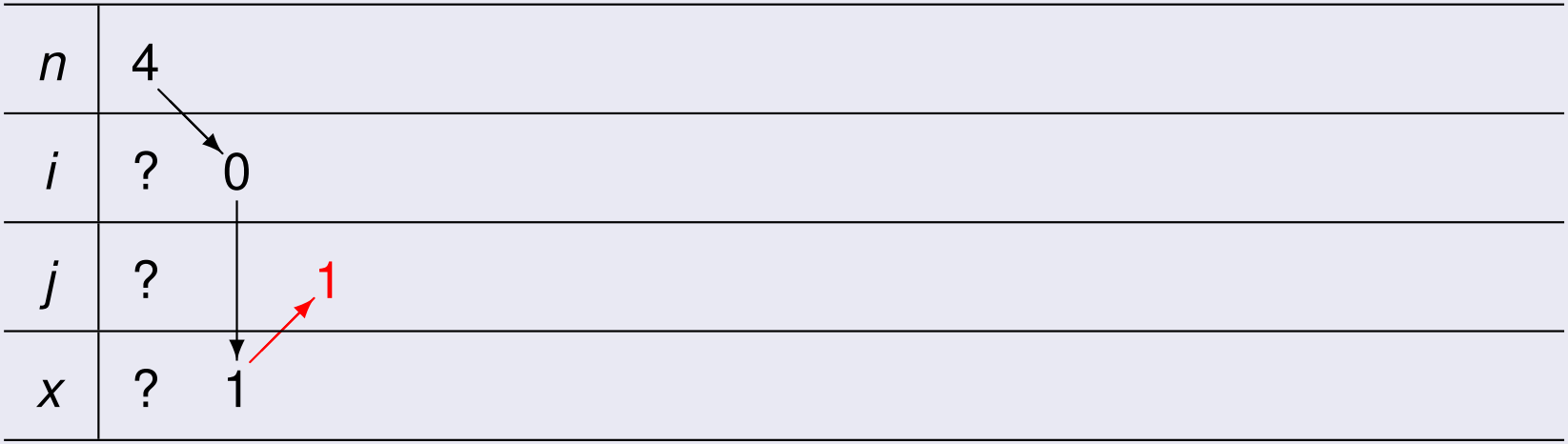
1_

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire "␣", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

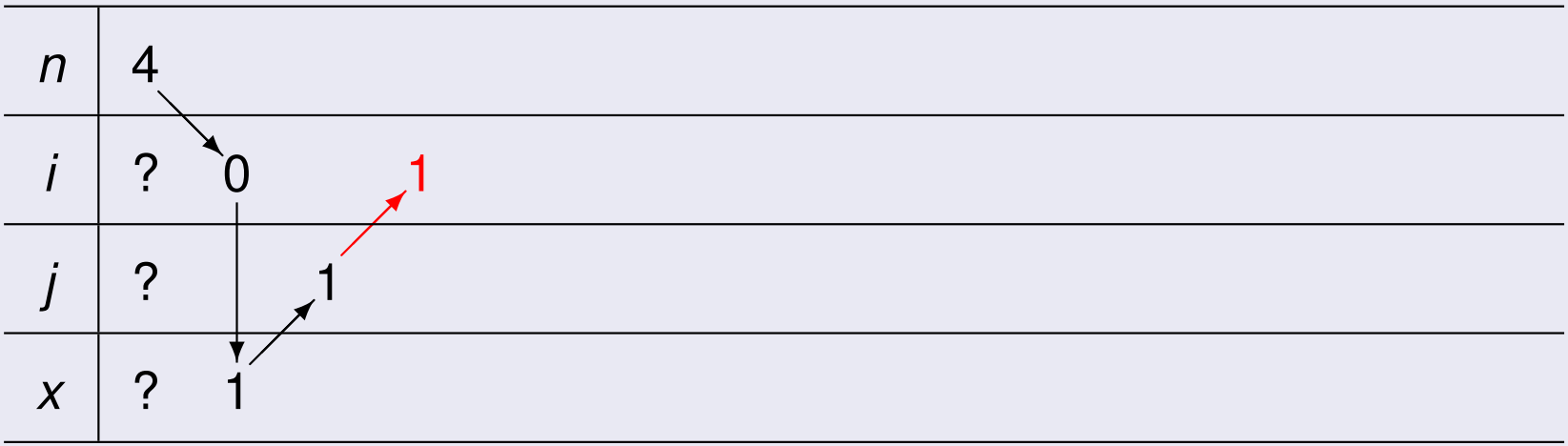
```
1

_
```

Algorithme

```
pour i de 0 à n − 1 faire
    x ← 1
    écrire x
    pour j de 1 à i faire
        x ← (x × (i − j + 1))/j
        écrire "␣", x
    fpour
    écrire "\n"
fpour
```

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

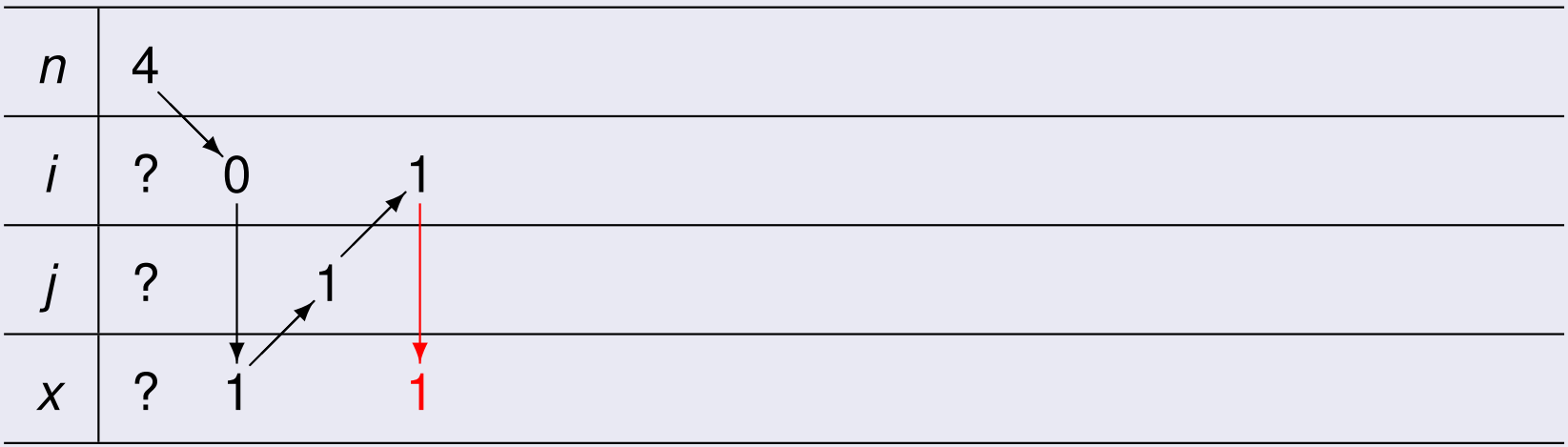
```
1

_
```

Algorithme

```
pour i de 0 à n − 1 faire
    x ← 1
    écrire x
    pour j de 1 à i faire
        x ← (x × (i − j + 1))/j
        écrire "␣", x
    fpour
    écrire "\n"
fpour
```

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

1

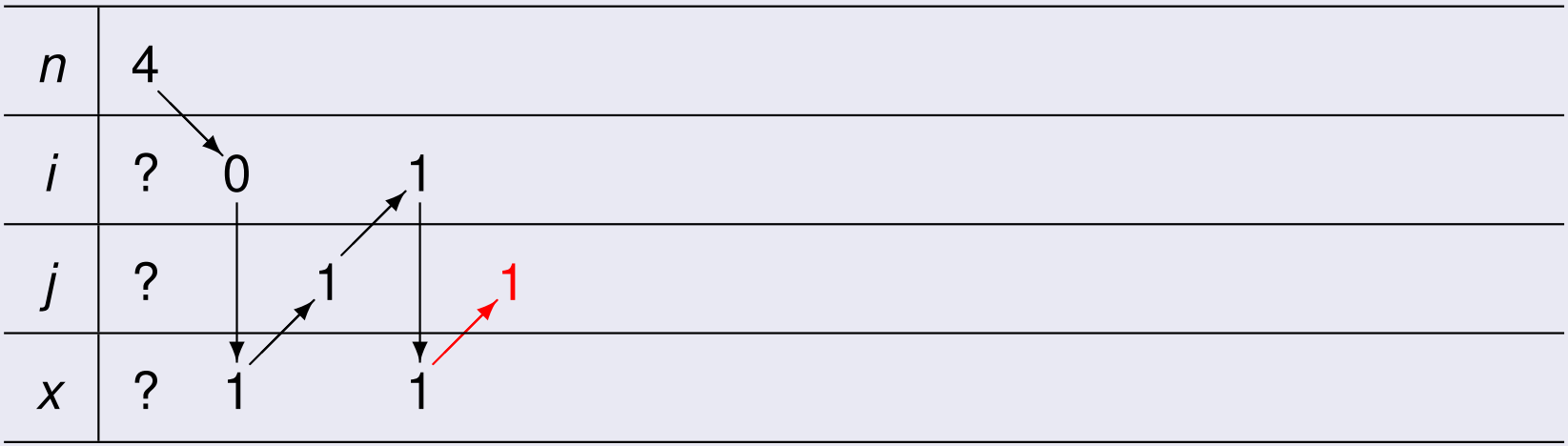
1_

Algorithme

```
pour i de 0 à n − 1 faire
    x ← 1
    écrire x
    pour j de 1 à i faire
        x ← (x × (i − j + 1))/j
        écrire " ", x
    fpour
    écrire "\n"
fpour
```

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

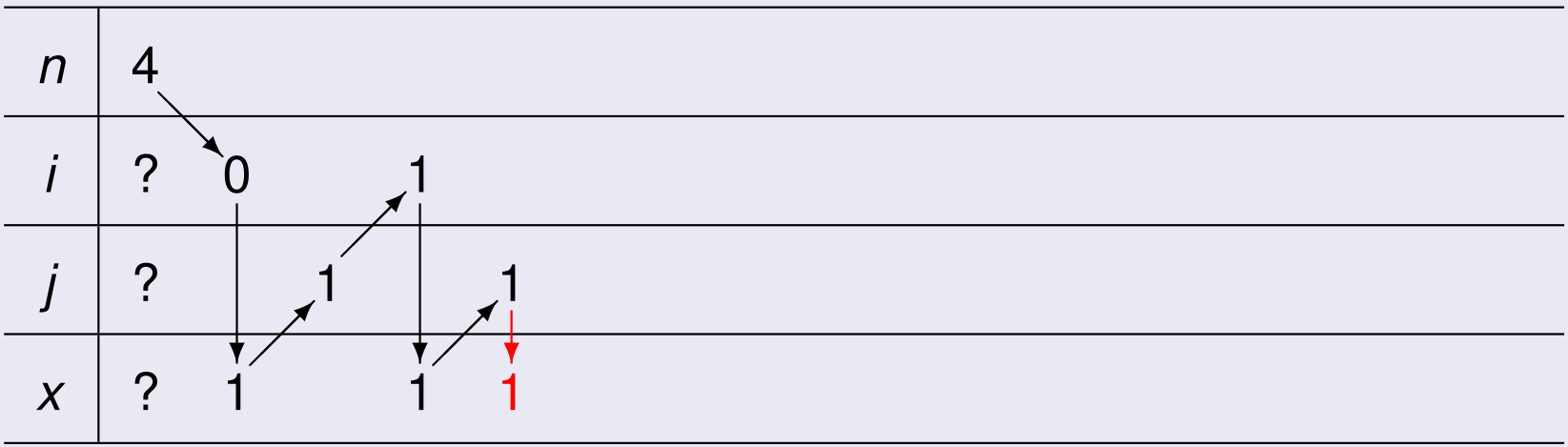
```
1
1_
```

Algorithme

```
pour i de 0 à n − 1 faire
    x ← 1
    écrire x
    pour j de 1 à i faire
        x ← (x × (i − j + 1))/j
        écrire "␣", x
    fpour
    écrire "\n"
fpour
```

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

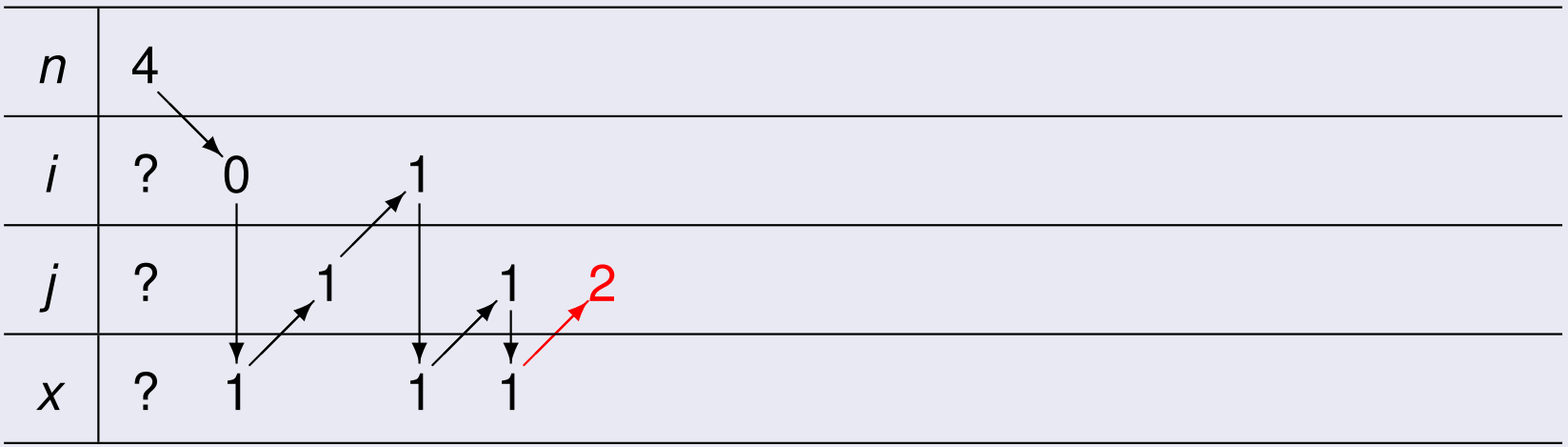
```
1
1␣1_
```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire "␣", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

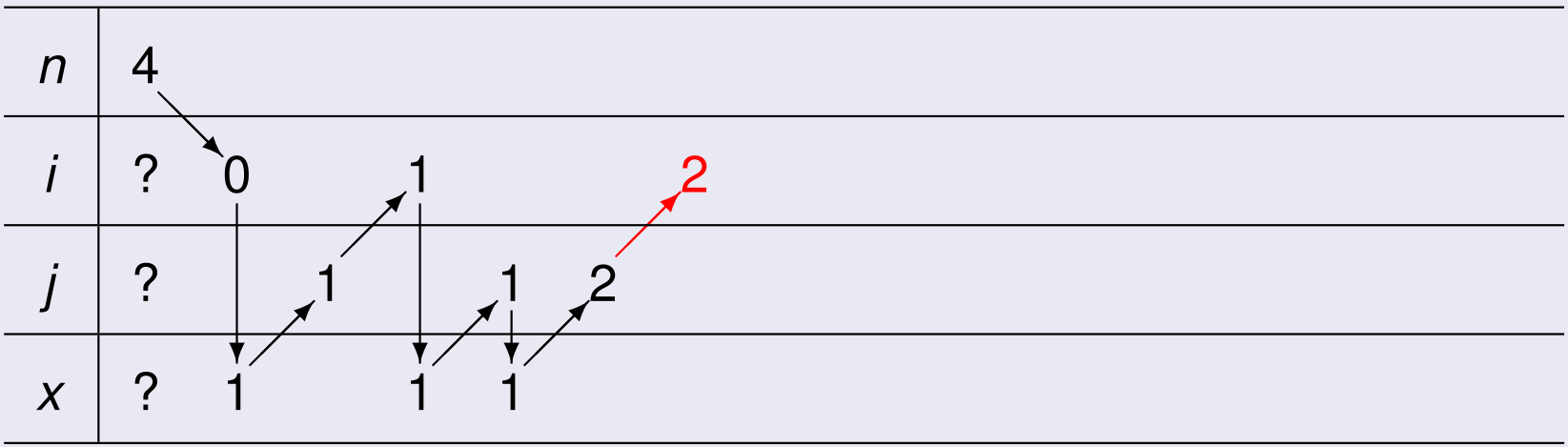
```
1
1␣1
—
```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire "␣", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

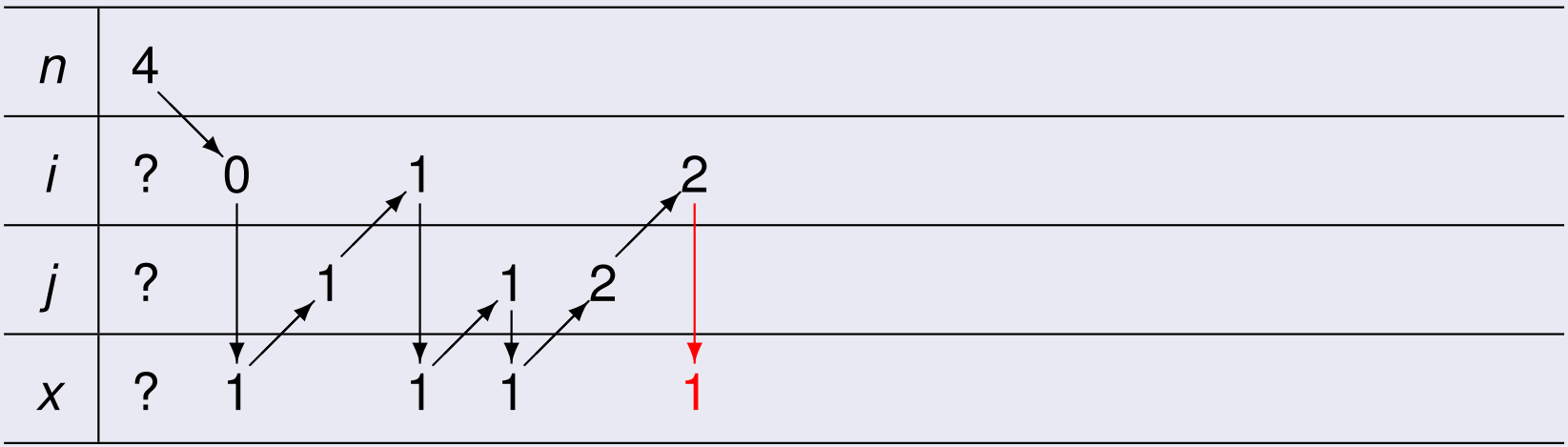
```
1
1␣1
—
```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire "␣", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

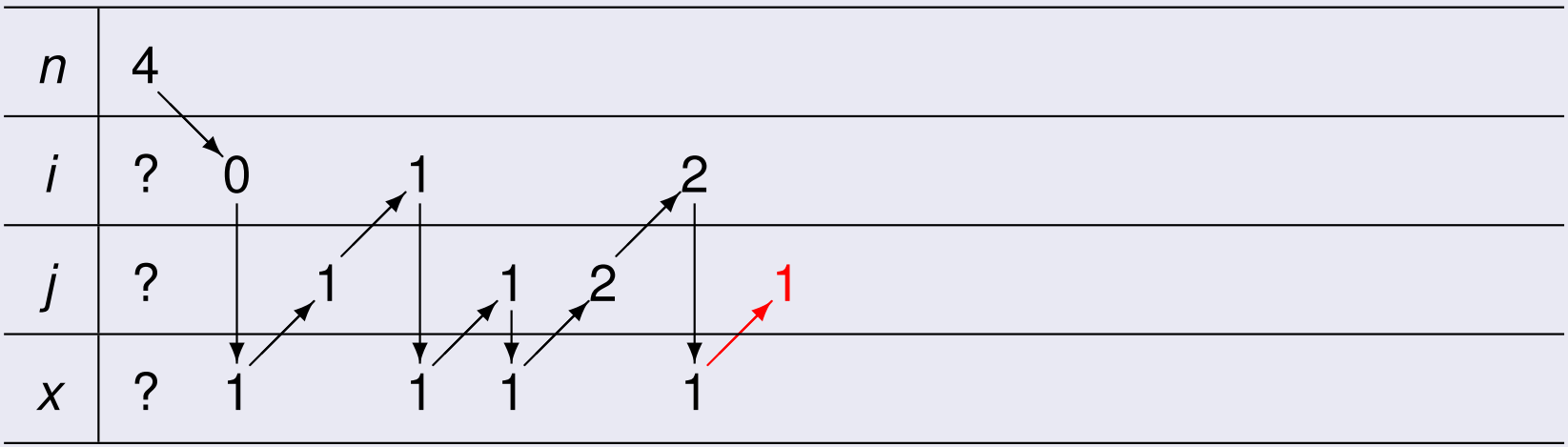
```
1
1 1
1
```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire " ", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

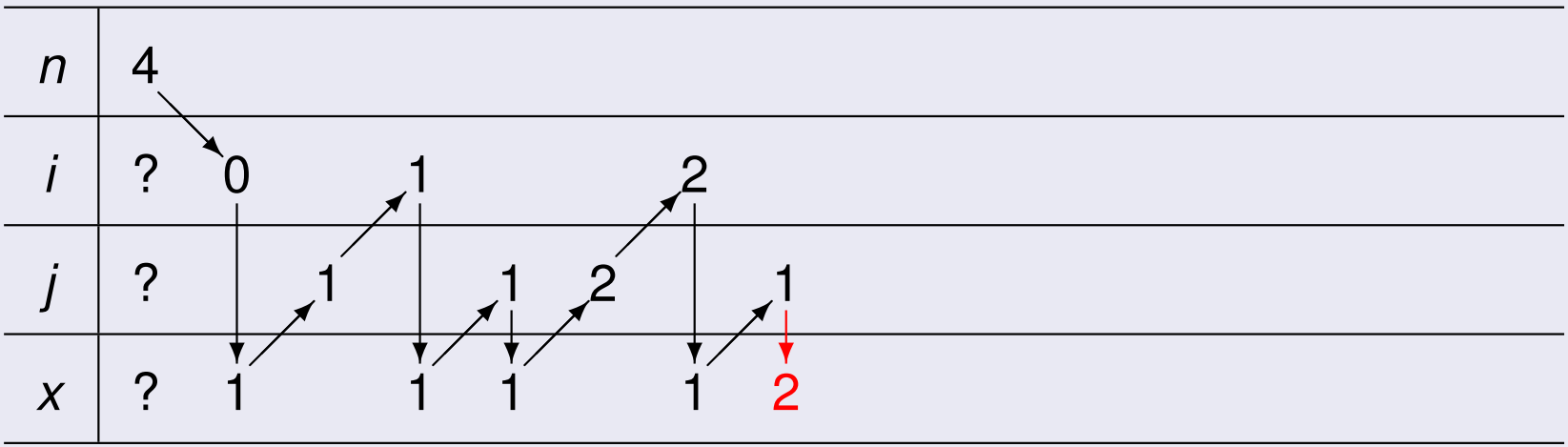
```
1
1 1
1
```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire " ", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

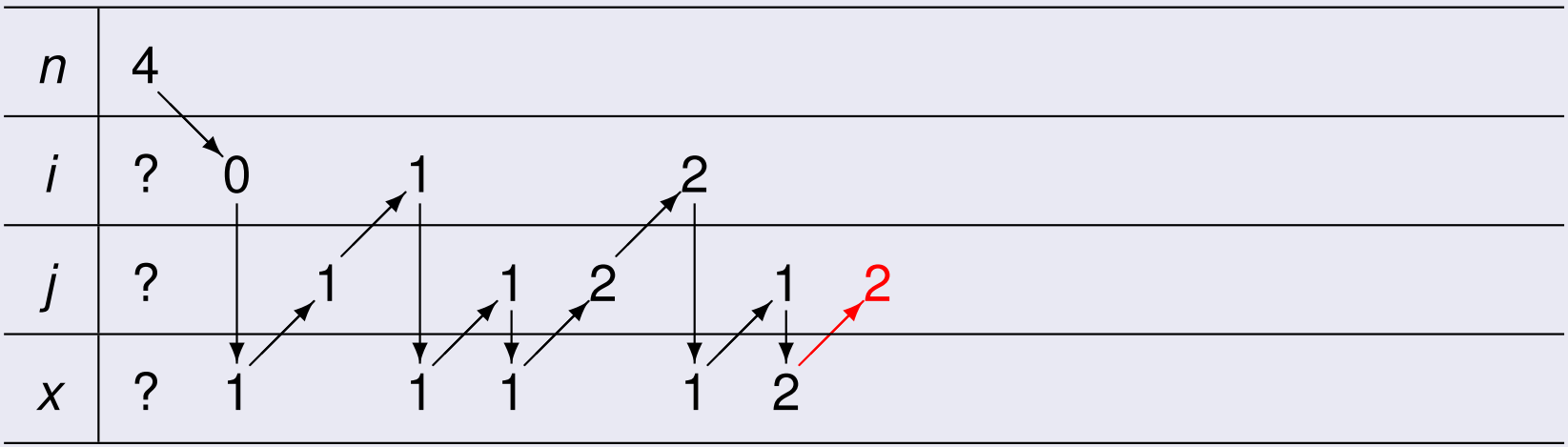
```
1
1␣1
1␣2_
```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire "␣", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

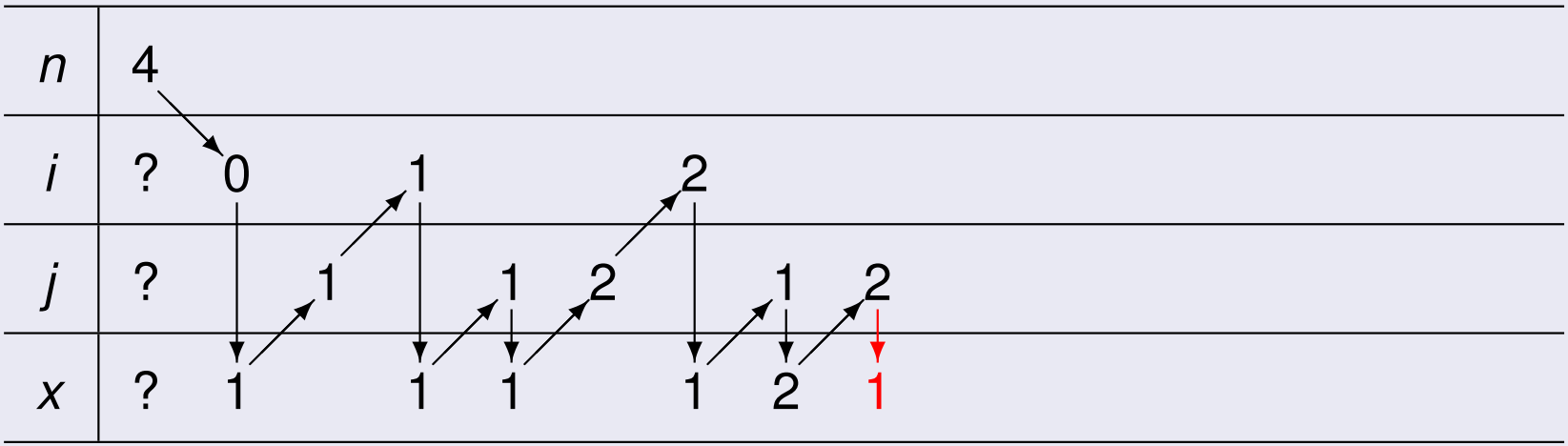
```
1
1 1
1 2
```

Algorithme

```
pour i de 0 à n − 1 faire
    x ← 1
    écrire x
    pour j de 1 à i faire
        x ← (x × (i − j + 1))/j
        écrire " ", x
    fpour
    écrire "\n"
fpour
```

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

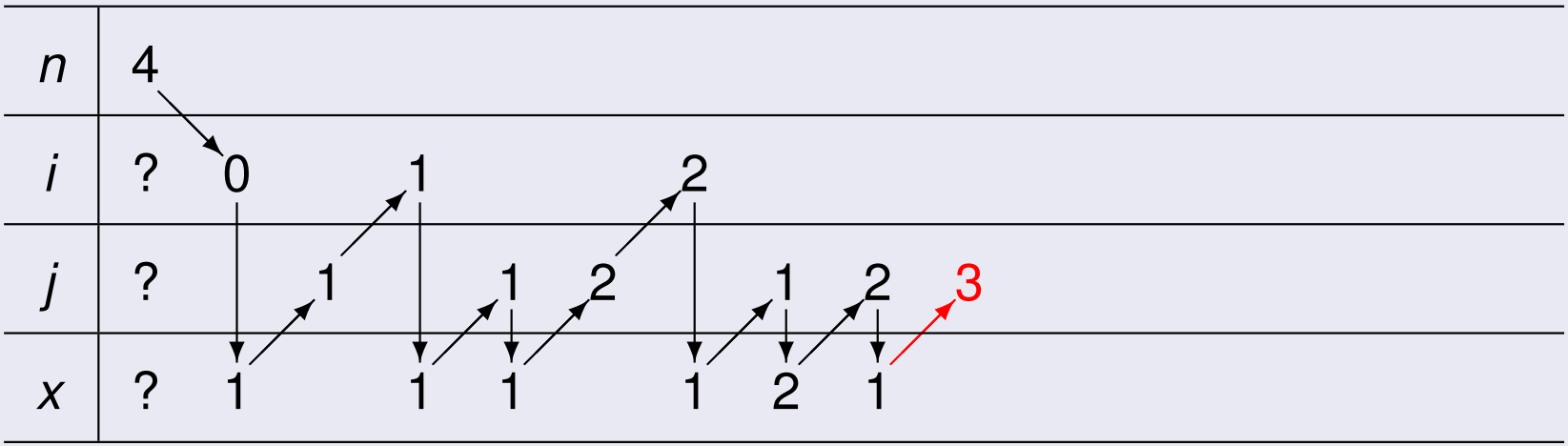
```
1
1 1
1 2 1
```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire " ", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

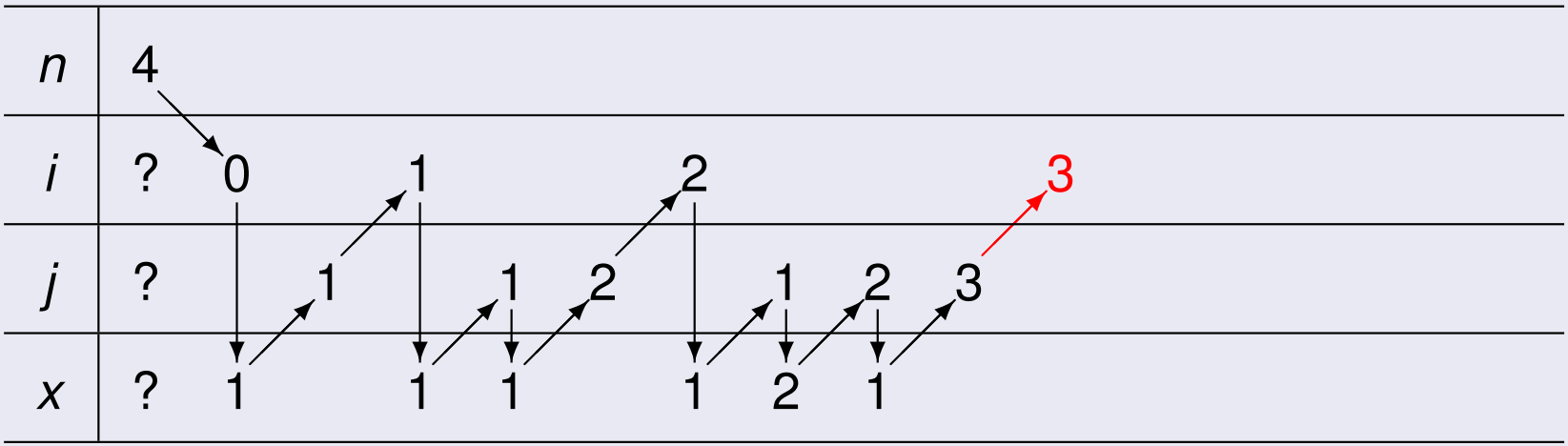
```
1
1 1
1 2 1
_
```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire "␣", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

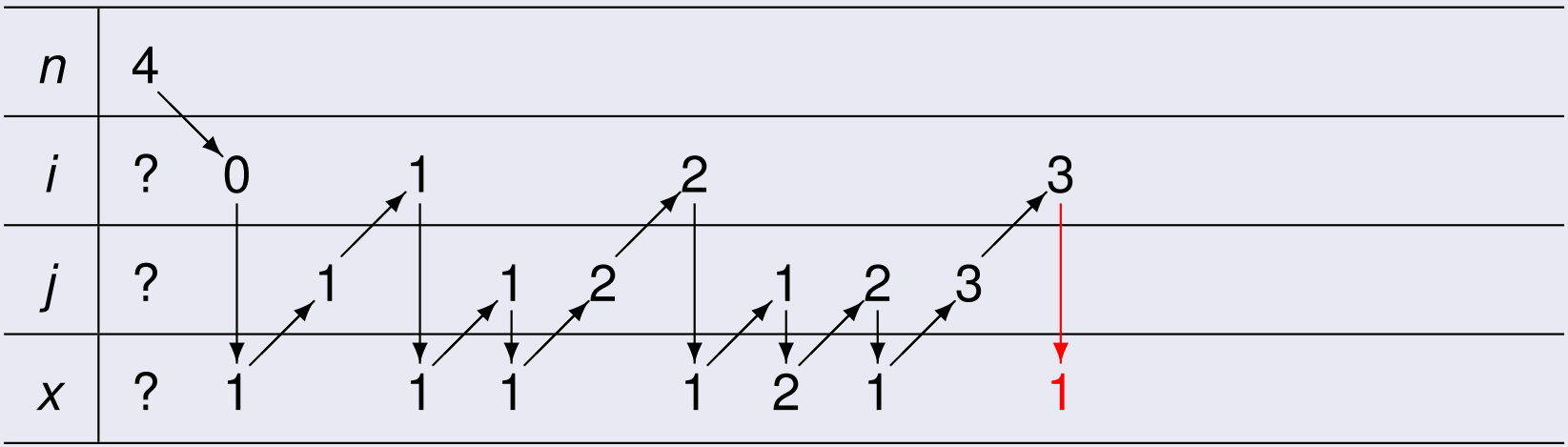
```
1
1 1
1 2 1

```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire " ", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

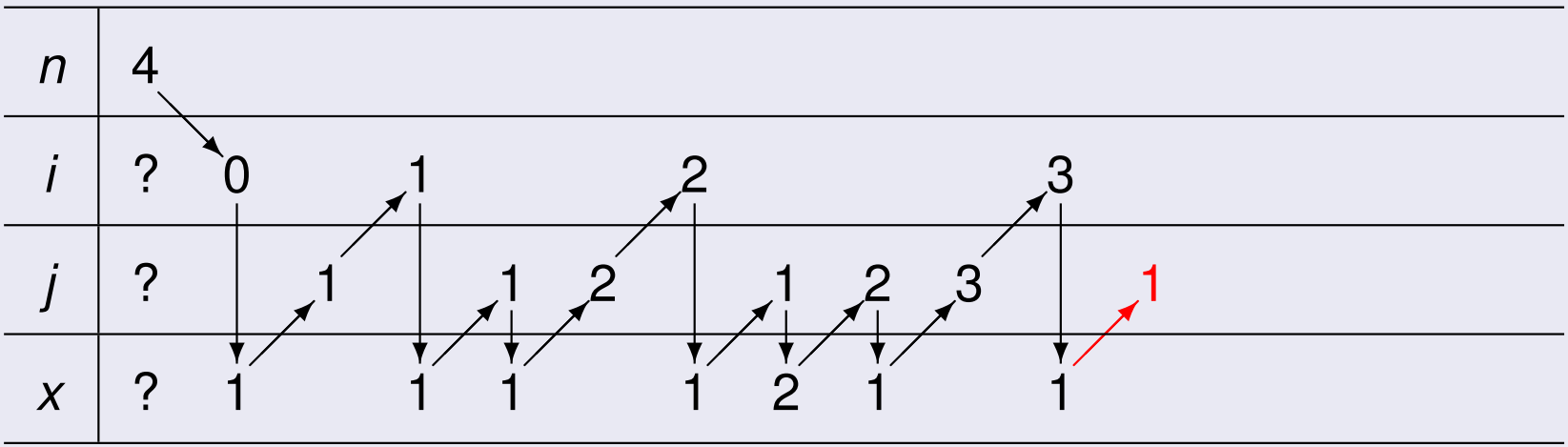
```
1
1 1
1 2 1
1
```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire " ", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

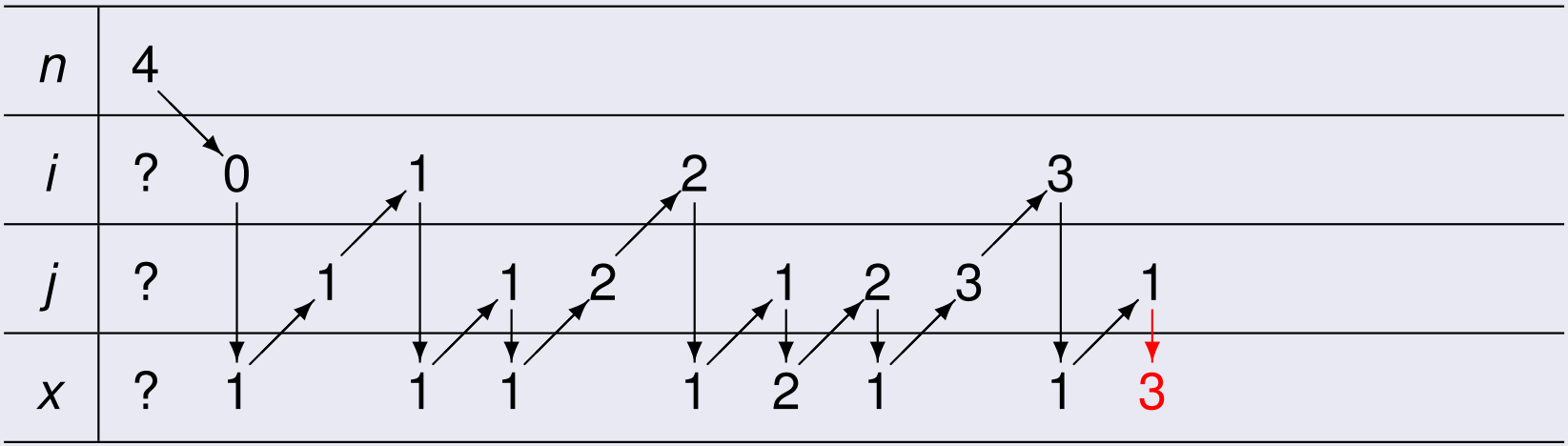
```
1
1 1
1 2 1
1
```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire " ", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

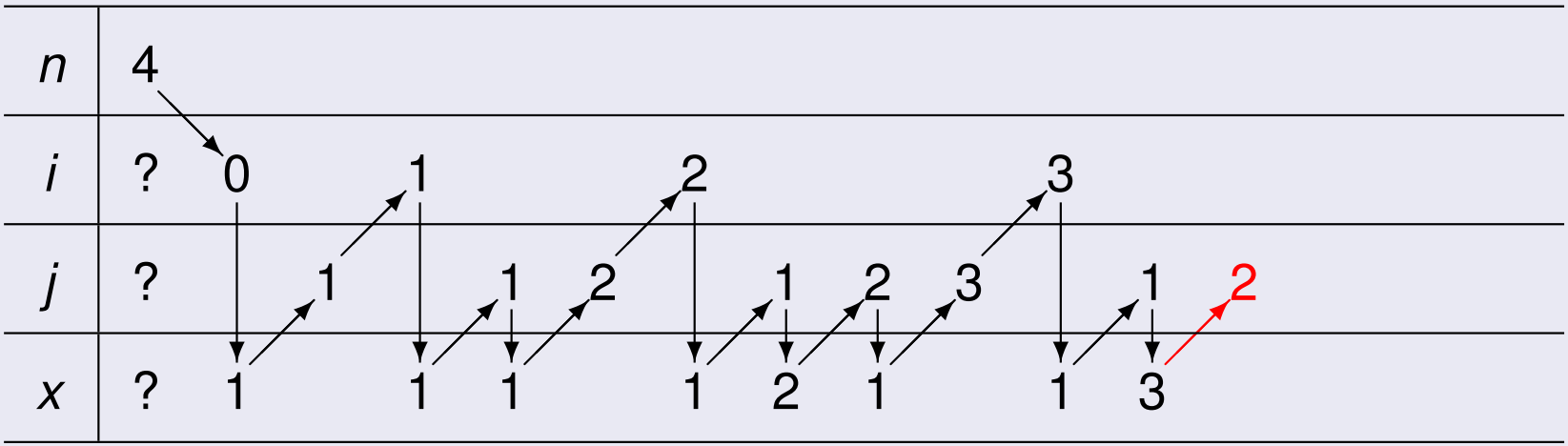
```
1
1 1
1 2 1
1 3
```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire " ", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

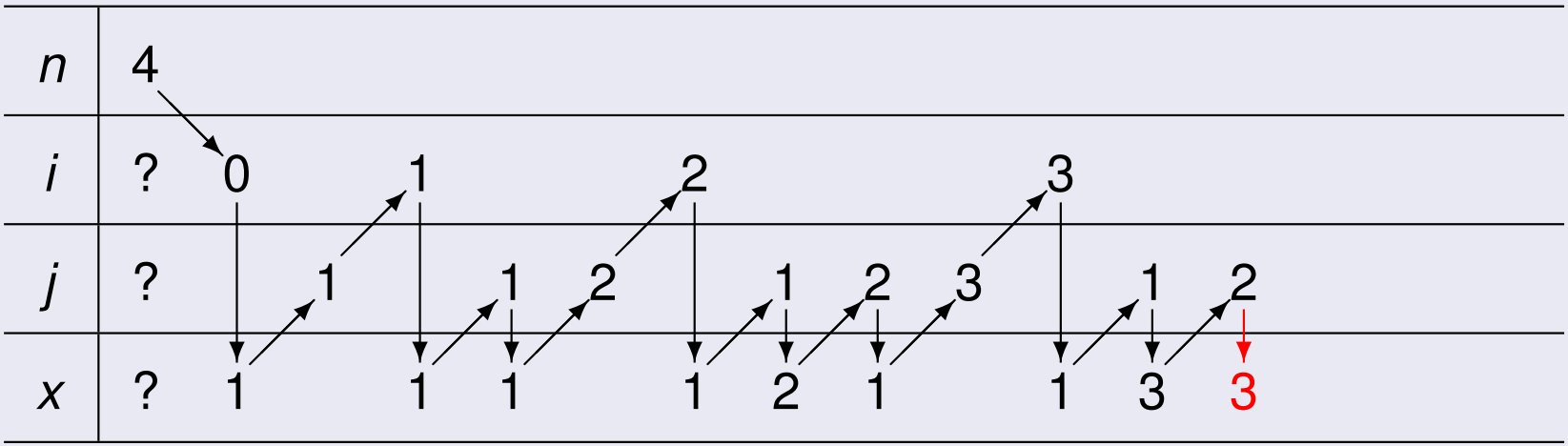
```
1
1 1
1 2 1
1 3
```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire " ", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

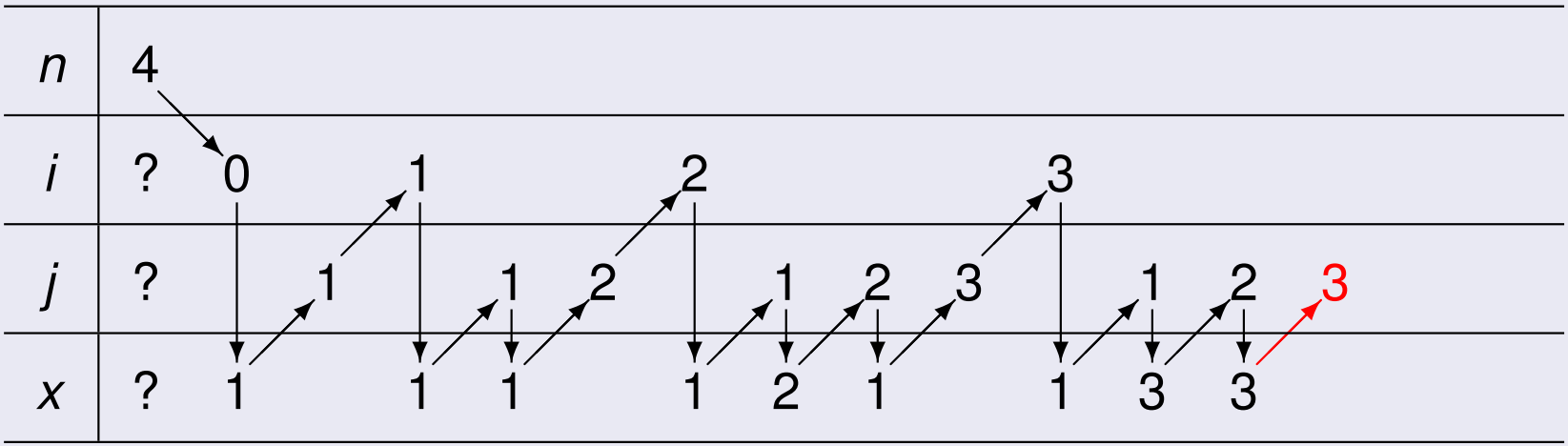
```
1
1 1
1 2 1
1 3 3 1
```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire " ", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

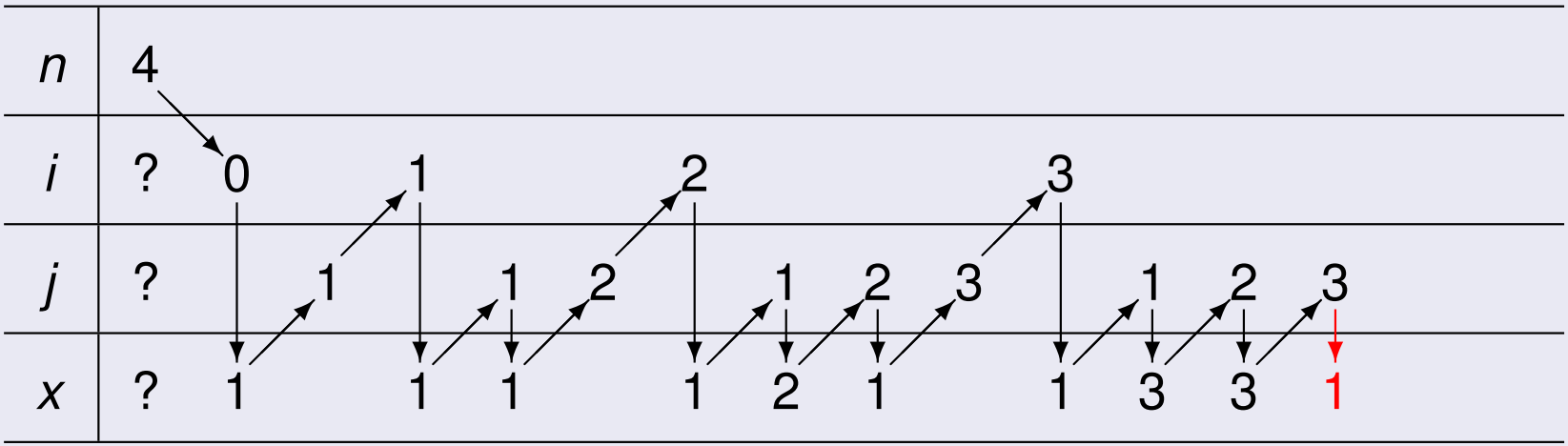
```
1
1 1
1 2 1
1 3 3
```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire " ", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

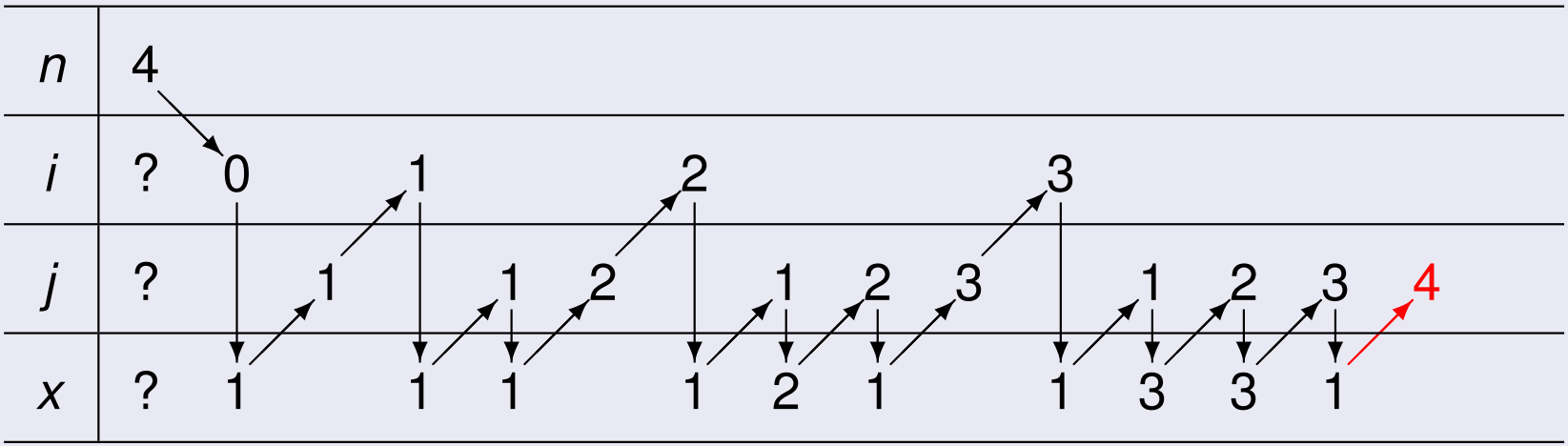
```
1
1 1
1 2 1
1 3 3 1
```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire " ", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

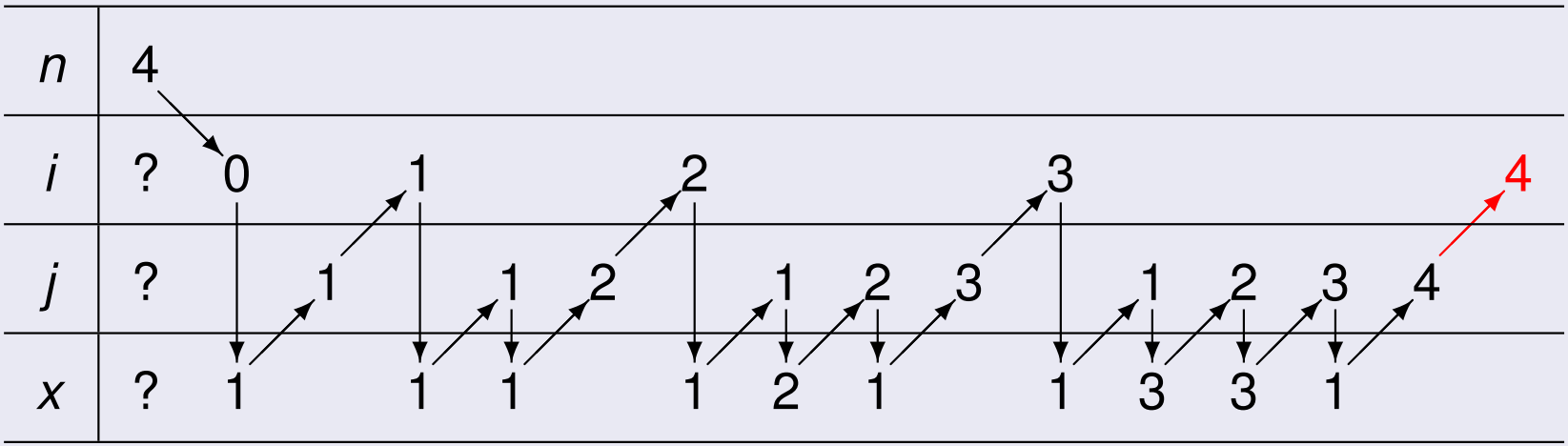
```
1
1 1
1 2 1
1 3 3 1
```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire " ", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

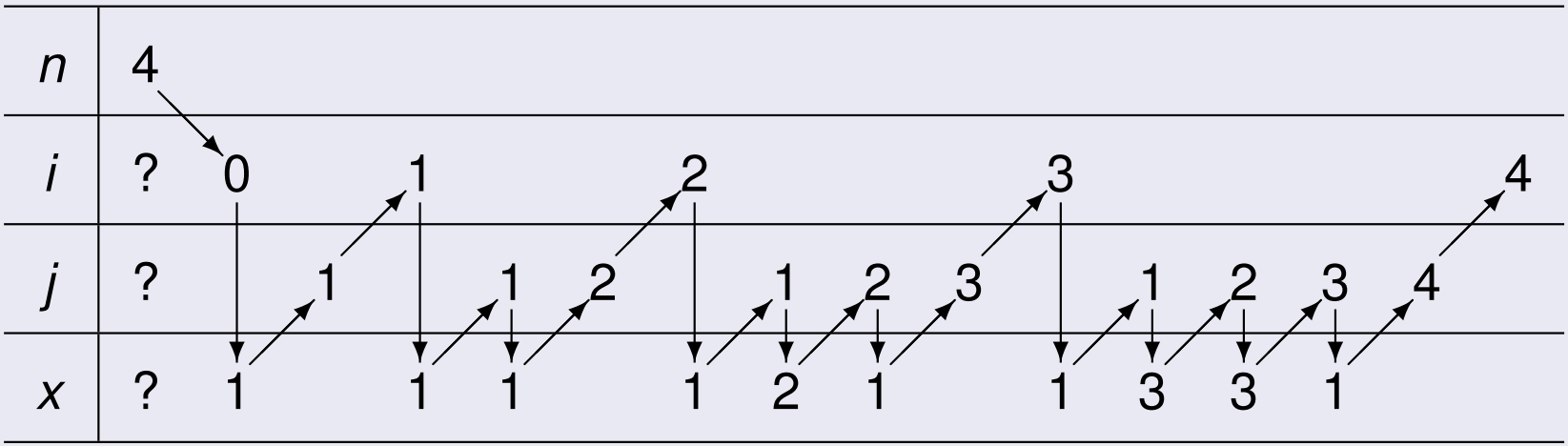
```
1
1 1
1 2 1
1 3 3 1
```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire " ", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

Variables : évolution des valeurs au cours du temps



Sorties

```
1
1 1
1 2 1
1 3 3 1
```

Algorithme

pour i **de** 0 **à** $n-1$ **faire**
  $x \leftarrow 1$
  écrire x
  pour j **de** 1 **à** i **faire**
    $x \leftarrow (x \times (i-j+1))/j$
    écrire "␣", x
  fpour
  écrire "\n"
fpour

Exemple : triangle de Pascal

```
Sub pascal()

Dim n As Integer
Dim i As Integer
Dim j As Integer
Dim x As Integer

Feuil1.Cells.Clear

Do
n = InputBox("Combien de lignes souhaitez-vous ?")
Loop Until n >= 1

If n = 1 Then
Feuil1.Cells(1, 1) = 1
End If

For i = 1 To n-1
Feuil1.Cells(1, 1) = 1
x = 1
  For j = 1 To i
  x = x * (i - j + 1) / j
  Feuil1.Cells(i + 1, 1) = 1
  Feuil1.Cells(i + 1, j + 1) = x
  Next j
Next i

End Sub
```

Plan

Les fonctions

- Morceaux d'algorithme réutilisables à volonté.
- Les fonctions permettent la décomposition des algorithmes en sous-problèmes (raffinements successifs).
- Prennent en entrée des paramètres.
- Restituent à l'algorithme appelant un ou plusieurs résultats.
- Définies par :
  - *Un 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

  - *Un corps :*
    - Algorithme de la fonction

En-tête de fonction

- Repéré par le mot clef « **fonction** ».

Exemple :

fonction nomFonction(mode *nomParam* : typeParam, . . .) : **ret** typeRet
Indique ce que fait la fonction et le rôle de ses paramètres.

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.

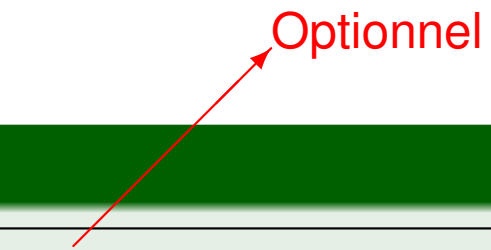
nomParam identifiant du paramètre dans la fonction

typeParam 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** ».

En-tête de fonction

- Repéré par le mot clef « **fonction** ».



Exemple :

fonction nomFonction(mode *nomParam* : typeParam, . . .) : **ret** typeRet
Indique ce que fait la fonction et le rôle de ses paramètres.

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.

nomParam identifiant du paramètre dans la fonction

typeParam 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 de la fonction

- Construit comme un algorithme classique :
  - Idée de l'algorithme
  - Lexique *local* des constantes
  - Lexique *local* 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 lexique des variables, dans le lexique des fonctions (liste des en-têtes).
- La *définition* des fonctions est placée après l'algorithme principal, dans la section définition des fonctions (corps des fonctions).

Retour de fonction

- Lorsque la fonction a un *retour*, sa dernière instruction exécutée doit être
  retourner valeurDeRetour

  Cela permet de renvoyer effectivement une valeur à l'algorithme appelant.
- Lorsque la fonction n'a pas de retour, elle peut arrêter son exécution avec l'instruction
  retourner

  Cette instruction est *implicite* à la fin de l'algorithme de la fonction.
- Les fonctions sans retour sont appelées *procédures*.

Variables, paramètres et appel de fonction

Variables locales et paramètres

- Les variables locales à une fonction ont *toujours* un rôle intermédiaire. Il n'est donc pas nécessaire de le préciser.
- Les paramètres de la fonction *ne doivent pas* être redéfinis dans le lexique local des variables de la fonction.
- 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*.

Illustration

Lexique des fonctions

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

Lexique des variables

x	(entier)	premier entier	DONNÉE
y	(entier)	second entier	DONNÉE
z	(entier)	somme des deux entiers	INTERMÉDIAIRE

Algorithme

$z \leftarrow \text{somme}(x, y)$

écrire z

Illustration

Lexique des fonctions

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

Lexique des variables

x	(entier)	premier entier	DONNÉE
y	(entier)	second entier	DONNÉE
z	(entier)	somme des deux entiers	INTERMÉDIAIRE

Algorithme

$z \leftarrow \text{somme}(x, y)$
écrire z

---

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

Lexique local des variables

c	(entier)	somme de a et b

Algorithme de somme

$c \leftarrow a + b$
retourner c

Illustration

Lexique des fonctions

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

Lexique des variables

x	(entier)	premier entier	DONNÉE
y	(entier)	second entier	DONNÉE
z	(entier)	somme des deux entiers	INTERMÉDIAIRE

Algorithme

$z \leftarrow \text{somme}(x, y)$

écrire z

---

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

a et *b* sont les paramètres formels

Lexique local des variables

c (entier) somme de *a* et *b*

Algorithme de somme

$c \leftarrow a + b$

retourner *c*

Illustration

Lexique des fonctions

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

Lexique des variables

x	(entier)	premier entier	DONNÉE
y	(entier)	second entier	DONNÉE
z	(entier)	somme des deux entiers	INTERMÉDIAIRE

Algorithme

$z \leftarrow$ somme(x, y) → *x* et *y* sont les paramètres effectifs

écrire *z*

---

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

a et *b* sont les paramètres formels

Lexique local des variables

c (entier) somme de *a* et *b*

Algorithme de somme

$c \leftarrow a + b$

retourner *c*

Exemple : algorithme principal

Calcule et affiche le volume d'un certain nombre de prismes à base triangulaire équilatérale. Toutes les informations sont entrées par l'utilisateur.

Lexique des fonctions

fonction volPrisme(**in** *côté* : réel, **in** *hauteur* : réel) : **ret** réel

Calcule le volume d'un prisme de côté et de hauteur donnés.

Lexique des variables

nbVol	(entier)	nombre de volumes à calculer	INTERMÉDIAIRE
i	(entier)	compteur de boucle	INTERMÉDIAIRE
hauteurPri	(réel)	hauteur du prisme	INTERMÉDIAIRE
côtéPri	(réel)	côté du triangle formant la base	INTERMÉDIAIRE

Algorithme

nbVol ← lire
pour *i* **de** 1 **à** *nbVol* **faire**
  côtéPri ← lire
  hauteurPri ← lire
  écrire "`Le volume du prisme est `", volPrisme(*côtéPri*, *hauteurPri*)
fpour

Exemple : algorithme de la fonction

fonction volPrisme(**in** *côté* : réel, **in** *hauteur* : réel) : **ret** réel

Calcule le volume d'un prisme à base triangulaire de côté et de hauteur donnés.

Lexique local des variables

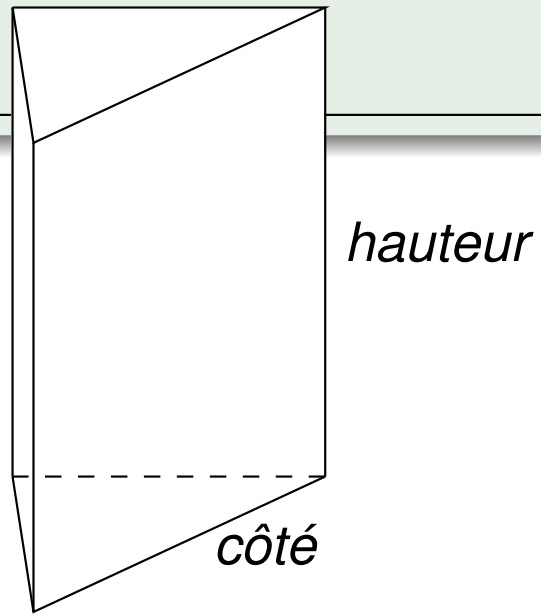
hauteurTri	(réel)	hauteur du triangle formant la base
surfTri	(réel)	surface du triangle formant la base

Algorithme de volPrisme

hauteurTri $\leftarrow$ *côté* $\times \sqrt{3}/2$

surfTri $\leftarrow$ *côté* $\times$ *hauteurTri*$/2$

retourner *surfTri* $\times$ *hauteur*



Déroulement de l'exemple

Variables

nbVol	?	*i*	?	*côtéPri*	?	*hauteurPri*	?

Algorithme principal

nbVol ← lire
pour *i* **de** 1 **à** *nbVol* **faire**
  côtéPri ← lire
  hauteurPri ← lire
  écrire "`Le volume du prisme est `", volPrisme(*côtéPri*, *hauteurPri*)
fpour

Entrées

Sorties

Déroulement de l'exemple

Variables

nbVol	2	*i*	?	*côtéPri*	?	*hauteurPri*	?

Algorithme principal

nbVol ← lire
pour *i* **de** 1 **à** *nbVol* **faire**
  côtéPri ← lire
  hauteurPri ← lire
  écrire "`Le volume du prisme est `", volPrisme(*côtéPri*, *hauteurPri*)
fpour

Entrées

2

Sorties

Déroulement de l'exemple

Variables

nbVol	2	*i*	1	*côtéPri*	?	*hauteurPri*	?

Algorithme principal

nbVol ← lire
pour *i* **de** 1 **à** *nbVol* **faire**
    côtéPri ← lire
    hauteurPri ← lire
    écrire "`Le volume du prisme est `", volPrisme(*côtéPri*, *hauteurPri*)
fpour

Entrées

2

Sorties

Déroulement de l'exemple

Variables

nbVol	2	*i*	1	*côtéPri*	10	*hauteurPri*	20

Algorithme principal

nbVol ← lire
pour *i* **de** 1 **à** *nbVol* **faire**
  côtéPri ← lire
  hauteurPri ← lire
  écrire "`Le volume du prisme est `", volPrisme(*côtéPri*, *hauteurPri*)
fpour

Entrées

```
2
10
20
```

Sorties

Déroulement de l'exemple

Variables

nbVol	2	*i*	1	*côtéPri*	10	*hauteurPri*	20

côté	10	*hauteur*	20	*retour de* volPrisme	?

Algorithme principal

nbVol ← lire
pour *i* **de** 1 **à** *nbVol* **faire**
  côtéPri ← lire
  hauteurPri ← lire
  écrire "`Le volume du prisme est `", volPrisme(*côtéPri*, *hauteurPri*)
fpour

Entrées

```
2
10
20
```

Sorties

Déroulement de l'exemple

Variables

nbVol	2	*i*	1	*côtéPri*	10	*hauteurPri*	20

côté	10	*hauteur*	20	*retour de* volPrisme	?
hauteurTri	?	*surfTri*	?		

Algorithme de volPrisme

hauteurTri $\leftarrow$ *côté* $\times \sqrt{3}/2$
surfTri $\leftarrow$ *côté* $\times$ *hauteurTri*$/2$
retourner *surfTri* $\times$ *hauteur*

Entrées

```
2
10
20
```

Sorties

Déroulement de l'exemple

Variables

nbVol	2	*i*	1	*côtéPri*	10	*hauteurPri*	20

côté	10	*hauteur*	20	*retour de* volPrisme	?
hauteurTri	8,660	*surfTri*	?		

Algorithme de volPrisme

hauteurTri $\leftarrow$ *côté* $\times \sqrt{3}/2$
surfTri $\leftarrow$ *côté* $\times$ *hauteurTri*$/2$
retourner *surfTri* $\times$ *hauteur*

Entrées

```
2
10
20
```

Sorties

Déroulement de l'exemple

Variables

nbVol	2	*i*	1	*côtéPri*	10	*hauteurPri*	20

côté	10	*hauteur*	20	*retour de* volPrisme	?
hauteurTri	8,660	*surfTri*	43,300		

Algorithme de volPrisme

hauteurTri $\leftarrow$ *côté* $\times \sqrt{3}/2$

surfTri $\leftarrow$ *côté* $\times$ *hauteurTri*$/2$

retourner *surfTri* $\times$ *hauteur*

Entrées

```
2
10
20
```

Sorties

Déroulement de l'exemple

Variables

nbVol	2	*i*	1	*côtéPri*	10	*hauteurPri*	20

côté	10	*hauteur*	20	*retour de* volPrisme	866,000
hauteurTri	8,660	*surfTri*	43,300		

Algorithme de volPrisme

hauteurTri $\leftarrow$ *côté* $\times \sqrt{3}/2$
surfTri $\leftarrow$ *côté* $\times$ *hauteurTri*$/2$
retourner *surfTri* $\times$ *hauteur*

Entrées

```
2
10
20
```

Sorties

Déroulement de l'exemple

Variables

nbVol	2	*i*	1	*côtéPri*	10	*hauteurPri*	20

retour de volPrisme : 866,000

Algorithme principal

nbVol ← lire
pour *i* **de** 1 **à** *nbVol* **faire**
  côtéPri ← lire
  hauteurPri ← lire
  écrire "Le volume du prisme est ", volPrisme(*côtéPri*, *hauteurPri*)
fpour

Entrées

```
2
10
20
```

Sorties

```
Le volume du prisme est 866,000
```

Déroulement de l'exemple

Variables

nbVol	2	*i*	2	*côtéPri*	10	*hauteurPri*	20

Algorithme principal

nbVol ← lire
pour *i* **de** 1 **à** *nbVol* **faire**
    côtéPri ← lire
    hauteurPri ← lire
    écrire "`Le volume du prisme est `", volPrisme(*côtéPri*, *hauteurPri*)
fpour

Entrées

```
2
10
20
```

Sorties

```
Le volume du prisme est 866,000
```

Déroulement de l'exemple

Variables

nbVol	2	*i*	2	*côtéPri*	30	*hauteurPri*	40

Algorithme principal

nbVol ← lire
pour *i* **de** 1 **à** *nbVol* **faire**
  côtéPri ← lire
  hauteurPri ← lire
  écrire "`Le volume du prisme est `", volPrisme(*côtéPri*, *hauteurPri*)
fpour

Entrées

```
...
30
40
```

Sorties

```
Le volume du prisme est 866,000
```

Déroulement de l'exemple

Variables

nbVol	2	*i*	2	*côtéPri*	30	*hauteurPri*	40

côté	30	*hauteur*	40	*retour de* volPrisme	?

Algorithme principal

nbVol ← lire
pour *i* **de** 1 **à** *nbVol* **faire**
  côtéPri ← lire
  hauteurPri ← lire
  écrire "`Le volume du prisme est `", volPrisme(*côtéPri*, *hauteurPri*)
fpour

Entrées

```
...
30
40
```

Sorties

```
Le volume du prisme est 866,000
```

Déroulement de l'exemple

Variables

nbVol	2	*i*	2	*côtéPri*	30	*hauteurPri*	40

côté	30	*hauteur*	40	*retour de* volPrisme	?
hauteurTri	?	*surfTri*	?		

Algorithme de volPrisme

hauteurTri $\leftarrow$ *côté* $\times \sqrt{3}/2$
surfTri $\leftarrow$ *côté* $\times$ *hauteurTri*$/2$
retourner *surfTri* $\times$ *hauteur*

Entrées

```
...
30
40
```

Sorties

```
Le volume du prisme est 866,000
```

Déroulement de l'exemple

Variables

nbVol	2	*i*	2	*côtéPri*	30	*hauteurPri*	40

côté	30	*hauteur*	40	*retour de* volPrisme	?
hauteurTri	25,981	*surfTri*	?		

Algorithme de volPrisme

hauteurTri $\leftarrow$ *côté* $\times \sqrt{3}/2$
surfTri $\leftarrow$ *côté* $\times$ *hauteurTri*$/2$
retourner *surfTri* $\times$ *hauteur*

Entrées

```
...
30
40
```

Sorties

```
Le volume du prisme est 866,000
```

Déroulement de l'exemple

Variables

nbVol	2	*i*	2	*côtéPri*	30	*hauteurPri*	40

côté	30	*hauteur*	40	*retour de* volPrisme	?

hauteurTri	25,981	*surfTri*	389,715

Algorithme de volPrisme

hauteurTri $\leftarrow$ *côté* $\times \sqrt{3}/2$

surfTri $\leftarrow$ *côté* $\times$ *hauteurTri*$/2$

retourner *surfTri* $\times$ *hauteur*

Entrées

```
...
30
40
```

Sorties

```
Le volume du prisme est 866,000
```

Déroulement de l'exemple

Variables

nbVol	2	*i*	2	*côtéPri*	30	*hauteurPri*	40

côté	30	*hauteur*	40	*retour de* volPrisme	15 588,600

hauteurTri	25,981	*surfTri*	389,715

Algorithme de volPrisme

hauteurTri $\leftarrow$ *côté* $\times \sqrt{3}/2$

surfTri $\leftarrow$ *côté* $\times$ *hauteurTri*$/2$

retourner *surfTri* $\times$ *hauteur*

Entrées

```
...
30
40
```

Sorties

```
Le volume du prisme est 866,000
```

Déroulement de l'exemple

Variables

nbVol	*i*	*côtéPri*	*hauteurPri*
2	2	30	40

retour de volPrisme : 15 588,600

Algorithme principal

nbVol ← lire
pour *i* **de** 1 **à** *nbVol* **faire**
  côtéPri ← lire
  hauteurPri ← lire
  écrire "Le volume du prisme est ", volPrisme(*côtéPri*, *hauteurPri*)
fpour

Entrées

```
...
30
40
```

Sorties

```
Le volume du prisme est 866,000
Le volume du prisme est 15 588,600
```

Déroulement de l'exemple

Variables

nbVol	2	*i*	3	*côtéPri*	30	*hauteurPri*	40

Algorithme principal

nbVol ← lire
pour *i* **de** 1 **à** *nbVol* **faire**
    côtéPri ← lire
    hauteurPri ← lire
    écrire "`Le volume du prisme est `", volPrisme(*côtéPri*, *hauteurPri*)
fpour

Entrées

```
...
30
40
```

Sorties

```
Le volume du prisme est 866,000
Le volume du prisme est 15 588,600
```

Déroulement de l'exemple

Variables

Algorithme principal

nbVol ← lire
pour *i* **de** 1 **à** *nbVol* **faire**
  côtéPri ← lire
  hauteurPri ← lire
  écrire "`Le volume du prisme est `", volPrisme(*côtéPri*, *hauteurPri*)
fpour

Entrées

```
...
30
40
```

Sorties

```
Le volume du prisme est 866,000
Le volume du prisme est 15 588,600
```

Fonction sans retour

Exemple : algorithme principal

Dessine trois carrés d'étoiles de côté 1, 3 et 5.

Lexique des fonctions

fonction dessineCarré(**in** *côté* : entier) : **vide**

Dessine un carré d'étoiles de côté donné.

Lexique des variables

i	(entier)	compteur de carrés	INTERMÉDIAIRE

Algorithme

pour *i* **de** 1 **à** 5 **par pas de** 2 **faire**
  dessineCarré(*i*)
fpour

Fonction sans retour

Exemple : algorithme de la fonction

fonction dessineCarré(**in** *côté* : entier) : **vide**

Dessine un carré d'étoiles de côté donné.

Lexique local des variables

i	(entier)	compteur de lignes
j	(entier)	compteur de colonnes

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	?

Sorties

Algorithme principal

pour i **de** 1 **à** 5 **par pas de** 2 **faire**

  dessineCarré(i)

fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	1

Sorties

Algorithme principal

pour i **de** 1 **à** 5 **par pas de** 2 **faire**
  dessineCarré(i)
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	1
côté	1

Sorties

Algorithme principal

pour *i* **de** 1 **à** 5 **par pas de** 2 **faire**

  dessineCarré(*i*)

fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	1

côté	1		
i	?	*j*	?

Sorties

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	1

côté	1		
i	1	*j*	?

Sorties

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	1

côté	1		
i	1	*j*	2

Sorties

*

Algorithme de dessineCarré

```
pour i de 1 à côté faire
    pour j de 1 à côté faire
        écrire "*"
    fpour
    écrire "\n"
fpour
```

Déroulement de l'exemple : fonction sans retour

Variables

i	1

côté	1		
i	2	*j*	2

Sorties

*

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	1

côté	1		
i	2	*j*	2

Sorties

*

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour
// Instruction **retourner** implicite.

Déroulement de l'exemple : fonction sans retour

Variables

i	1

Sorties

*

Algorithme principal

pour *i* **de** 1 **à** 5 **par pas de** 2 **faire**

  dessineCarré(*i*)

fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	3

Sorties

*

Algorithme principal

pour i **de** 1 **à** 5 **par pas de** 2 **faire**
  dessineCarré(i)
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	3
côté	3

Sorties

*

Algorithme principal

pour *i* **de** 1 **à** 5 **par pas de** 2 **faire**

  dessineCarré(*i*)

fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	3

côté	3		
i	?	*j*	?

Sorties

*

Algorithme de dessineCarré

```
pour i de 1 à côté faire
    pour j de 1 à côté faire
        écrire "*"
    fpour
    écrire "\n"
fpour
```

Déroulement de l'exemple : fonction sans retour

Variables

i	3

côté	3		
i	1	*j*	?

Sorties

*

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	3

côté	3		
i	1	*j*	4

Sorties

```
*
***
```

Algorithme de dessineCarré

```
pour i de 1 à côté faire
    pour j de 1 à côté faire
        écrire "*"
    fpour
    écrire "\n"
fpour
```

Déroulement de l'exemple : fonction sans retour

Variables

i	3

côté	3		
i	2	*j*	4

Sorties

```
*
***
```

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	3

côté	3		
i	2	*j*	4

Sorties

```
*
***
***
```

Algorithme de dessineCarré

```
pour i de 1 à côté faire
    pour j de 1 à côté faire
        écrire "*"
    fpour
    écrire "\n"
fpour
```

Déroulement de l'exemple : fonction sans retour

Variables

i	3

côté	3		
i	3	*j*	4

Sorties

```
*
***
***
```

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	3

côté	3		
i	3	*j*	4

Sorties

```
*
***
***
***
```

Algorithme de dessineCarré

```
pour i de 1 à côté faire
    pour j de 1 à côté faire
        écrire "*"
    fpour
    écrire "\n"
fpour
```

Déroulement de l'exemple : fonction sans retour

Variables

i	3

côté	3		
i	4	*j*	4

Sorties

```
*
***
***
***
```

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	3

côté	3		
i	4	*j*	4

Sorties

```
*
***
***
***
```

Algorithme de dessineCarré

```
pour i de 1 à côté faire
    pour j de 1 à côté faire
        écrire "*"
    fpour
    écrire "\n"
fpour
// Instruction retourner implicite.
```

Déroulement de l'exemple : fonction sans retour

Variables

i	3

Sorties

```
*
***
***
***
```

Algorithme principal

pour i **de** 1 **à** 5 **par pas de** 2 **faire**

  dessineCarré(i)

fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	5

Sorties

```
*
***
***
***
```

Algorithme principal

pour i **de** 1 **à** 5 **par pas de** 2 **faire**
  dessineCarré(i)
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	5
côté	5

Sorties

```
*
***
***
***
```

Algorithme principal

pour *i* **de** 1 **à** 5 **par pas de** 2 **faire**

  dessineCarré(*i*)

fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	5

côté	5		
i	?	*j*	?

Sorties

```
*
***
***
***
```

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	5

côté	5		
i	1	*j*	?

Sorties

```
*
***
***
***
```

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	5

côté	5		
i	1	*j*	6

Sorties

```
*
***
***
***
*****
```

Algorithme de dessineCarré

```
pour i de 1 à côté faire
    pour j de 1 à côté faire
        écrire "*"
    fpour
    écrire "\n"
fpour
```

Déroulement de l'exemple : fonction sans retour

Variables

i	5

côté	5		
i	2	*j*	6

Sorties

```
*
***
***
***
*****
```

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	5

côté	5		
i	2	*j*	6

Sorties

```
*
***
***
***
*****
*****
```

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	5

côté	5		
i	3	*j*	6

Sorties

```
*
***
***
***
*****
*****
```

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	5

côté	5		
i	3	*j*	6

Sorties

```
*
***
***
***
*****
*****
*****
```

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	5

côté	5		
i	4	*j*	6

Sorties

```
*
***
***
***
*****
*****
*****
```

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	5

côté	5		
i	4	*j*	6

Sorties

```
*
***
***
***
*****
*****
*****
*****
```

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	5

côté	5		
i	5	*j*	6

Sorties

```
*
***
***
***
*****
*****
*****
*****
```

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	5

côté	5		
i	5	*j*	6

Sorties

```
*
***
***
***
*****
*****
*****
*****
*****
```

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	5

côté	5		
i	6	*j*	6

Sorties

```
*
***
***
***
*****
*****
*****
*****
*****
```

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	5

côté	5		
i	6	*j*	6

Sorties

```
*
***
***
***
*****
*****
*****
*****
*****
```

Algorithme de dessineCarré

pour *i* **de** 1 **à** *côté* **faire**
  pour *j* **de** 1 **à** *côté* **faire**
    écrire "*"
  fpour
  écrire "\n"
fpour
// Instruction **retourner** implicite.

Déroulement de l'exemple : fonction sans retour

Variables

i	5

Sorties

```
*
***
***
***
*****
*****
*****
*****
*****
```

Algorithme principal

pour i **de** 1 **à** 5 **par pas de** 2 **faire**

  dessineCarré(i)

fpour

Déroulement de l'exemple : fonction sans retour

Variables

i	7

Sorties

```
*
***
***
***
*****
*****
*****
*****
*****
```

Algorithme principal

pour i **de** 1 **à** 5 **par pas de** 2 **faire**
  dessineCarré(i)
fpour

Déroulement de l'exemple : fonction sans retour

Variables

Sorties

```
*
***
***
***
*****
*****
*****
*****
*****
```

Algorithme principal

pour *i* **de** 1 **à** 5 **par pas de** 2 **faire**

  dessineCarré(*i*)

fpour

Paramètres **out**

- Le mécanisme de retour de fonction ne permet de retourner qu'un seul résultat.
- Pour retourner plusieurs résultats, on peut utiliser les paramètres de mode **out**.
- Les résultats sont alors retournés *via* les variables passées en paramètres.

Exemple : conversion heures/minutes/secondes

fonction hms(**in** *durée* : entier,
out *heures* : entier, **out** *minutes* : entier, **out** *secondes* : entier) : **vide**

Convertit une durée exprimée en secondes en heures, minutes, secondes.

Algorithme de hms

$heures \leftarrow durée/3600$
$minutes \leftarrow (durée \bmod 3600)/60$
$secondes \leftarrow durée \bmod 60$

Paramètres **out**

Exemple : algorithme principal

Lexique des fonctions

fonction hms(**in** *durée* : entier,
out *heures* : entier, **out** *minutes* : entier, **out** *secondes* : entier) : **vide**

Convertit une durée exprimée en secondes en heures, minutes, secondes.

Lexique des variables

d	(entier)	une durée en secondes	INTERMÉDIAIRE
h	(entier)	le nombre d'heures	INTERMÉDIAIRE
m	(entier)	le nombre de minutes	INTERMÉDIAIRE
s	(entier)	le nombre de secondes	INTERMÉDIAIRE

Algorithme

$d \leftarrow 12345$
hms(d, h, m, s)
écrire d, "`secondes =`", h, "`heures`", m, "`minutes`", s, "`secondes`"

Déroulement de l'exemple

Variables

d	h	m	s
?	?	?	?

---

fonction hms(**in** *durée* : entier,
out *heures* : entier, **out** *minutes* : entier, **out** *secondes* : entier) : **vide**

Algorithme principal

$d \leftarrow 12345$
hms(d, h, m, s)
écrire d, "`secondes =`", h, "`heures`", m, "`minutes`", s, "`secondes`"

Sorties

Déroulement de l'exemple

Variables

d	h	m	s
12345	?	?	?

---

fonction hms(**in** *durée* : entier,
out *heures* : entier, **out** *minutes* : entier, **out** *secondes* : entier) : **vide**

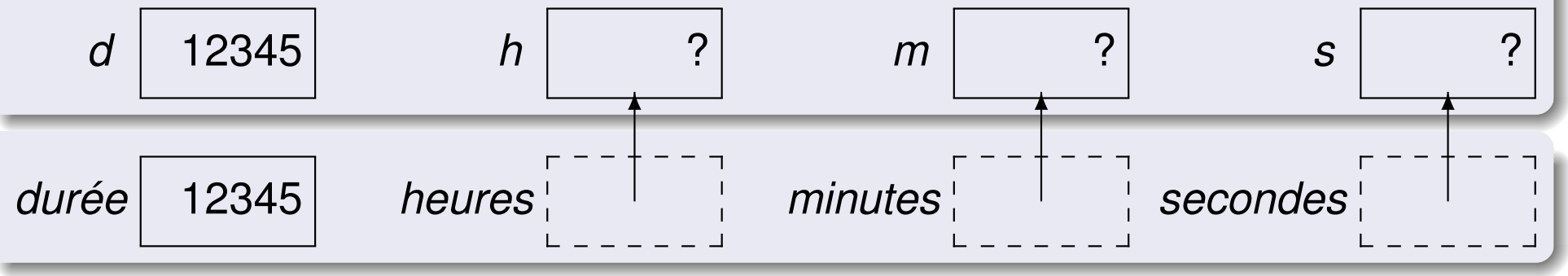
Algorithme principal

$d \leftarrow 12345$
hms(d, h, m, s)
écrire d, "`secondes =`", h, "`heures`", m, "`minutes`", s, "`secondes`"

Sorties

Déroulement de l'exemple

Variables



---

fonction hms(**in** *durée* : entier,
out *heures* : entier, **out** *minutes* : entier, **out** *secondes* : entier) : **vide**

Algorithme principal

$d \leftarrow 12345$

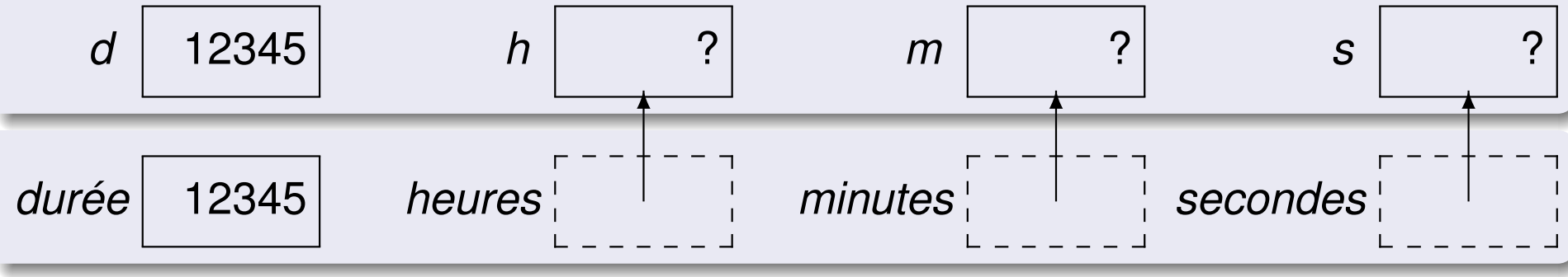
hms(d, h, m, s)

écrire d, "`secondes =`", h, "`heures`", m, "`minutes`", s, "`secondes`"

Sorties

Déroulement de l'exemple

Variables



---

fonction hms(**in** *durée* : entier,
out *heures* : entier, **out** *minutes* : entier, **out** *secondes* : entier) : **vide**

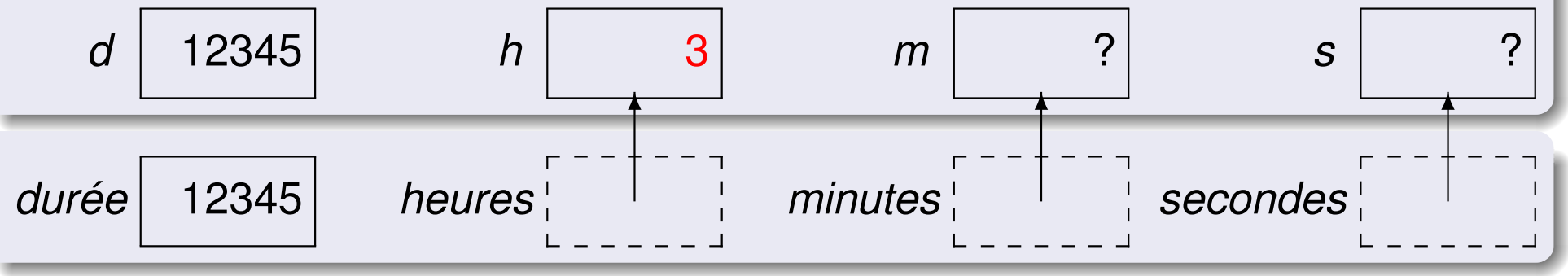
Algorithme de hms

$heures \leftarrow durée/3600$
$minutes \leftarrow (durée \bmod 3600)/60$
$secondes \leftarrow durée \bmod 60$

Sorties

Déroulement de l'exemple

Variables



---

fonction hms(**in** *durée* : entier,
out *heures* : entier, **out** *minutes* : entier, **out** *secondes* : entier) : **vide**

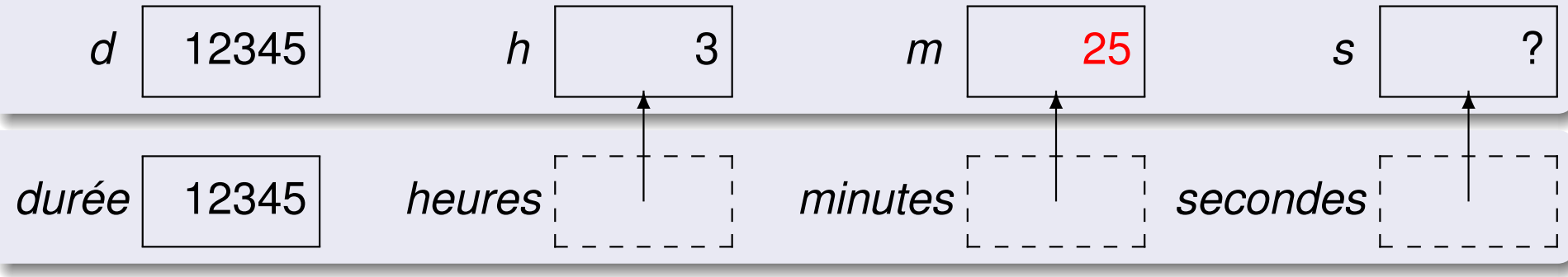
Algorithme de hms

heures $\leftarrow$ *durée*/3600
minutes $\leftarrow$ (*durée* mod 3600)/60
secondes $\leftarrow$ *durée* mod 60

Sorties

Déroulement de l'exemple

Variables



---

fonction hms(**in** *durée* : entier,
out *heures* : entier, **out** *minutes* : entier, **out** *secondes* : entier) : **vide**

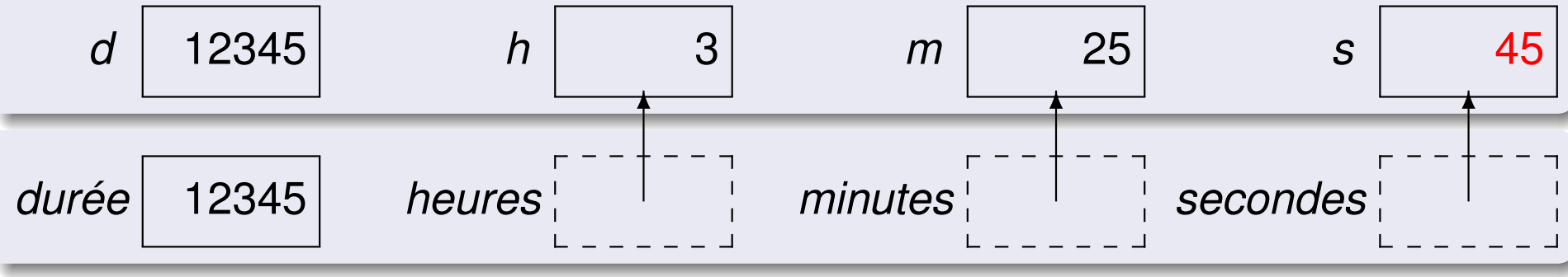
Algorithme de hms

heures $\leftarrow$ *durée*/3600
minutes $\leftarrow$ (*durée* mod 3600)/60
secondes $\leftarrow$ *durée* mod 60

Sorties

Déroulement de l'exemple

Variables



---

fonction hms(**in** *durée* : entier,
out *heures* : entier, **out** *minutes* : entier, **out** *secondes* : entier) : **vide**

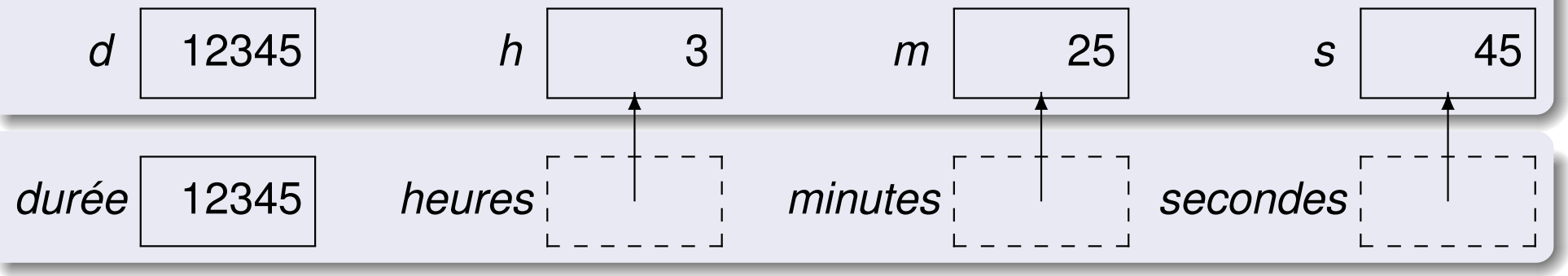
Algorithme de hms

$heures \leftarrow durée/3600$
$minutes \leftarrow (durée \bmod 3600)/60$
$secondes \leftarrow durée \bmod 60$

Sorties

Déroulement de l'exemple

Variables



---

fonction hms(**in** *durée* : entier,
out *heures* : entier, **out** *minutes* : entier, **out** *secondes* : entier) : **vide**

Algorithme de hms

$heures \leftarrow durée/3600$
$minutes \leftarrow (durée \bmod 3600)/60$
$secondes \leftarrow durée \bmod 60$

Sorties

Déroulement de l'exemple

Variables

d	12345	h	3	m	25	s	45

---

fonction hms(**in** *durée* : entier,
out *heures* : entier, **out** *minutes* : entier, **out** *secondes* : entier) : **vide**

Algorithme principal

$d \leftarrow 12345$
hms(d, h, m, s)
écrire d, "`secondes =`", h, "`heures`", m, "`minutes`", s, "`secondes`"

Sorties

Déroulement de l'exemple

Variables

d	12345	h	3	m	25	s	45

---

fonction hms(**in** *durée* : entier,
out *heures* : entier, **out** *minutes* : entier, **out** *secondes* : entier) : **vide**

Algorithme principal

$d \leftarrow 12345$
hms(d, h, m, s)
écrire d,"`secondes =`", h,"`heures`", m,"`minutes`", s,"`secondes`"

Sorties

```
12345 secondes = 3 heures 25 minutes 45 secondes
```

Déroulement de l'exemple

Variables

---

fonction hms(**in** *durée* : entier,
out *heures* : entier, **out** *minutes* : entier, **out** *secondes* : entier) : **vide**

Algorithme principal

$d \leftarrow 12345$
hms(d, h, m, s)
écrire d, "`secondes =`", h, "`heures`", m, "`minutes`", s, "`secondes`"

Sorties

```
12345 secondes = 3 heures 25 minutes 45 secondes
```

Paramètres **in-out**

- Lorsque les paramètres fournissent des données à la fonction, et servent également à retourner des résultats, il sont de mode **in-out**.
- Il faut, dans ce cas aussi, fournir des variables pour les paramètres.

Exemple : échange de deux valeurs

fonction échanger(**in-out** *a* : entier, **in-out** *b* : entier) : **vide**

Échange les valeurs des deux paramètres.

Lexique local des variables

c (entier) variable d'échange

Algorithme de échanger

$c \leftarrow a$
$a \leftarrow b$
$b \leftarrow c$

Paramètres **in-out**

Exemple : algorithme principal

Lexique des fonctions

fonction échanger(**in-out** *a* : entier, **in-out** *b* : entier) : **vide**

Échange les valeurs des deux paramètres.

Lexique des variables

x	(entier)	une première variable	INTERMÉDIAIRE
y	(entier)	une deuxième variable	INTERMÉDIAIRE

Algorithme

$x \leftarrow 123$
$y \leftarrow 456$
écrire “`avant : x =`”, x, “`y =`”, y
échanger(x, y)
écrire “`après : x =`”, x, “`y =`”, y

Déroulement de l'exemple

Variables

x ? y ?

Algorithme principal

$x \leftarrow 123$
$y \leftarrow 456$
écrire "`avant : x =`", x, "`et y =`", y
échanger(x, y)
écrire "`après : x =`", x, "`et y =`", y

Sorties

Déroulement de l'exemple

Variables

x	123	y	456

Algorithme principal

$x \leftarrow 123$
$y \leftarrow 456$
écrire "`avant : x =`", x, "`et y =`", y
échanger(x, y)
écrire "`après : x =`", x, "`et y =`", y

Sorties

Déroulement de l'exemple

Variables

x	123	y	456

Algorithme principal

$x \leftarrow 123$
$y \leftarrow 456$
écrire "`avant : x =`", x, "`et y =`", y
échanger(x, y)
écrire "`après : x =`", x, "`et y =`", y

Sorties

```
avant : x = 123 et y = 456
```

Déroulement de l'exemple

Variables

Algorithme principal

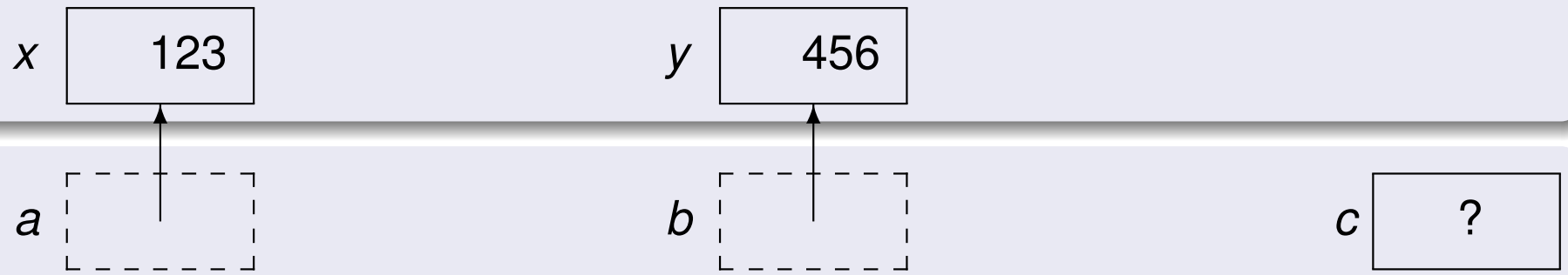
$x \leftarrow 123$
$y \leftarrow 456$
écrire "`avant : x =`", x, "`et y =`", y
échanger(x, y)
écrire "`après : x =`", x, "`et y =`", y

Sorties

```
avant : x = 123 et y = 456
```

Déroulement de l'exemple

Variables



Algorithme de échanger

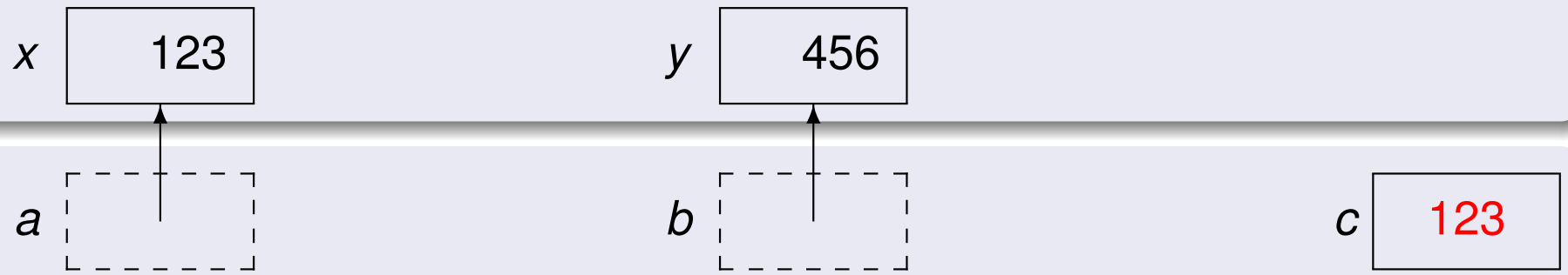
$c \leftarrow a$
$a \leftarrow b$
$b \leftarrow c$

Sorties

```
avant : x = 123 et y = 456
```

Déroulement de l'exemple

Variables



Algorithme de échanger

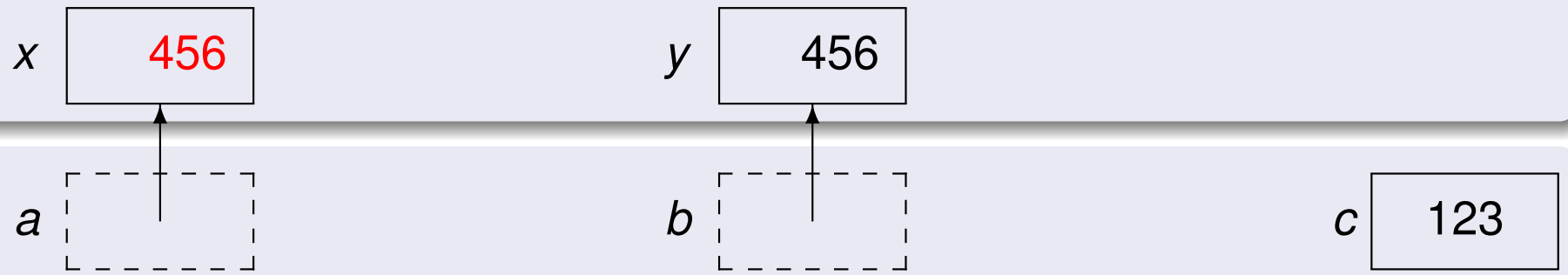
$c \leftarrow a$
$a \leftarrow b$
$b \leftarrow c$

Sorties

```
avant : x = 123 et y = 456
```

Déroulement de l'exemple

Variables



Algorithme de échanger

$c \leftarrow a$

$a \leftarrow b$

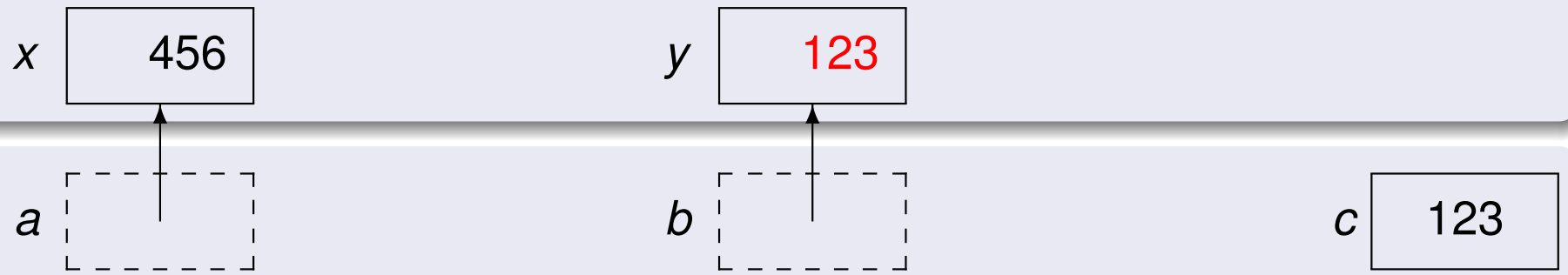
$b \leftarrow c$

Sorties

```
avant : x = 123 et y = 456
```

Déroulement de l'exemple

Variables



Algorithme de échanger

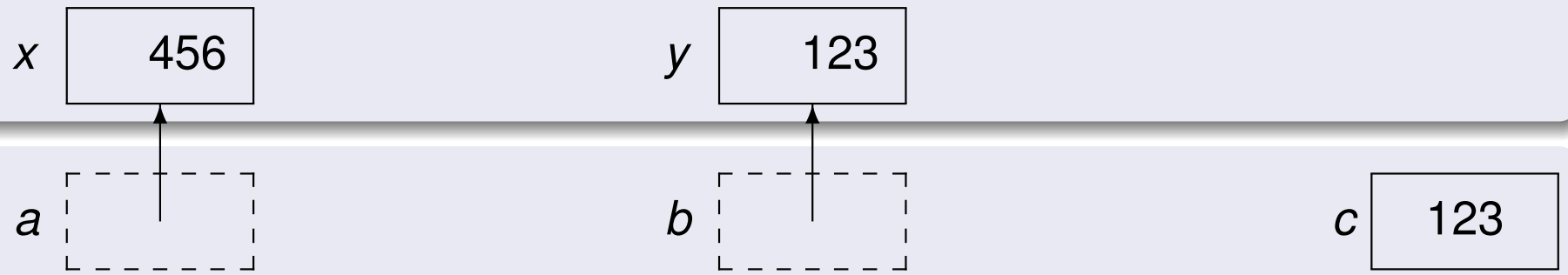
$c \leftarrow a$
$a \leftarrow b$
$b \leftarrow c$

Sorties

```
avant : x = 123 et y = 456
```

Déroulement de l'exemple

Variables



Algorithme de échanger

$c \leftarrow a$
$a \leftarrow b$
$b \leftarrow c$

Sorties

```
avant : x = 123 et y = 456
```

Déroulement de l'exemple

Variables

x	456
y	123

Algorithme principal

$x \leftarrow 123$
$y \leftarrow 456$
écrire "`avant : x =`", x, "`et y =`", y
échanger(x, y)
écrire "`après : x =`", x, "`et y =`", y

Sorties

```
avant : x = 123 et y = 456
```

Déroulement de l'exemple

Variables

x	456
y	123

Algorithme principal

$x \leftarrow 123$
$y \leftarrow 456$
écrire "`avant : x =`", x, "`et y =`", y
échanger(x, y)
écrire "`après : x =`", x, "`et y =`", y

Sorties

```
avant : x = 123 et y = 456
après : x = 456 et y = 123
```

Déroulement de l'exemple

Variables

Algorithme principal

$x \leftarrow 123$
$y \leftarrow 456$
écrire "`avant : x =`", x, "`et y =`", y
échanger(x, y)
écrire "`après : x =`", x, "`et y =`", y

Sorties

```
avant : x = 123 et y = 456
après : x = 456 et y = 123
```

Plan

Fonctions récursives

- Construire la solution d'un problème en utilisant la solution du *même* problème dans un contexte *différent* (plus simple).
- La suite des contextes doit tendre vers une solution directe (*cas terminal*).
- Les fonctions récursives sont adaptées à une certaine classe de problèmes.
- Exemple : la factorielle

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \times (n-1)! & \text{sinon.} \end{cases}$$

Exemple

Exemple : factorielle

fonction factorielle(**in** n : entier) : **ret** entier

Calcule la factorielle de n, positif ou nul.

Lexique local des variables

résultat (entier) résultat de la fonction

Algorithme de factorielle

si $n = 0$ **alors**
  $\mid$ *résultat* $\leftarrow 1$
sinon
  $\mid$ *résultat* $\leftarrow n \times \text{factorielle}(n-1)$
fsi
retourner *résultat*

Exemple d'exécution

- factorielle$(5) \Rightarrow 5 \times$ factorielle(4)

Exemple d'exécution

- factorielle$(5) \Rightarrow 5 \times$ factorielle(4)

  - factorielle$(4) \Rightarrow 4 \times$ factorielle(3)

Exemple d'exécution

- factorielle$(5) \Rightarrow 5 \times$ factorielle(4)
  - factorielle$(4) \Rightarrow 4 \times$ factorielle(3)
    - factorielle$(3) \Rightarrow 3 \times$ factorielle(2)

Exemple d'exécution

- factorielle(5) $\Rightarrow 5 \times$ factorielle(4)
  - factorielle(4) $\Rightarrow 4 \times$ factorielle(3)
    - factorielle(3) $\Rightarrow 3 \times$ factorielle(2)
      - factorielle(2) $\Rightarrow 2 \times$ factorielle(1)

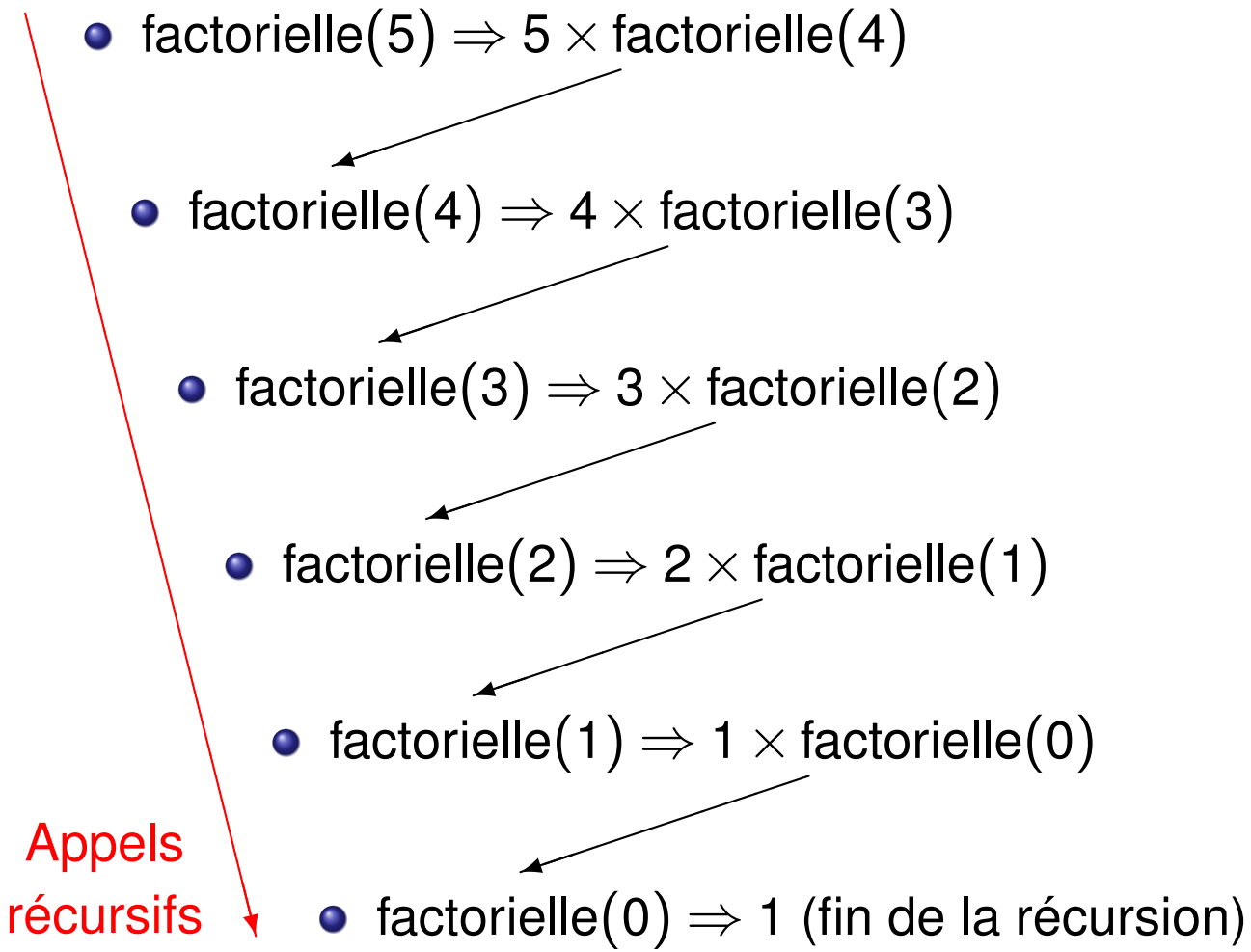
Exemple d'exécution

- factorielle(5) $\Rightarrow 5 \times$ factorielle(4)
  - factorielle(4) $\Rightarrow 4 \times$ factorielle(3)
    - factorielle(3) $\Rightarrow 3 \times$ factorielle(2)
      - factorielle(2) $\Rightarrow 2 \times$ factorielle(1)
        - factorielle(1) $\Rightarrow 1 \times$ factorielle(0)

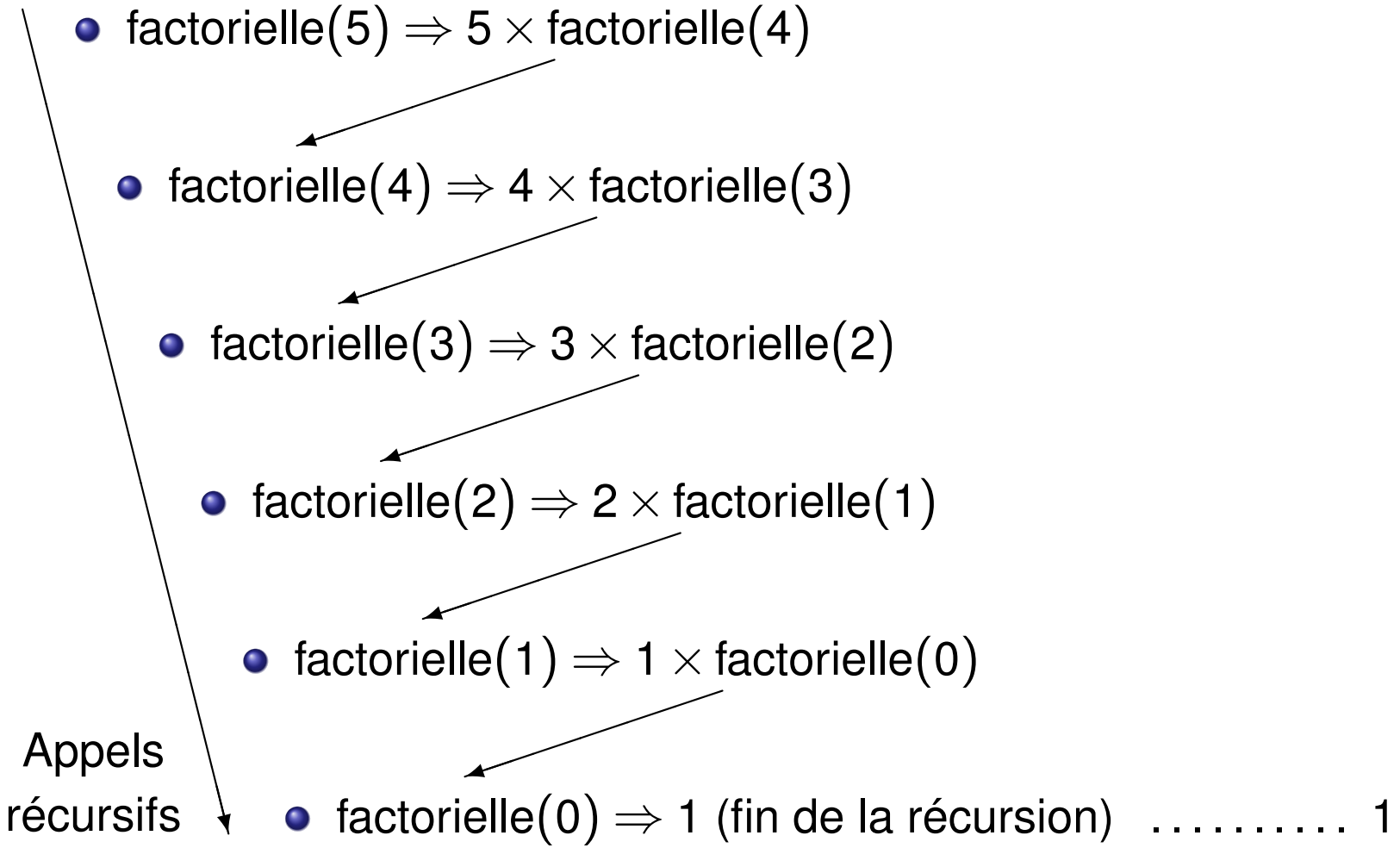
Exemple d'exécution

- factorielle$(5) \Rightarrow 5 \times$ factorielle(4)
  - factorielle$(4) \Rightarrow 4 \times$ factorielle(3)
    - factorielle$(3) \Rightarrow 3 \times$ factorielle(2)
      - factorielle$(2) \Rightarrow 2 \times$ factorielle(1)
        - factorielle$(1) \Rightarrow 1 \times$ factorielle(0)
          - factorielle$(0) \Rightarrow 1$ (fin de la récursion)

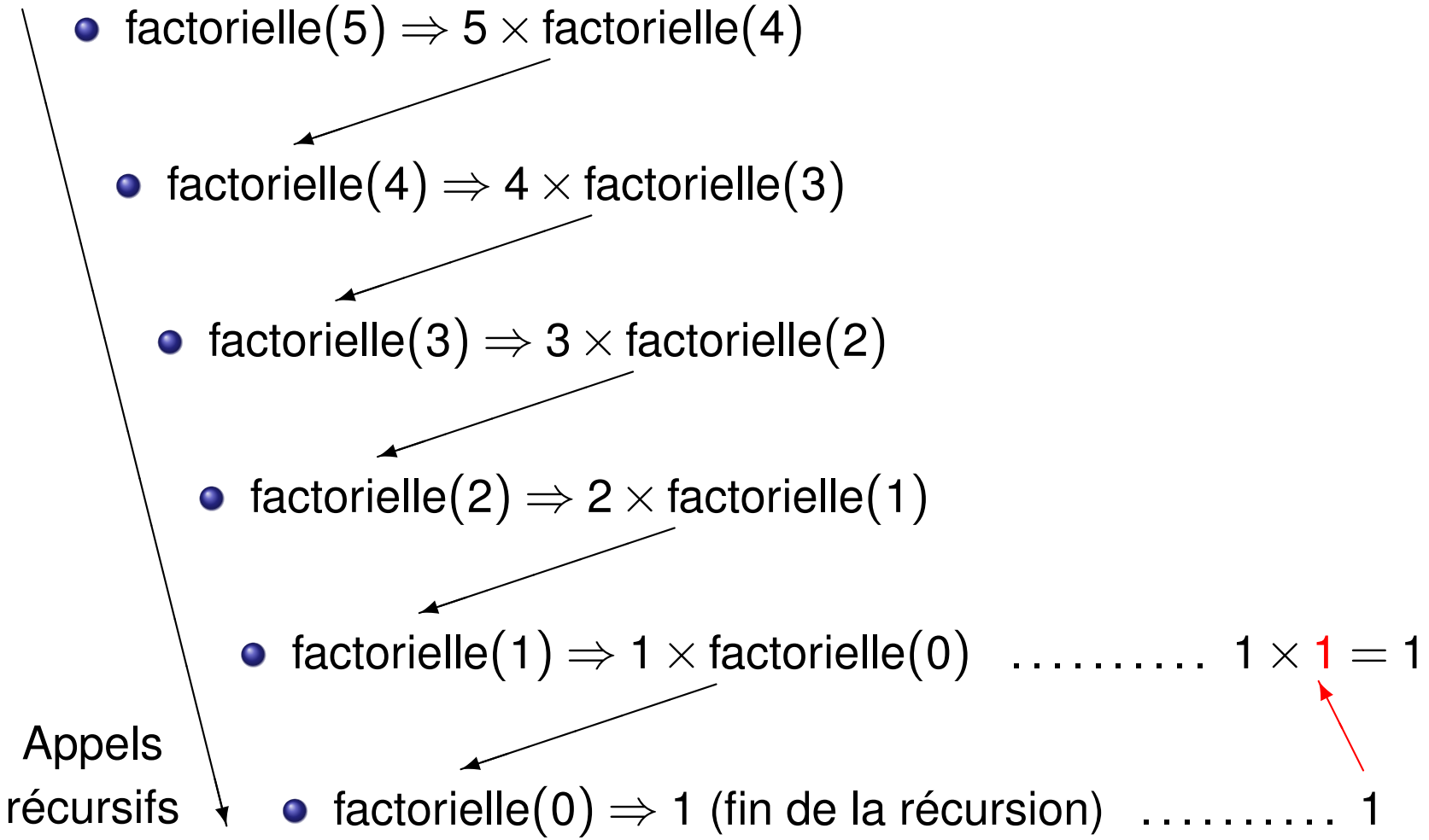
Exemple d'exécution



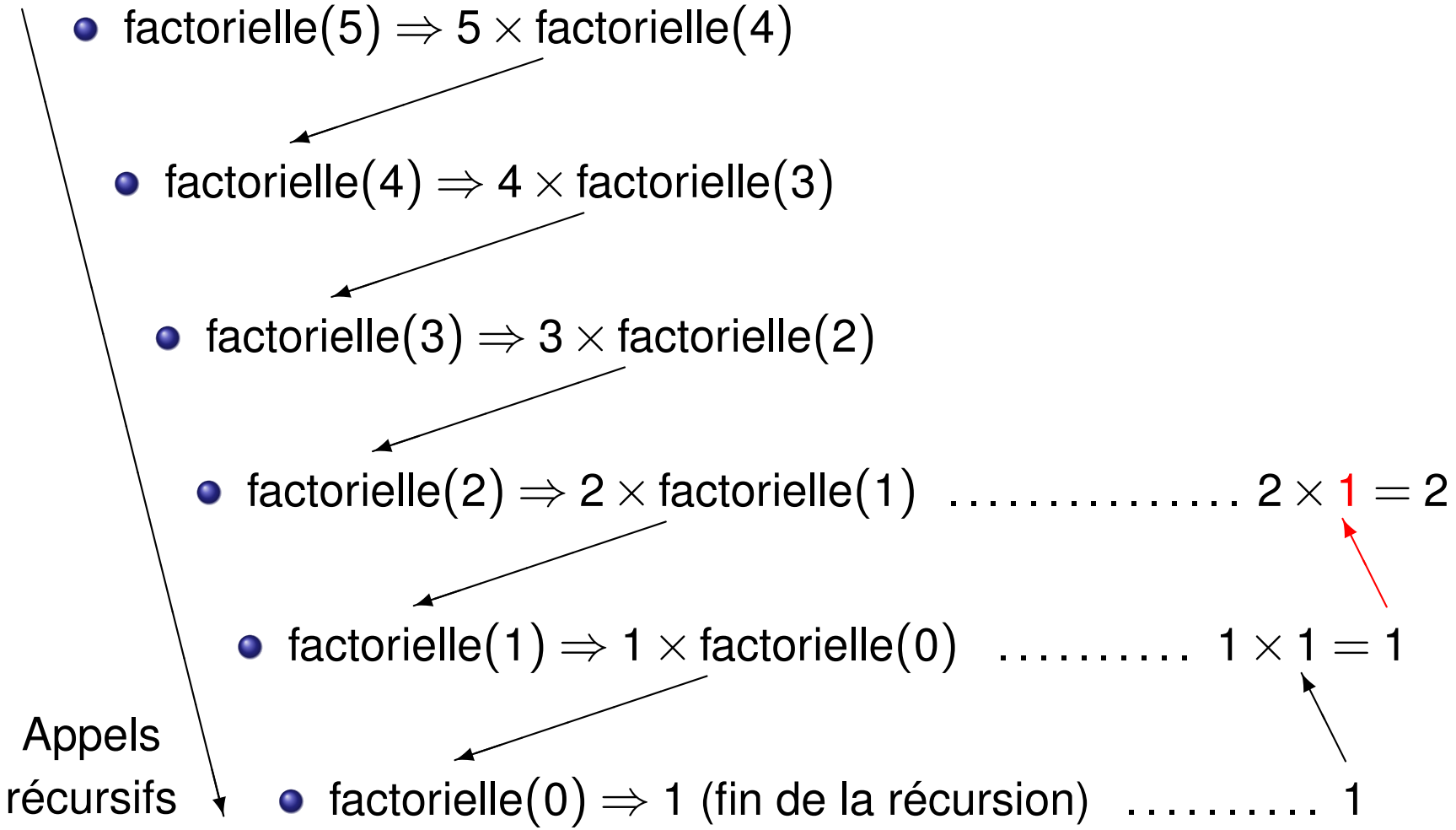
Exemple d'exécution



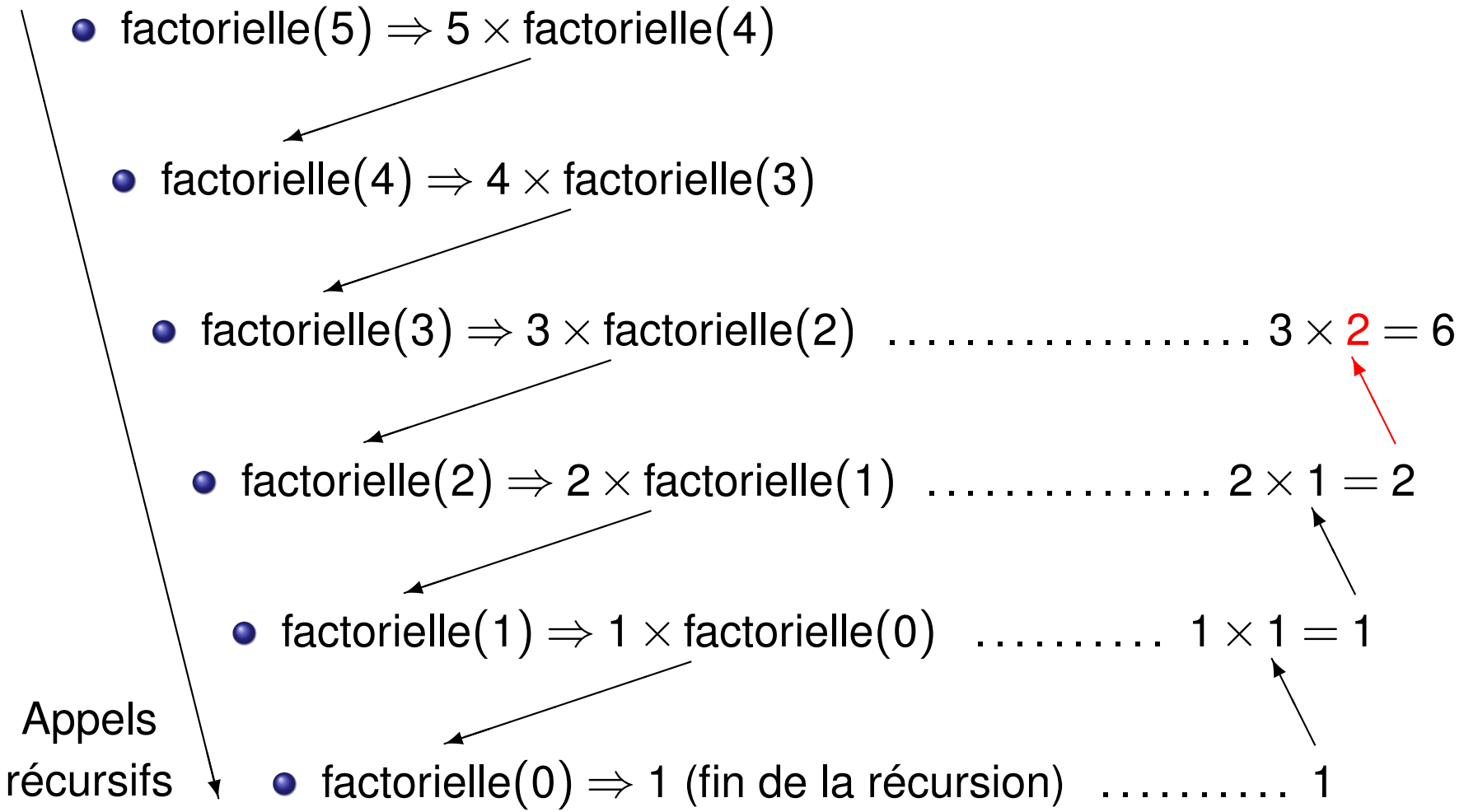
Exemple d'exécution



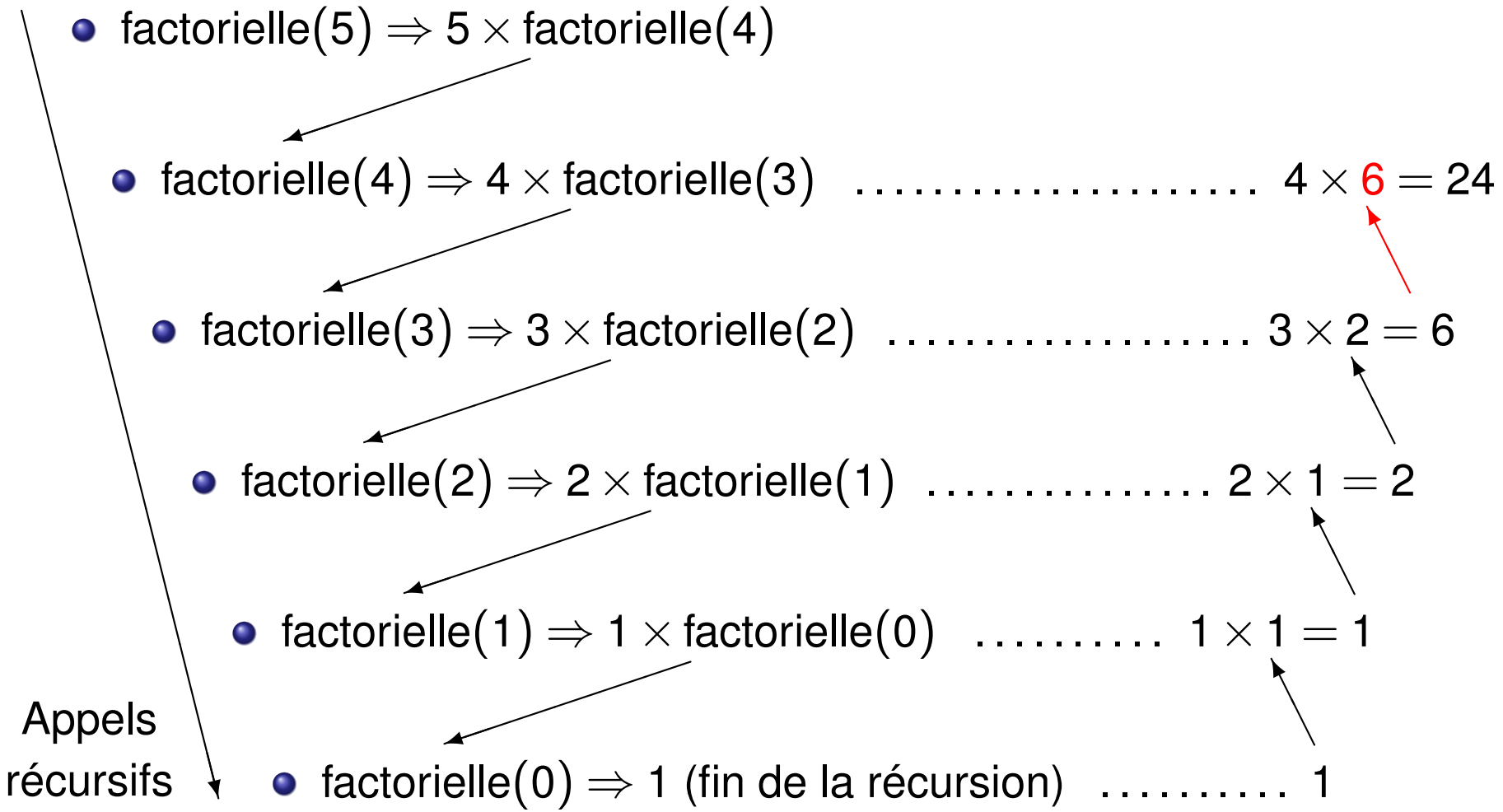
Exemple d'exécution



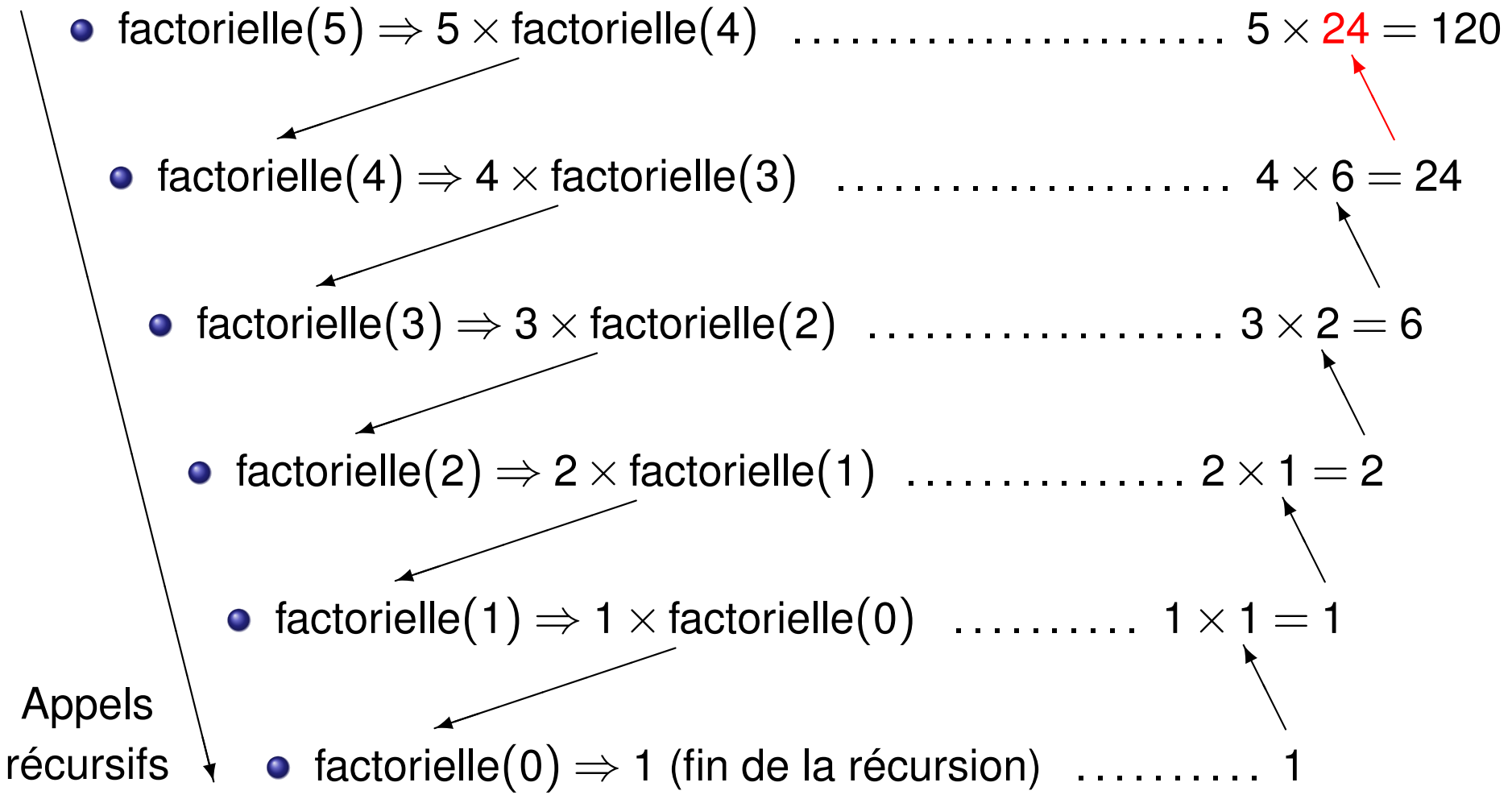
Exemple d'exécution



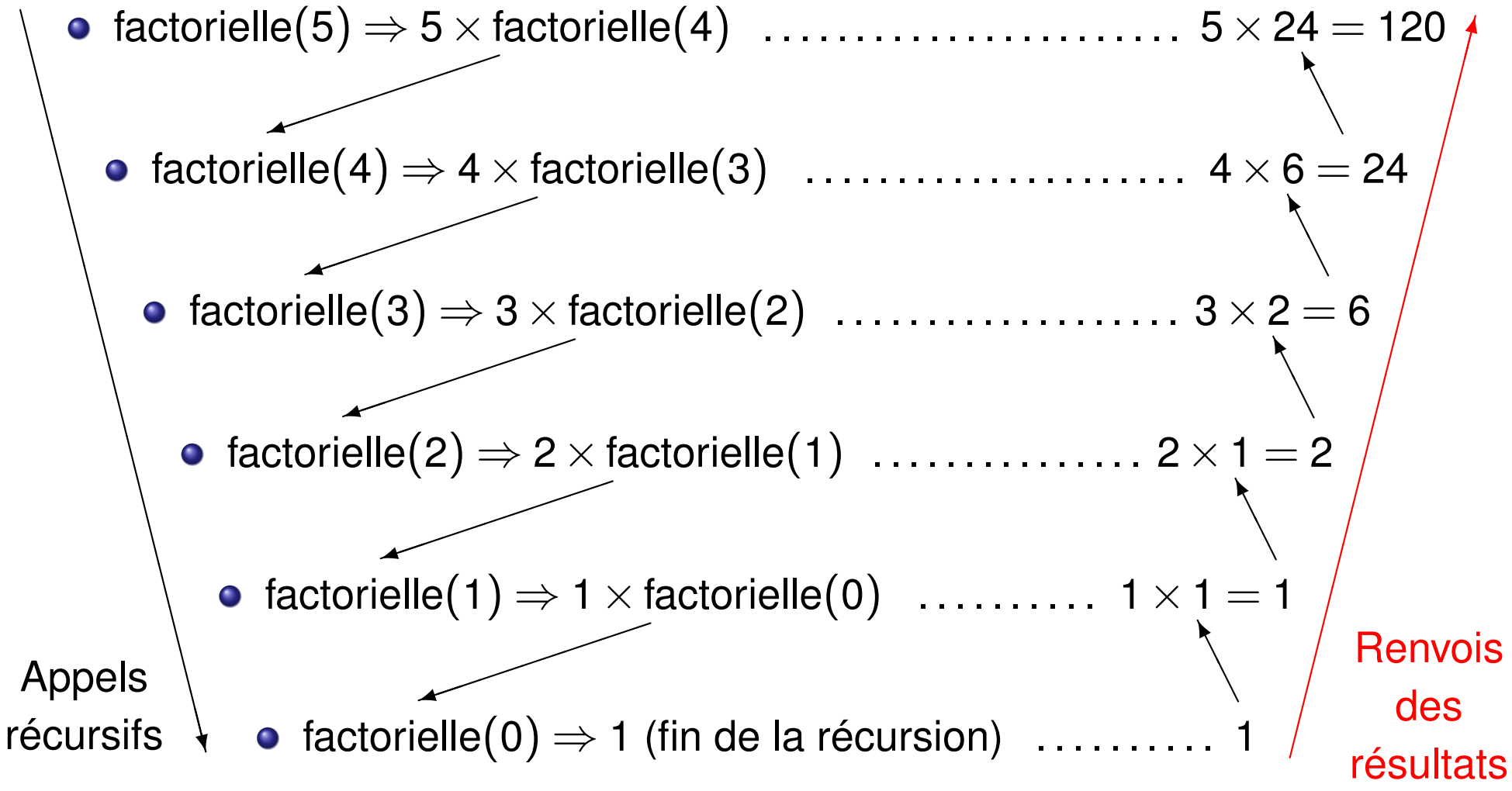
Exemple d'exécution



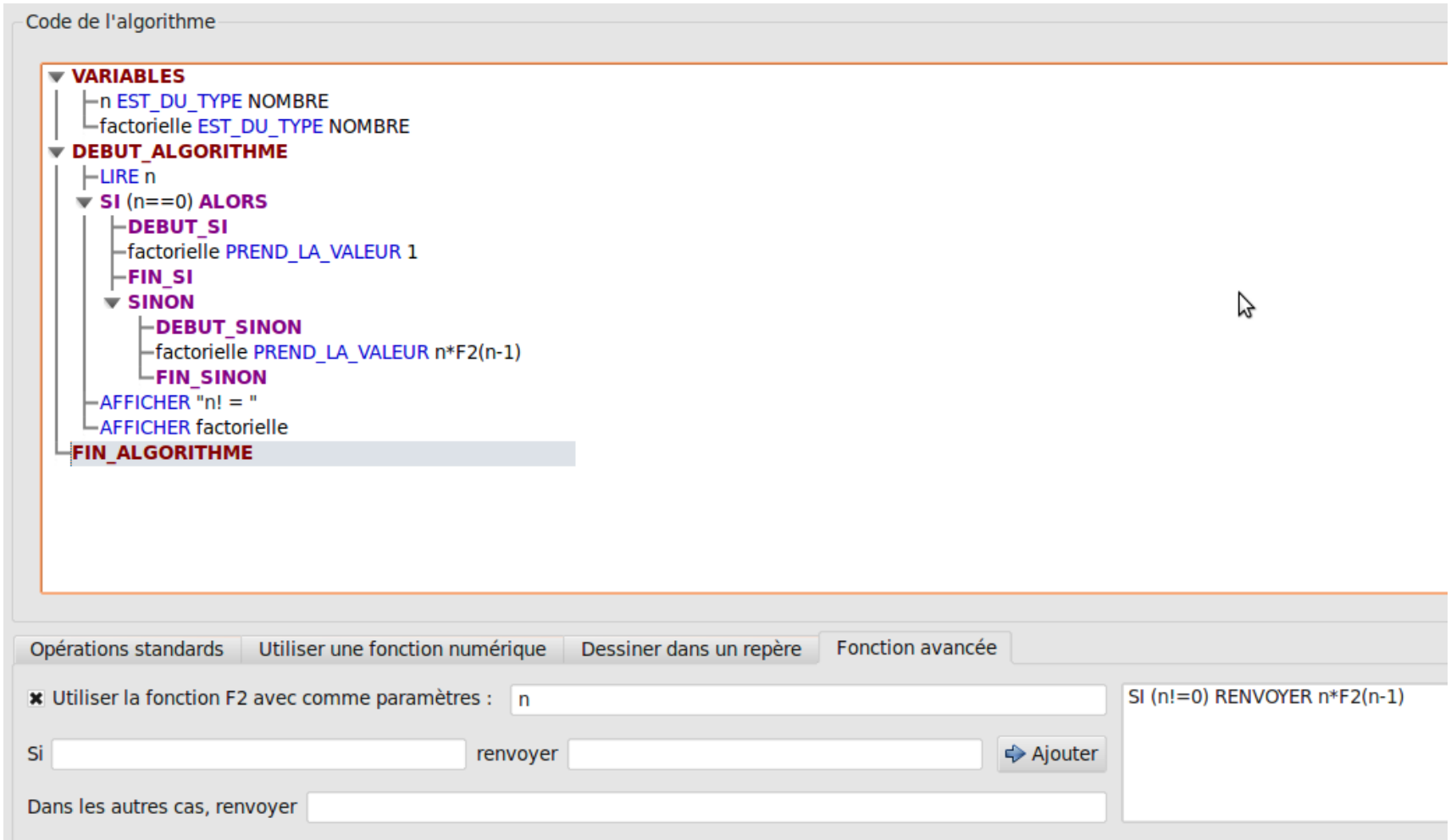
Exemple d'exécution



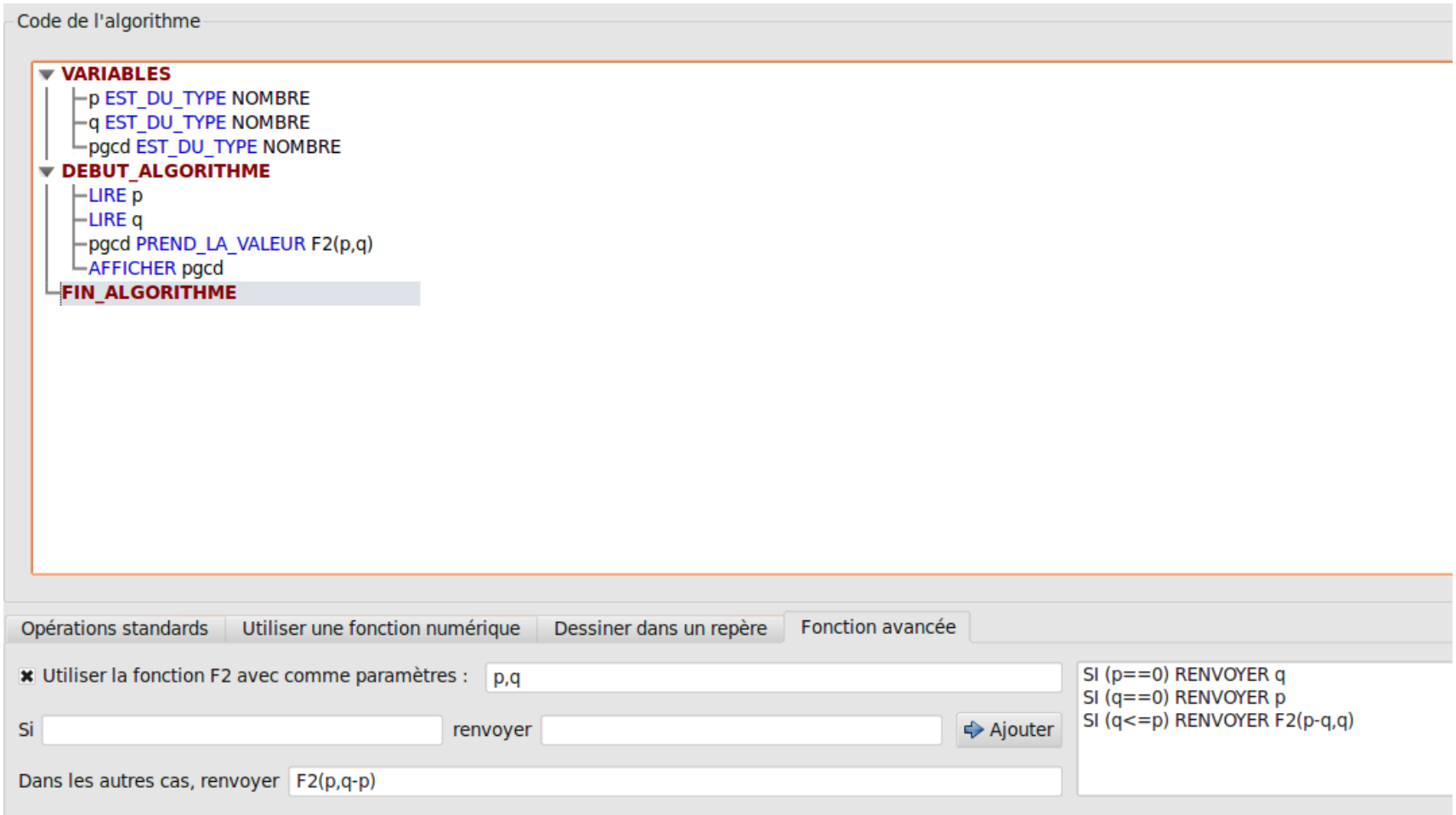
Exemple d'exécution



Exemple avec Algobox : factorielle



Exemple avec Algobox : pgcd



Caractéristiques des fonction récursives

- Une fonction récursive *valide* doit contenir :
  - *au moins* un appel à elle-même avec des *paramètres différents* ;
  - *au moins* un cas où elle ne s'appelle pas ;

  $\Rightarrow$ *au moins* une conditionnelle pour séparer ces différents cas.
- On dit que la récursivité est *terminale* lorsque l'appel récursif est la *dernière instruction* réalisée lors de l'appel courant.
- S'il y a des traitements *après* l'appel récursif, on a une récursivité *non terminale*.

Deux exemples...

Exemple 1 : litÉcritOrdo

fonction litÉcritOrdo(**in** *nb* : entier) : **vide**

Lit, puis écrit dans le même ordre, *nb* nombres entiers.

Lexique local des variables

valeur (entier) valeur lue

Algorithme de litÉcritOrdo

si $nb > 0$ **alors**
  valeur $\leftarrow$ lire
  écrire *valeur*
  litÉcritOrdo$(nb-1)$
fsi

Deux exemples...

Exemple 1 : litÉcritOrdo

fonction litÉcritOrdo(**in** *nb* : entier) : **vide**

Lit, puis écrit dans le même ordre, *nb* nombres entiers.

Lexique local des variables

valeur (entier) valeur lue

Algorithme de litÉcritOrdo

si $nb > 0$ **alors**
  valeur $\leftarrow$ lire
  écrire *valeur*
  litÉcritOrdo($nb - 1$)
fsi

- La récursivité est terminale.

Deux exemples

Exemple 2 : litÉcritInv

fonction litÉcritInv(**in** *nb* : entier) : **vide**

Lit, puis écrit dans l'ordre inverse, *nb* nombres entiers.

Lexique local des variables

valeur (entier) valeur lue

Algorithme de litÉcritInv

si $nb > 0$ **alors**
  valeur $\leftarrow$ lire
  litÉcritInv$(nb - 1)$
  écrire *valeur*
fsi

Deux exemples

Exemple 2 : litÉcritInv

fonction litÉcritInv(**in** *nb* : entier) : **vide**

Lit, puis écrit dans l'ordre inverse, *nb* nombres entiers.

Lexique local des variables

valeur (entier) valeur lue

Algorithme de litÉcritInv

si $nb > 0$ **alors**
  valeur $\leftarrow$ lire
  litÉcritInv($nb - 1$)
  écrire *valeur*
fsi

- La récursivité n'est pas terminale.

Autre exemple

- Calcul du coefficient binomial $\binom{n}{k}$:

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

Calcul de $\binom{n}{k}$

fonction cnk(**in** n : entier, **in** k : entier) : **ret** entier

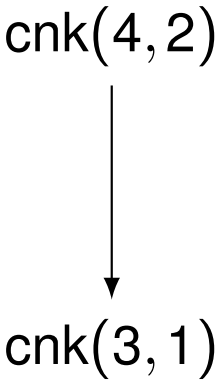
Algorithme de cnk

si $k > n$ **alors**
  retourner 0
sinon si $k = 0 \vee k = n$ **alors**
  retourner 1
sinon
  retourner $\text{cnk}(n-1, k-1) + \text{cnk}(n-1, k)$
fsi

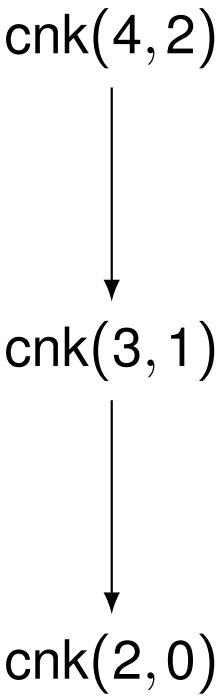
Exécution de cnk(4, 2)

cnk(4, 2)

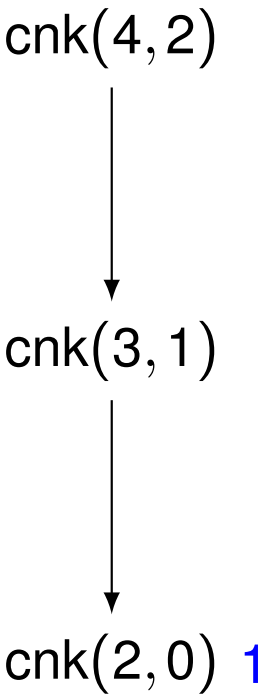
Exécution de cnk(4, 2)



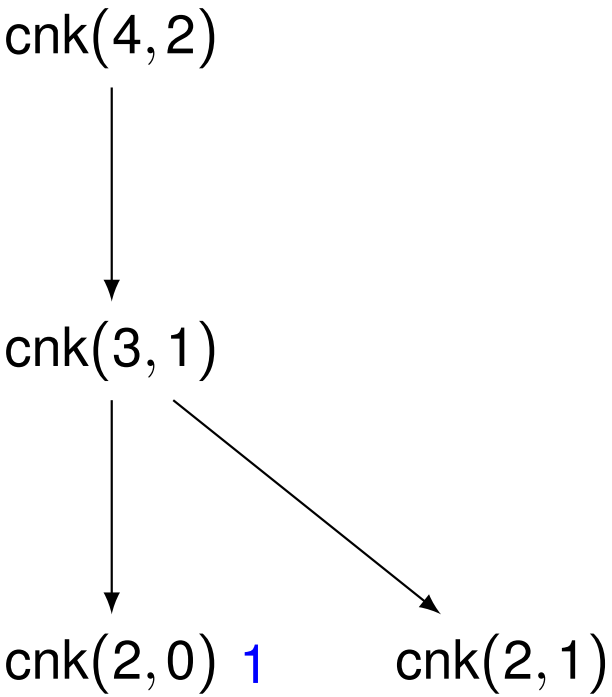
Exécution de cnk(4,2)



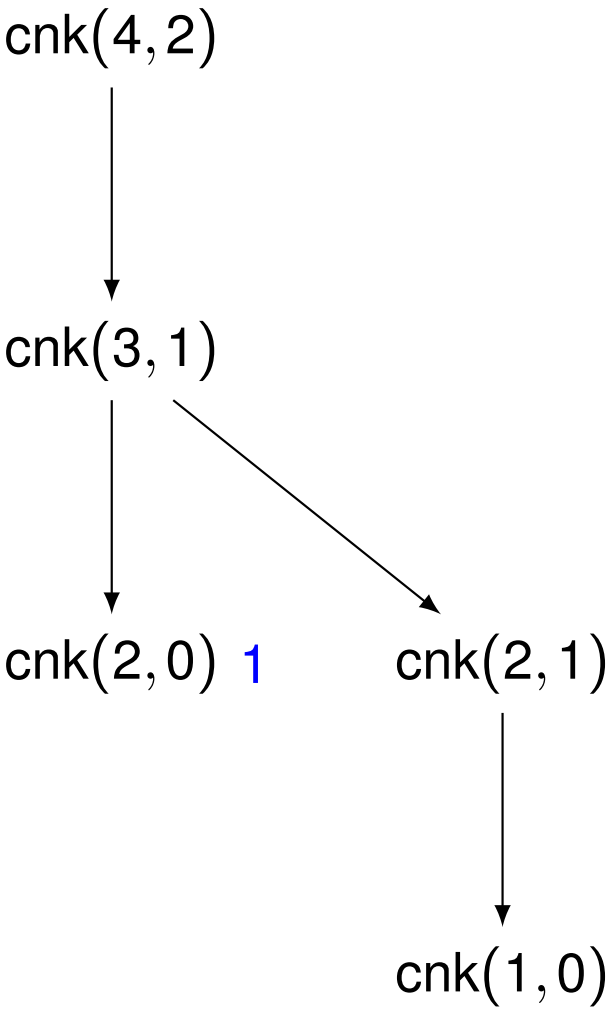
Exécution de cnk(4,2)



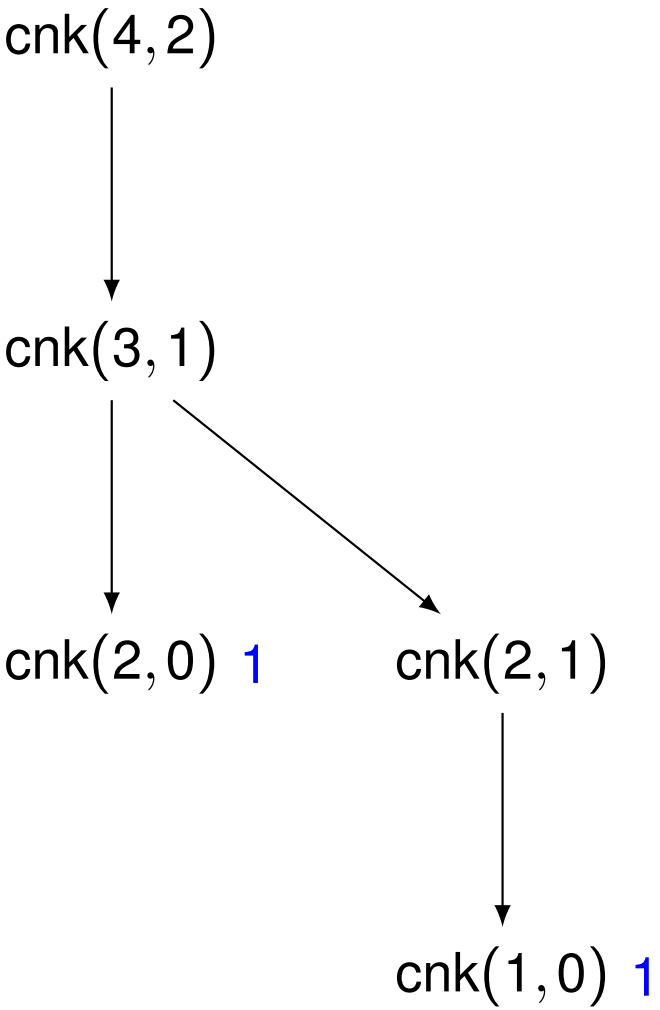
Exécution de cnk(4, 2)



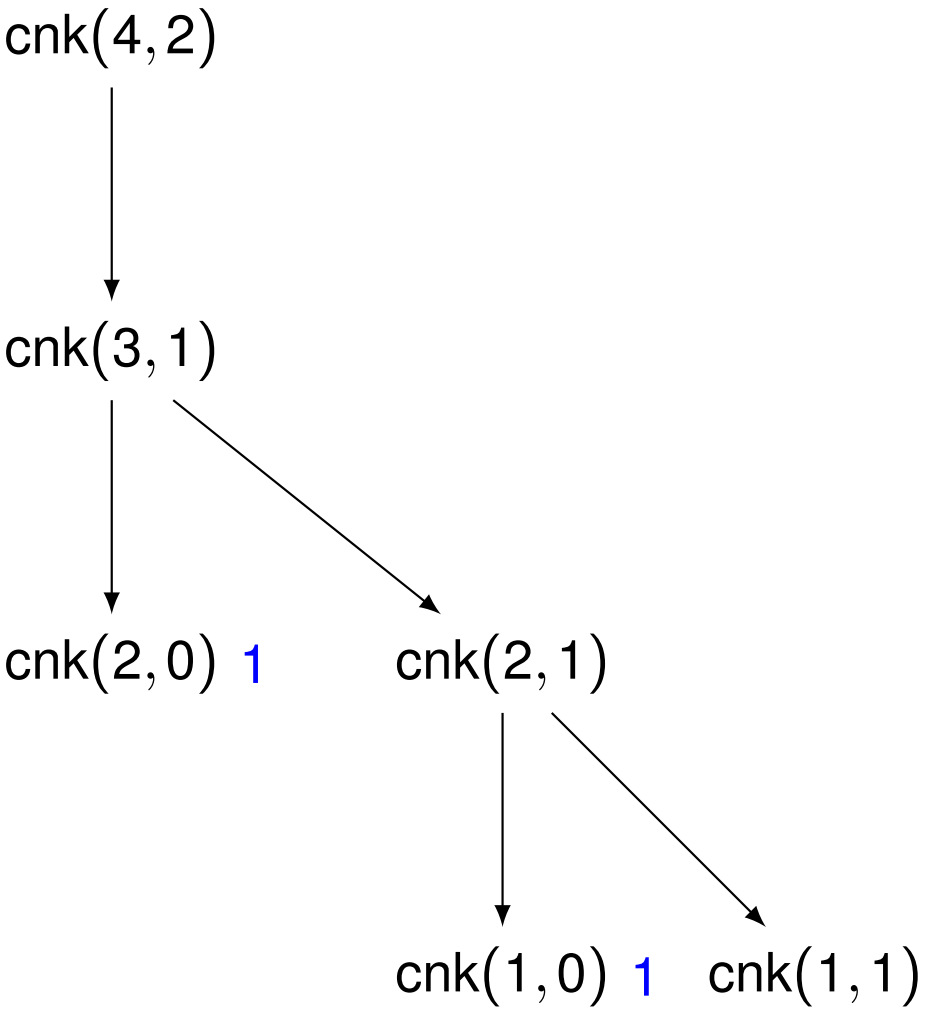
Exécution de cnk(4,2)



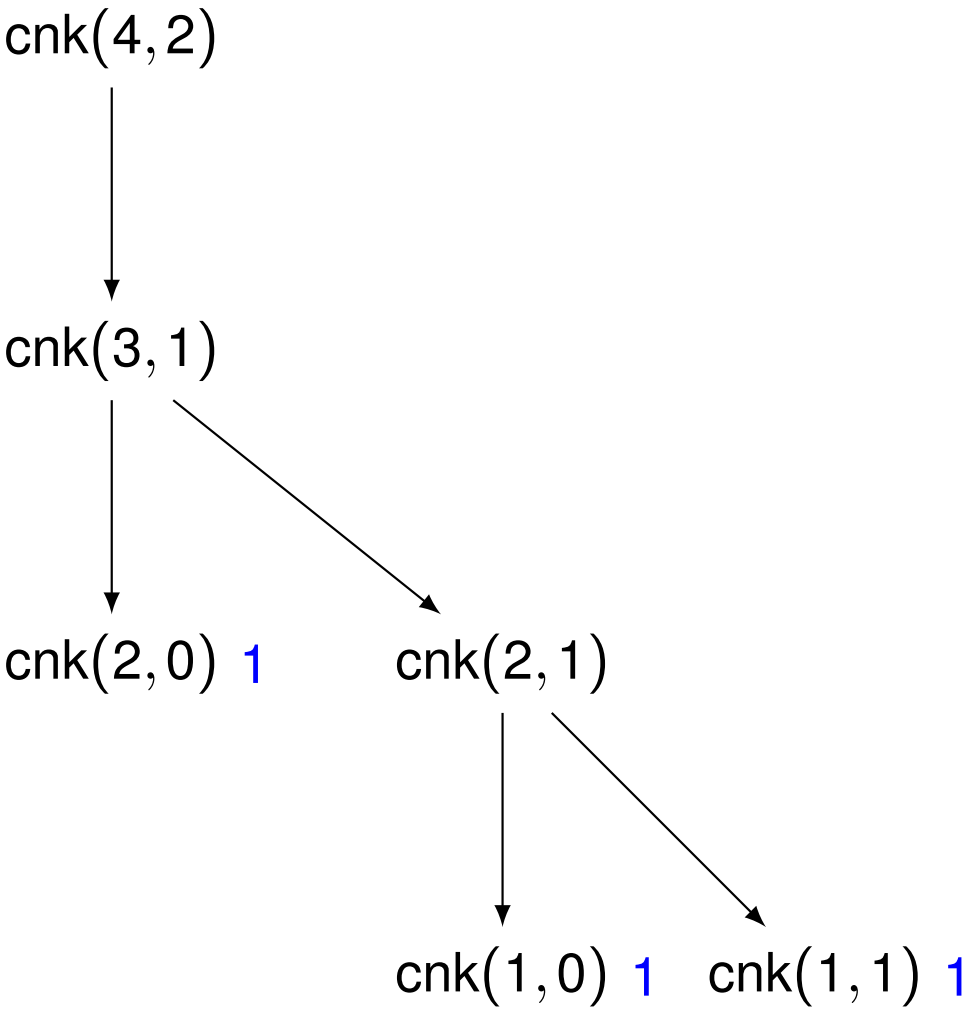
Exécution de cnk(4, 2)



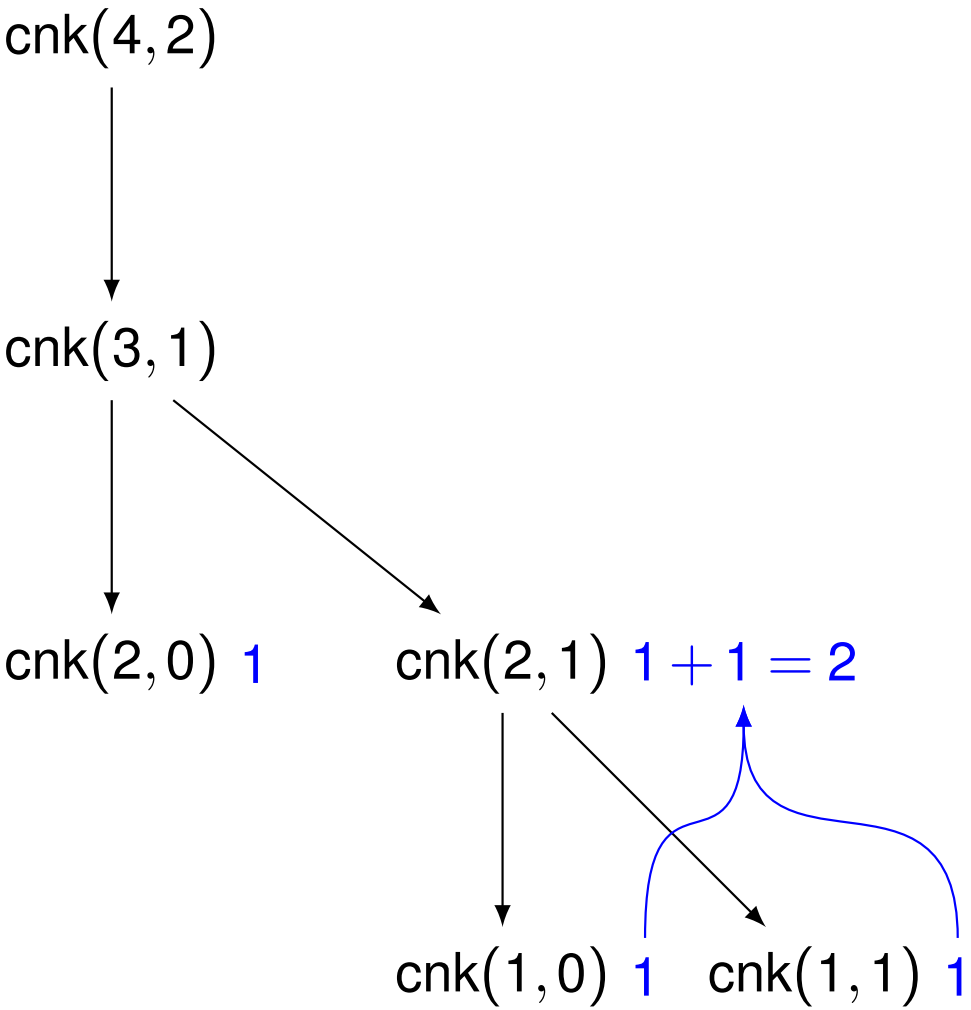
Exécution de cnk(4,2)



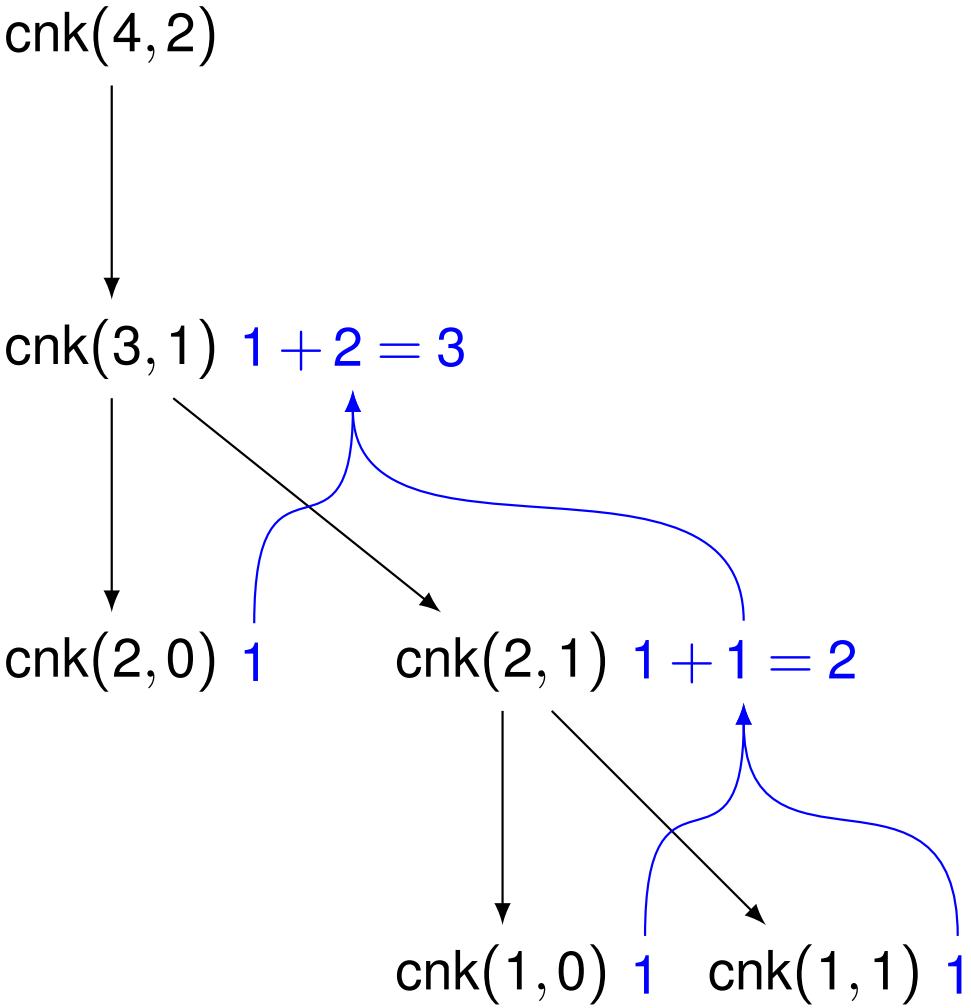
Exécution de cnk(4,2)



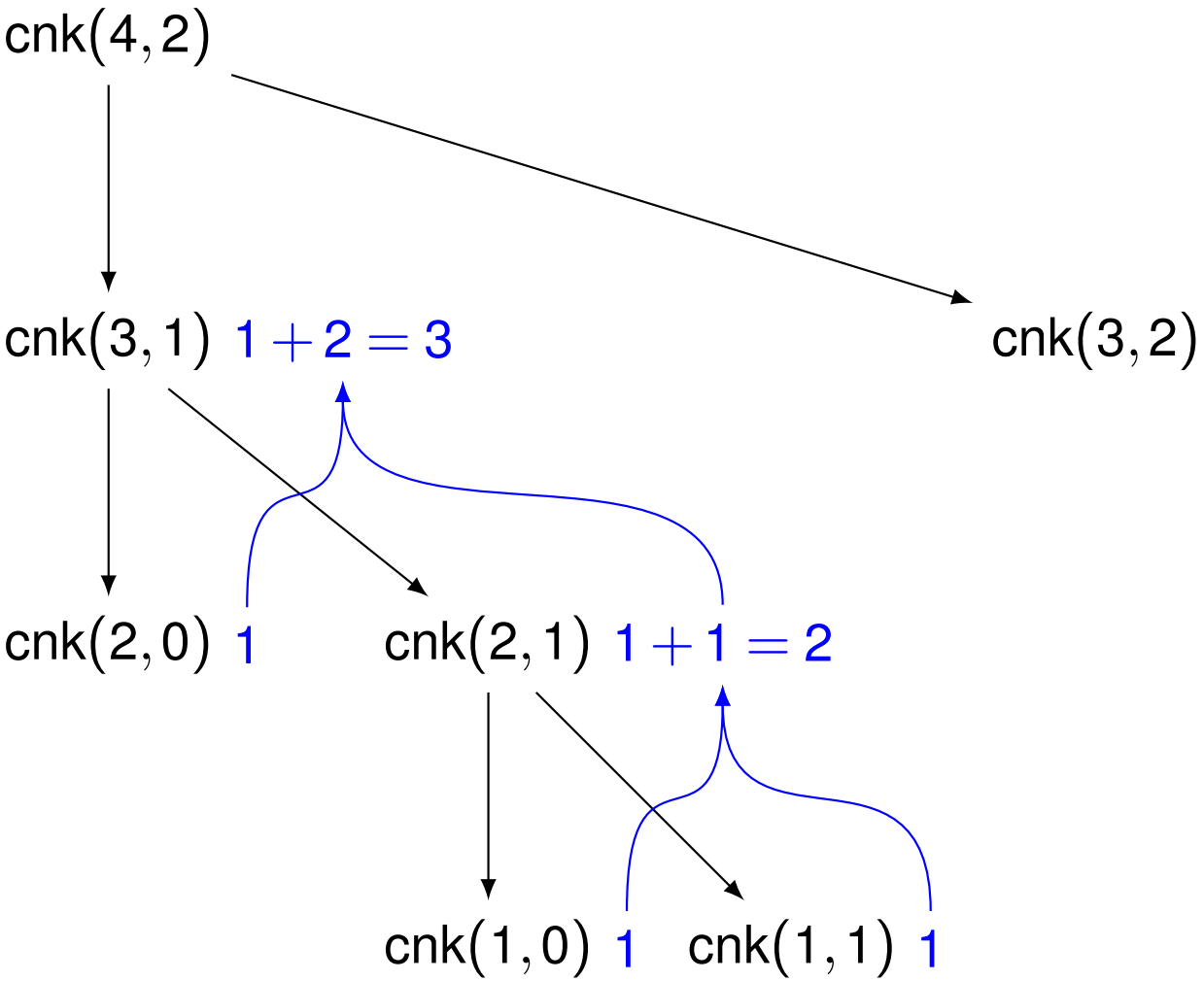
Exécution de cnk(4,2)



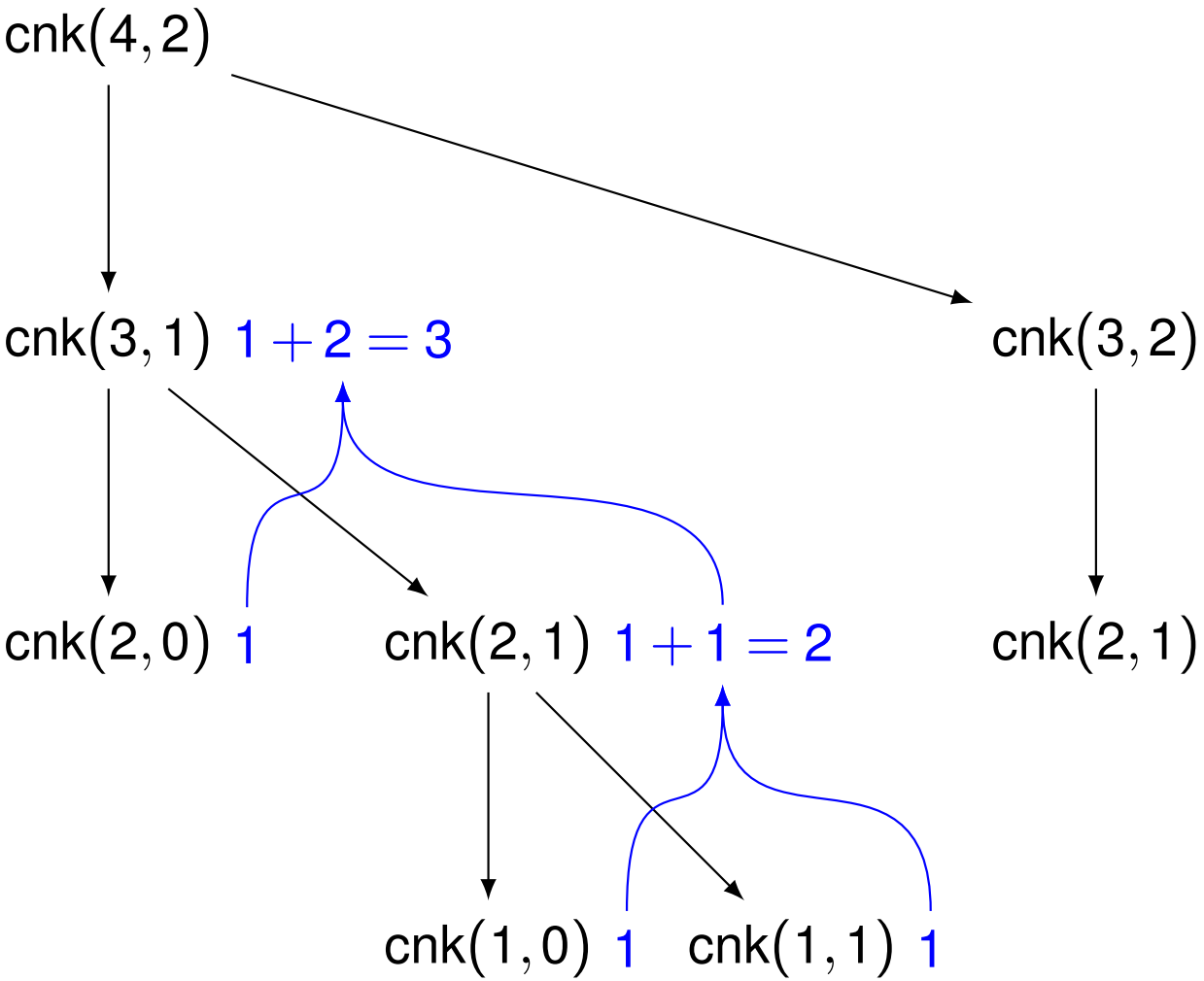
Exécution de cnk(4,2)



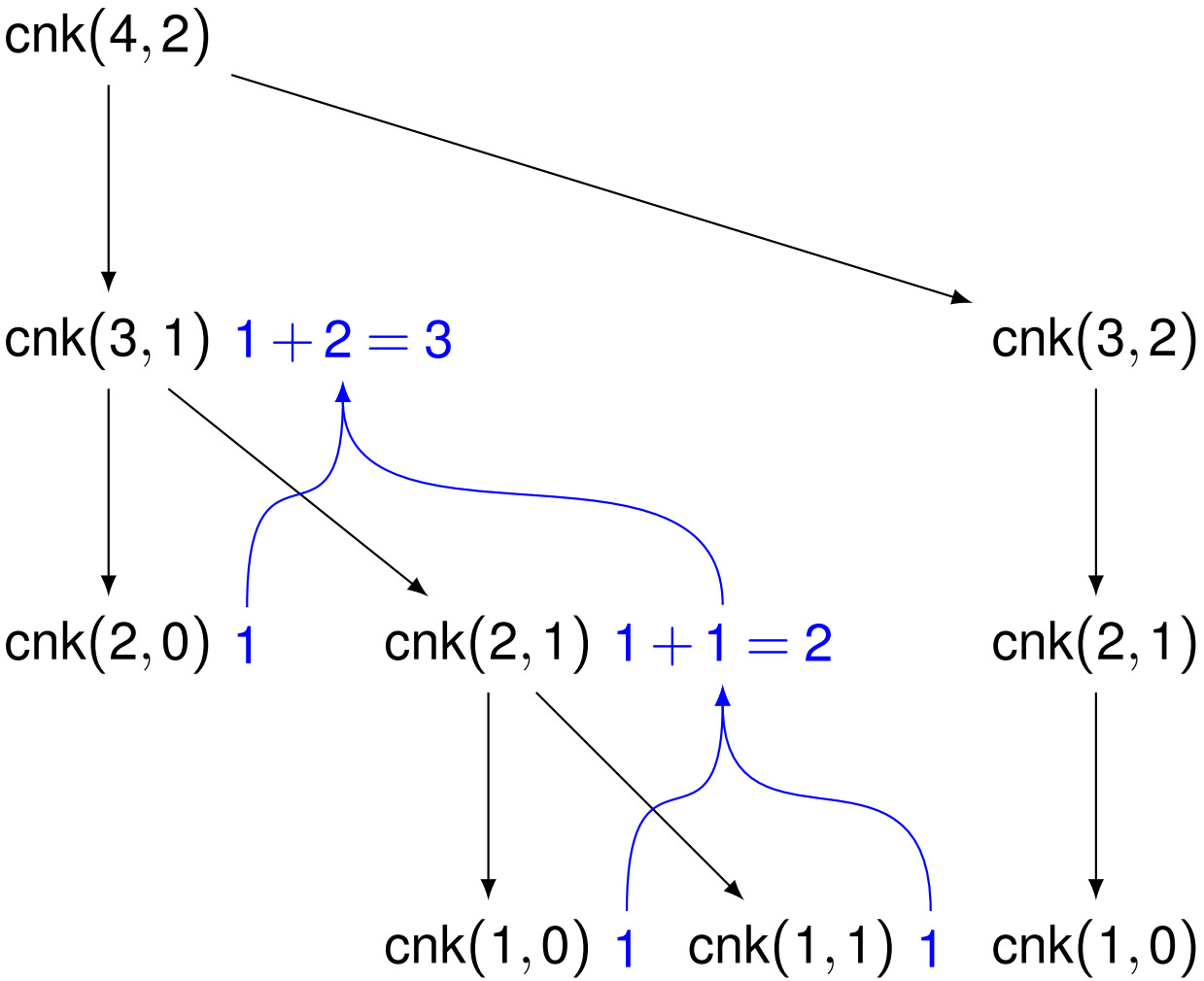
Exécution de cnk(4,2)



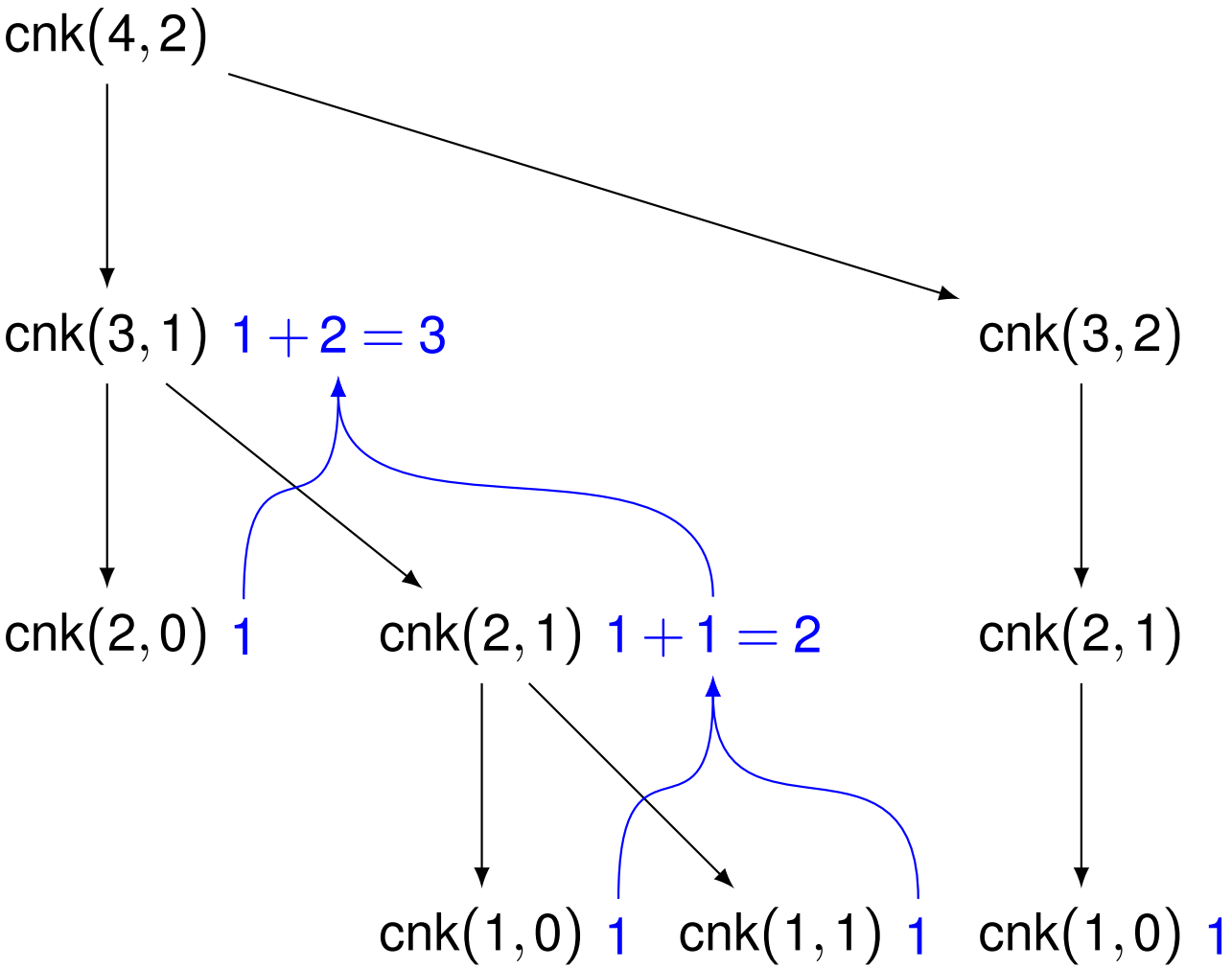
Exécution de cnk(4,2)



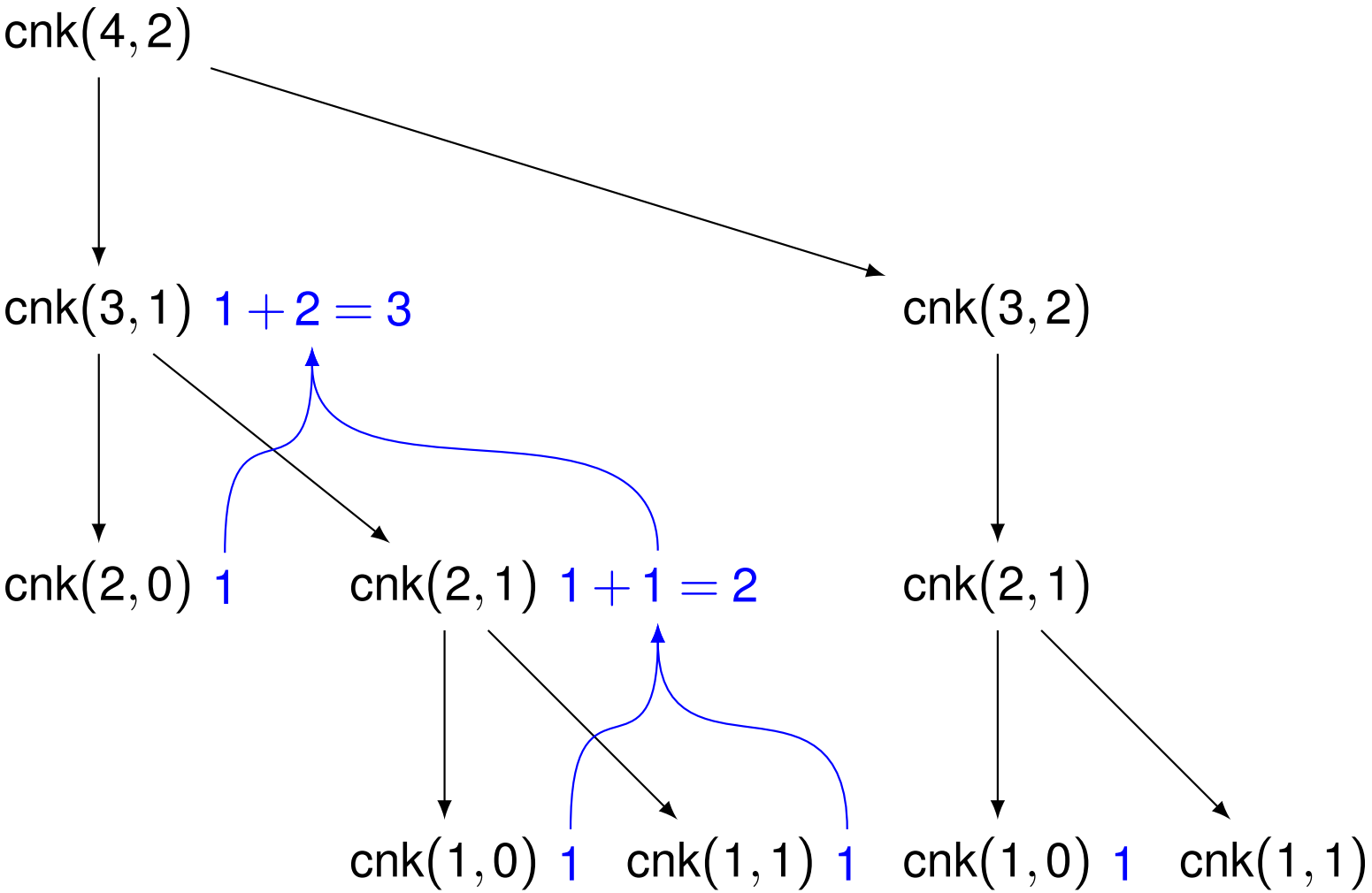
Exécution de cnk(4,2)



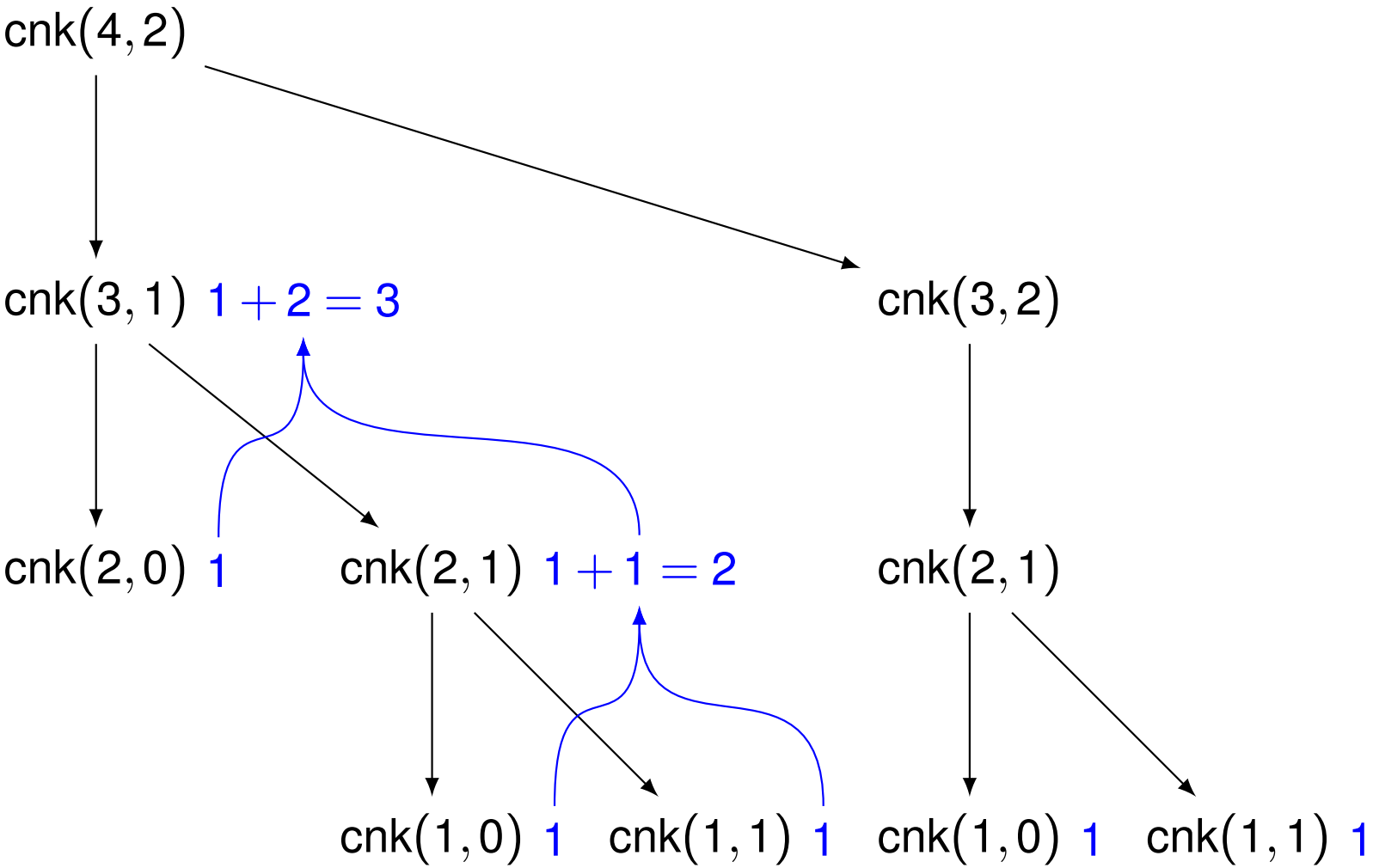
Exécution de cnk(4,2)



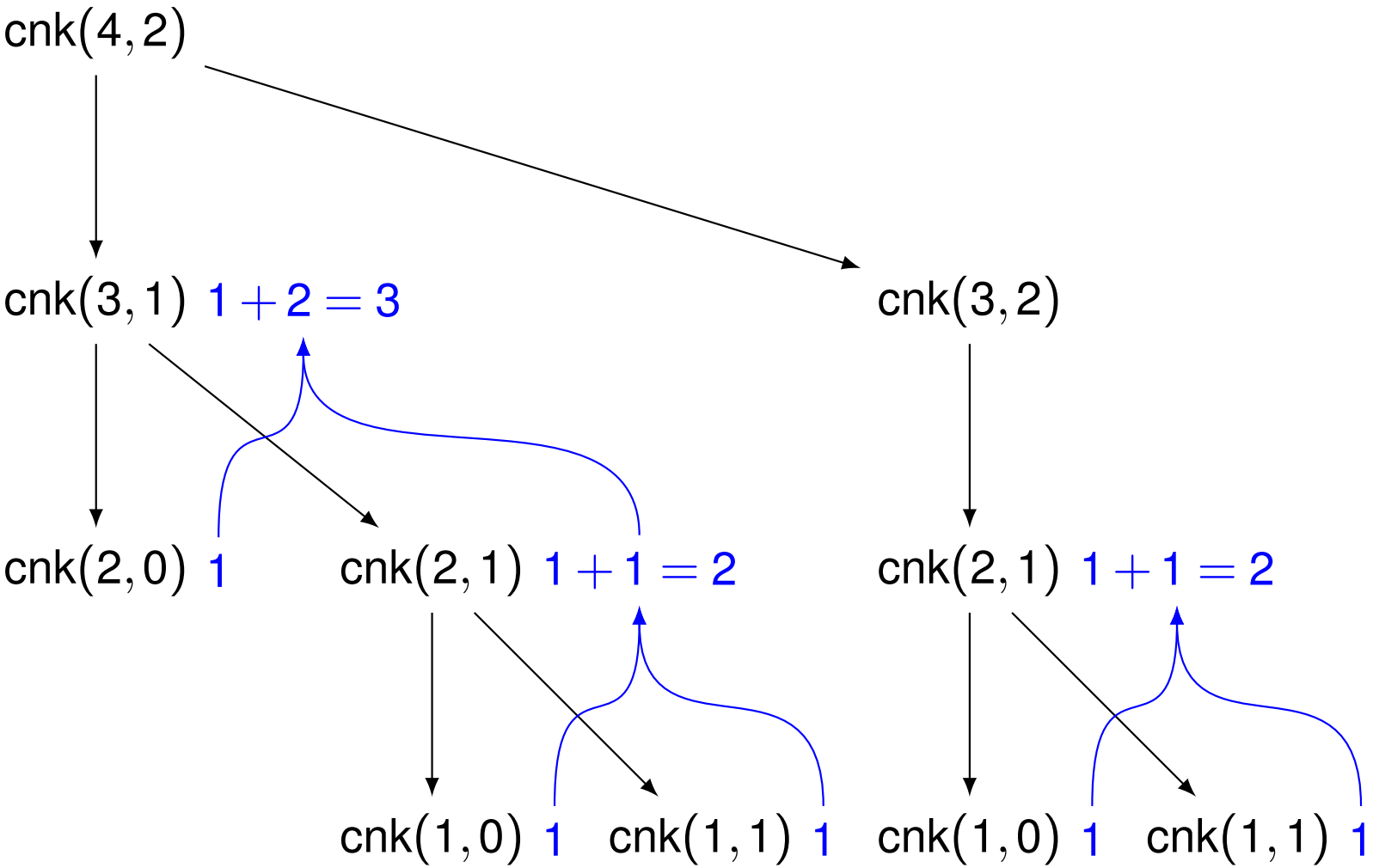
Exécution de cnk(4,2)



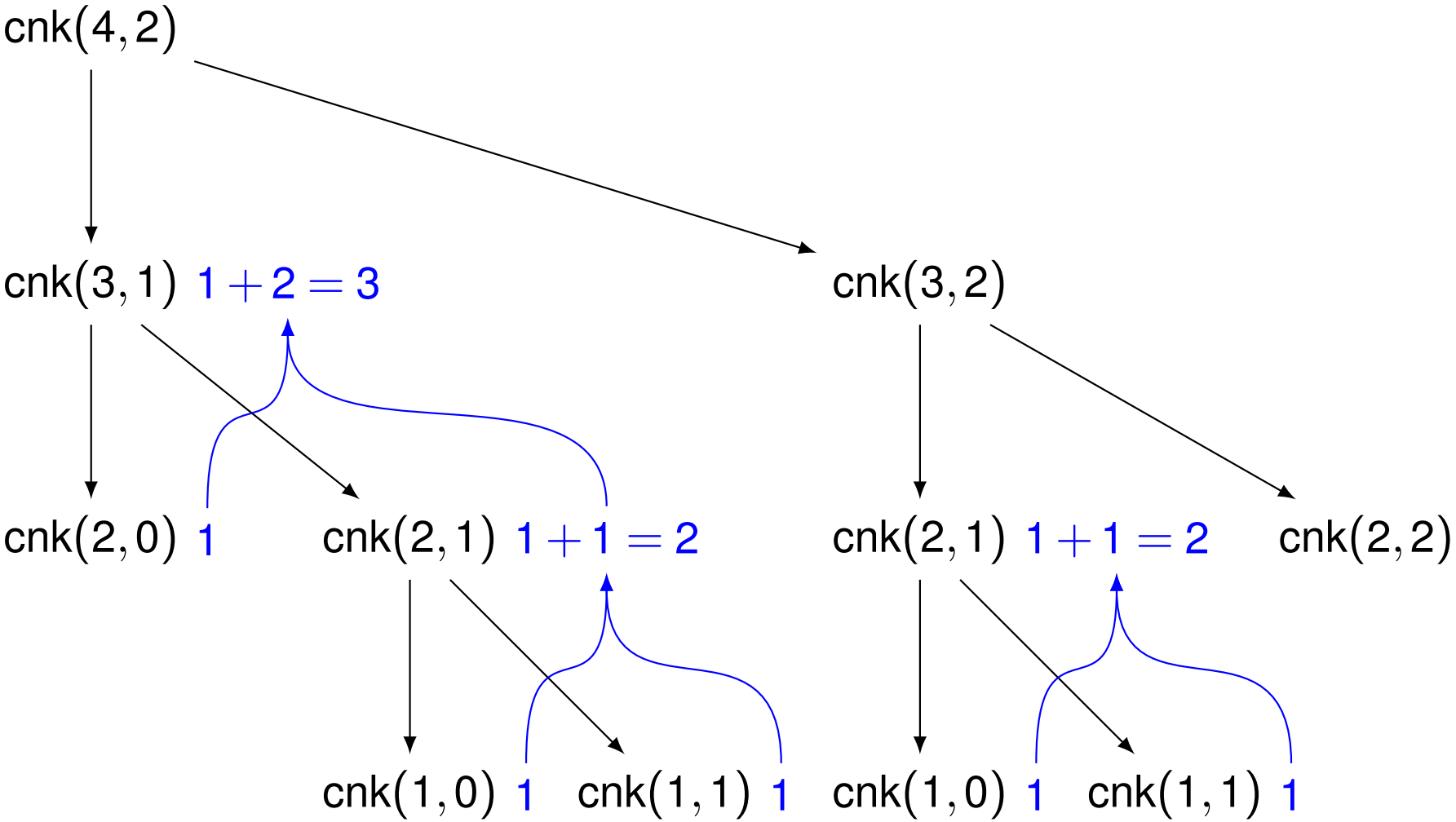
Exécution de cnk(4,2)



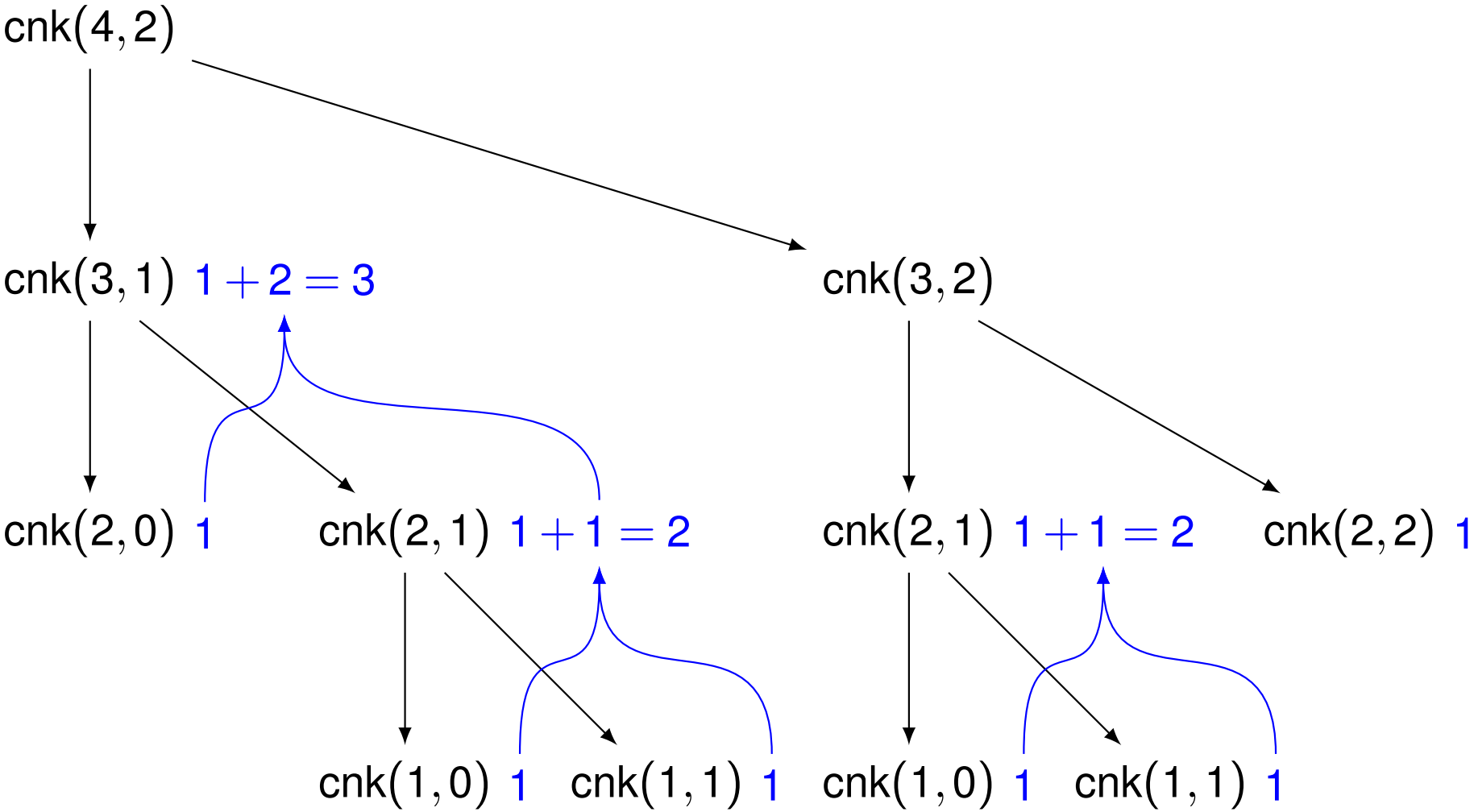
Exécution de cnk(4,2)



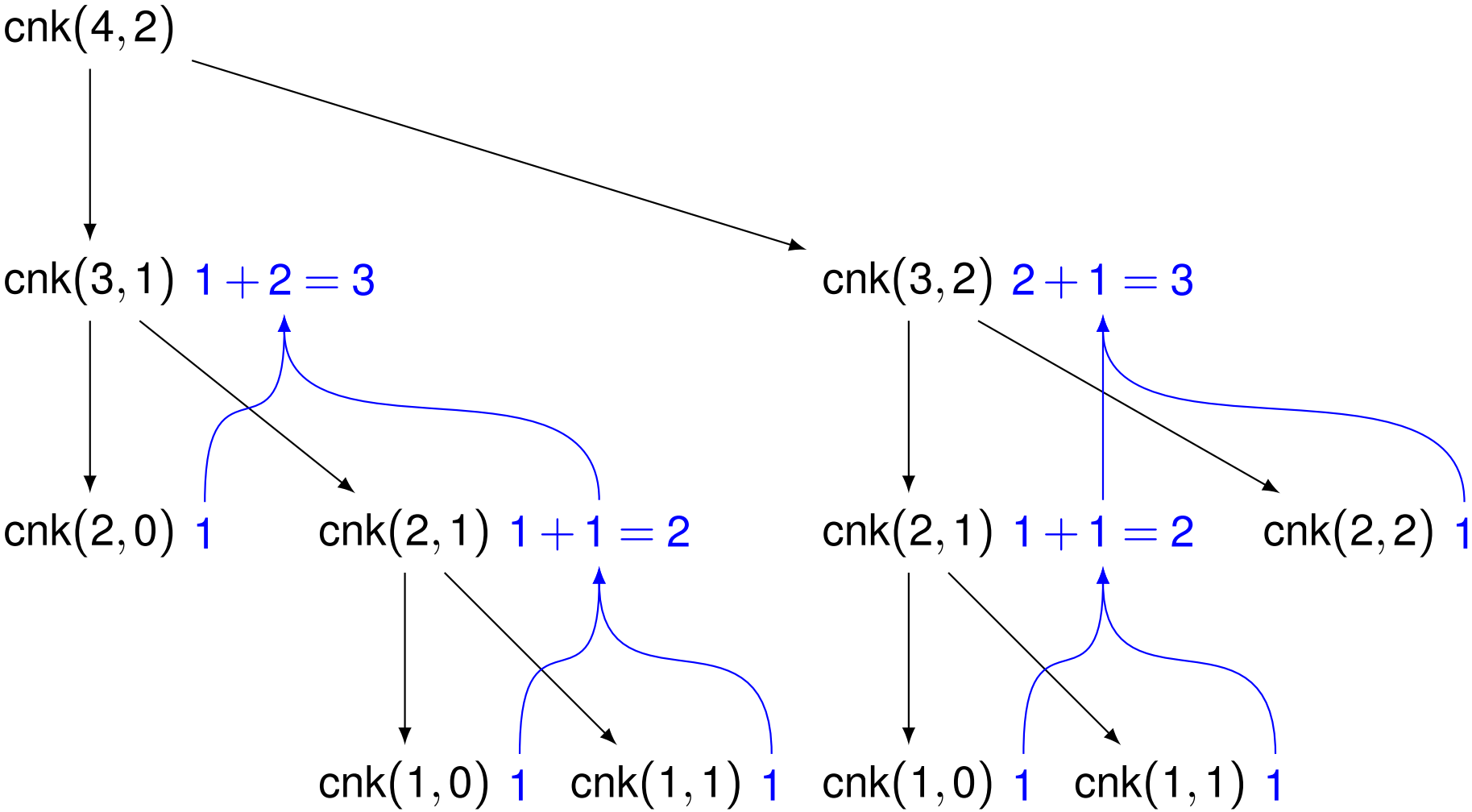
Exécution de cnk(4,2)



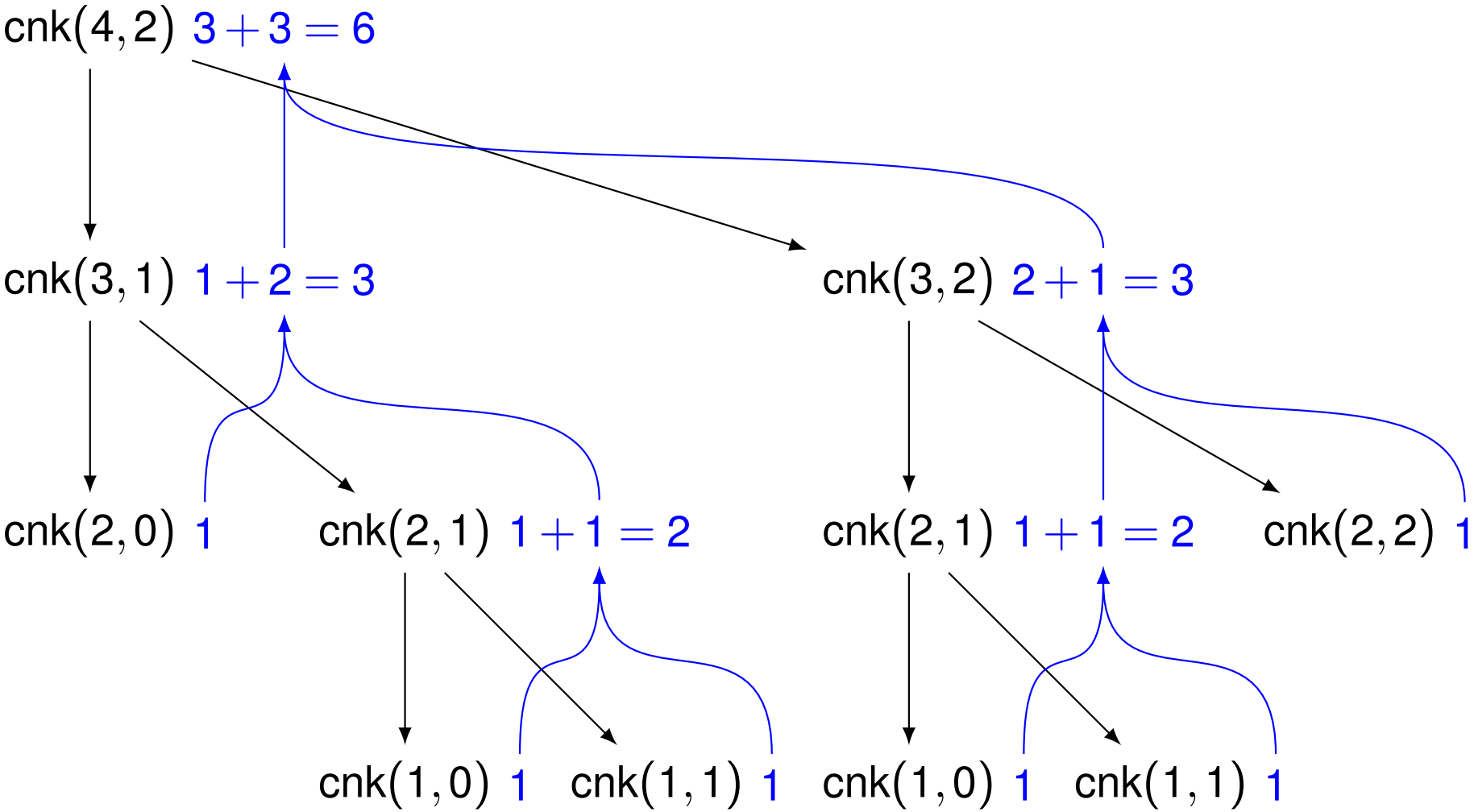
Exécution de cnk(4,2)



Exécution de cnk(4,2)



Exécution de cnk(4,2)



Plan

Les tableaux

- 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  un tableau de 6 réels

Déclaration (exemple)

Lexique des variables

tNotes	(tableau [6] de réels)	Liste de notes	INTERMÉDIAIRE

Les tableaux

- 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  un tableau de 6 réels

Déclaration (exemple)

Lexique des variables

tNotes (tableau [6] de réels) Liste de notes INTERMÉDIAIRE

Type des éléments du tableau

Nombre d'éléments dans le tableau

Les tableaux

- 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  un tableau de 6 réels

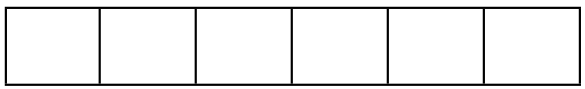
Déclaration (exemple)

Lexique des variables

tNotes	(tableau [6] de réels)	Liste de notes	INTERMÉDIAIRE

- Chaque élément est repéré dans le tableau par un indice qui varie de 0 à $taille - 1$ (ex. : $tNotes[-1]$ et $tNotes[6]$ n'existent pas !).

  0 1 2 3 4 5

  tNotes 
- On accède à la case 2 par $tNotes[2]$. Attention, c'est la 3e case !

Tableaux de constantes

- Placé dans le lexique des constantes.
- On indique le contenu par la liste des valeurs entre $\{\ldots\}$.

Exemple

- On a besoin de la liste des nombres de jours des mois de l'année
  $\Rightarrow$ on peut définir un tableau dans le lexique des constantes

---

Lexique des constantes

tJoursMois	(tableau [12] d'entiers)	$= \{31,28,31,30,31,30,$ $31,31,30,31,30,31\}$	nombre de jours par mois

---

Initialisation d'un tableau

- Lorsqu'on connaît les valeurs initiales à placer dans un tableau on peut le faire en une seule instruction (attention ceci dépend du langage de programmation utilisé) :

$$\textit{tableau} \leftarrow \{\textit{liste de valeurs}\}$$

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

Exemple

Lexique des variables

tMesValeurs (tableau [8] d'entiers) ... INTERMÉDIAIRE

Algorithme

```
// Initialisation du tableau tMesValeurs
```

tMesValeurs $\leftarrow \{5, 9, -4, 2, 12, 17, 2, 7\}$

...

Lecture/affichage d'un tableau

- 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.

Lecture/affichage d'un tableau

- 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

Algorithme

pour i **de** 0 **à** $nbElem - 1$ **faire** `// boucle de lecture...`
  $tab[i] \leftarrow$ lire
fpour
...
pour i **de** 0 **à** $nbElem - 1$ **faire** `// boucle d'écriture...`
  écrire $tab[i]$
fpour

Lecture/affichage d'un tableau

- 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

Algorithme

pour *i* **de** 0 **à** *nbElem* − 1 **faire** `// boucle de lecture...`
  tab[*i*] ← lire
fpour
...
pour *i* **de** 0 **à** *nbElem* − 1 **faire** `// boucle d'écriture...`
  écrire *tab*[*i*]
fpour

- Attention à ne pas *sortir* du tableau !

Tableaux avec Algobox

Code de l'algorithme

```
VARIABLES
    L EST_DU_TYPE LISTE
    i EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    L[1] PREND_LA_VALEUR 5:9:-4:2:12:17:2:7
    POUR i ALLANT_DE 1 A 8
        DEBUT_POUR
        AFFICHER L[i]
        AFFICHER " "
        FIN_POUR
FIN_ALGORITHME
```

Tableaux en Visual Basic

Déclaration

- Forme générale : `<nom>[<taille>]<type>`
- Exemple : `Dim tMesValeurs(8) as Integer`

Tableaux en Visual Basic

Déclaration

- Forme générale : `<nom>[<taille>]<type>`
- Exemple : `Dim tMesValeurs(8) as Integer`

Initialisation

- À la déclaration uniquement.
- Si la taille est omise, on obtient un tableau dynamique.

Tableaux en Visual Basic

Déclaration

- Forme générale : `<nom>[<taille>]<type>`
- Exemple : `Dim tMesValeurs(8) as Integer`

Initialisation

- À la déclaration uniquement.
- Si la taille est omise, on obtient un tableau dynamique.

Utilisation

- Similaire au langage algorithmique.
- Les indices sont placés entre parenthèses, et vont de 0 à *taille*. Pour obtenir un tableau de taille n dont le premier indice est 1 on le déclare ainsi...Dim tMesValeurs(1 To n) as Integer...
- Exemple : `a = tMesValeurs(5) + 6; tMesValeurs(6) = a + a;`
- Attention à ne pas *sortir* des tableaux !
- Aucune vérification n'est faite à l'exécution (comportement indéfini) !

Tableaux en Visual Basic : exemple

```
Const tmax = 100

Sub tableau()

Dim n As Integer
Dim t(1 To tmax) As Double

Do
n = InputBox("Combien de valeurs voulez-vous saisir ?")
Loop Until n > 0 And n <= tmax

For i = 1 To n
t(i) = InputBox("Entrez la " & i & " ème valeur du tableau")
Next

For i = 1 To n
MsgBox ("t[" & i & "] = " & t(i))
Next

End Sub
```

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**

Lit *taille* prix et les stocke dans *tPrix*.

Lexique local des variables

i (entier) Indice de la boucle de lecture

Algorithme de lirePrix

pour *i* **de** 0 **à** *taille* − 1 **faire**
  tPrix[*i*] ← lire
fpour

Exemple

- Édition d'une facture correspondant à l'achat de produits en quantités données par l'utilisateur, parmi une liste de de *NB_PROD* produits de prix donnés et numérotés de 1 à *NB_PROD*.

Lexique des constantes

NB_PROD (entier) = 10 nombre de produits en vente

Lexique des fonctions

fonction lirePrix(**out** *tPrix* : tableau de réels, **in** *taille* : entier) : **vide**

Lexique des variables

tPrix	(tableau [*NB_PROD*] de réels)	liste des prix des produits	INTERMÉDIAIRE
total	(réel)	montant total de la facture	INTERMÉDIAIRE
numProd	(entier)	suite des numéros de produits achetés	INTERMÉDIAIRE
quantité	(entier)	suite des quantités de produits	INTERMÉDIAIRE

Exemple (suite)

Algorithme

```
// remplissage de la liste des prix des produits
lirePrix(tPrix, NB_PROD)
total ← 0

// lecture du premier numéro de produit
numProd ← lire
tant que numProd ≠ EOF faire
    si numProd ≥ 1 ∧ numProd ≤ NB_PROD alors
        quantité ← lire
        total ← total + quantité × tPrix[numProd − 1]
    fsi
    numProd ← lire
ftant
écrire "Le montant total est ", total, "€"
```

Exemple (suite)

Algorithme

```
// remplissage de la liste des prix des produits
lirePrix(tPrix, NB_PROD)
total ← 0

// lecture du premier numéro de produit
numProd ← lire
tant que numProd ≠ EOF faire
    si numProd ≥ 1 ∧ numProd ≤ NB_PROD alors
        quantité ← lire
        total ← total + quantité × tPrix[numProd − 1]
    fsi
    numProd ← lire
ftant
écrire "Le montant total est ", total, "€"
```

- Vérification de la validité du numéro.

Exemple (suite)

Algorithme

```
// remplissage de la liste des prix des produits
lirePrix(tPrix, NB_PROD)
total ← 0

// lecture du premier numéro de produit
numProd ← lire
tant que numProd ≠ EOF faire
    si numProd ≥ 1 ∧ numProd ≤ NB_PROD alors
        quantité ← lire
        total ← total + quantité × tPrix[numProd − 1]
    fsi
    numProd ← lire
ftant
écrire "Le montant total est ", total, "€"
```

- Décalage pour les indices du tableau.

Parcours en sens inverse

- On peut aussi parcourir un tableau de la fin vers le début.

Exemple

Impression d'une liste de valeurs lues, en ordre inverse.

Lexique des variables

taille	(entier)	taille du tableau	DONNÉE
tab	(tableau [*taille*] d'entiers)	un tableau...	INTERMÉDIAIRE
i	(entier)	compteur de boucle	INTERMÉDIAIRE

Algorithme

pour *i* **de** 0 **à** *taille* − 1 **faire**
  tab[*i*] ← lire
fpour
pour *i* **de** *taille* − 1 **à** 0 **en descendant faire**
  écrire *tab*[*i*]
fpour

Compter les voyelles dans un texte

Exemple

Données

/ (texte lu)

Résultat

Affiche le nombre d'occurrences des lettres '`a`', '`e`', '`i`', '`o`', '`u`' et '`y`' dans un texte.

Idée

Lire le texte caractère par caractère pour compter les occurrences des voyelles.

Lexique des constantes

NB_VOY	(entier)	$= 6$	nombre de voyelles dans l'alphabet

Lexique des variables

tNbVoy	(tableau [*NB_VOY*] d'entiers)	compteurs	INTERMÉDIAIRE
carLu	(caractère)	caractère du texte	INTERMÉDIAIRE
i	(entier)	indice de compteur	INTERMÉDIAIRE

Compter les voyelles dans un texte (suite)

Algorithme

Algorithme

$tNbVoy \leftarrow \{0,0,0,0,0,0\}$
$carLu \leftarrow$ lire
tant que $carLu \neq$ EOF **faire**
  selon que $carLu$ **est**
    cas 'a' : $i \leftarrow 0$
    cas 'e' : $i \leftarrow 1$
    cas 'i' : $i \leftarrow 2$
    cas 'o' : $i \leftarrow 3$
    cas 'u' : $i \leftarrow 4$
    cas 'y' : $i \leftarrow 5$
    défaut : $i \leftarrow -1$
  fselon
...

(suite)

...
  si $i \neq -1$ **alors**
    $tNbVoy[i] \leftarrow tNbVoy[i] + 1$
  fsi
  $carLu \leftarrow$ lire
ftant
écrire "Nombre de a:", $tNbVoy[0]$
écrire "Nombre de e:", $tNbVoy[1]$
écrire "Nombre de i:", $tNbVoy[2]$
écrire "Nombre de o:", $tNbVoy[3]$
écrire "Nombre de u:", $tNbVoy[4]$
écrire "Nombre de y:", $tNbVoy[5]$

Avec un tableau de constantes

Variante de l'exemple

Idée

Comme précédemment, mais en utilisant un tableau constant de caractères à la place de la structure *selon que*.

Lexique des constantes

NB_VOY	(entier)	$= 6$	nombre de voyelles dans l'alphabet
tVoy	(tableau [*NB_VOY*] de caractères)	$=$ {'a','e','i', 'o','u','y'}	liste des voyelles

Lexique des variables

tNbVoy	(tableau [*NB_VOY*] d'entiers)	compteurs	INTERMÉDIAIRE
carLu	(caractère)	caractère du texte	INTERMÉDIAIRE
i	(entier)	indice de compteur	INTERMÉDIAIRE

Avec un tableau de constantes (suite)

Algorithme

Algorithme

$tNbVoy \leftarrow \{0,0,0,0,0,0\}$
$carLu \leftarrow$ lire
tant que $carLu \neq$ EOF **faire**
  pour i **de** 0 **à** $NB\_VOY - 1$ **faire**
    si $carLu = tVoy[i]$ **alors**
      $tNbVoy[i] \leftarrow tNbVoy[i] + 1$
    fsi
  fpour
  $carLu \leftarrow$ lire
ftant
pour i **de** 0 **à** $NB\_VOY - 1$ **faire**
  écrire "`Nombre de `", $tVoy[i]$, "**:**", $tNbVoy[i]$
fpour

Annexe

Références

Arnaud GIERSCH
Algorithmique 1ère Année
Communication privée, 2012

Sylvain CONTASSOT-VIVIER
Algorithmique 1ère Année
Communication privée, 2007

Thomas H. CORMEN, Charles E. LEISERSON, Ronald L. RIVEST et Clifford STEIN
Introduction to Algorithms
The MIT Press, 2e édition, 2001

ISO/IEC JTC1/SC22
ISO/IEC 14882:2003(E) : Programming languages – C++
ISO/IEC, 2e édition, 2003

Bjarne STROUSTRUP
Le langage C++
CampusPress France, 3e édition, 1999