Algorithmique et Programmation

DUT Informatique 1er semestre

Arnaud GIERSCH

arnaud.giersch@univ-fcomte.fr

IUT de Belfort-Montbéliard

Département Informatique

2014–2015

Plan

Plan

Plan

Introduction

- L'informatique met en jeu des ordinateurs qui fonctionnent selon de 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, 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.
- 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.
- 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 $x \geqslant y$.
- 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.
- 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 $a \geqslant b$.
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 $b \leftarrow 6$

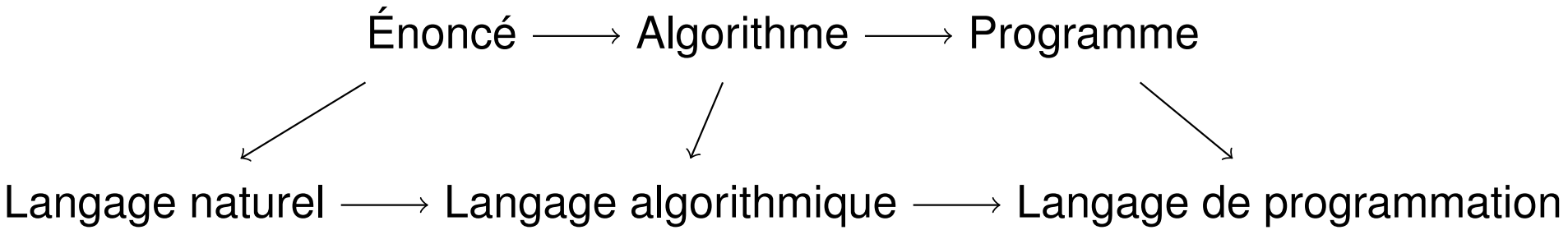
Étape 20 : $c \leftarrow 12 - 6 = 6$

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

Arrêt car $a = b$

le résultat est donc 6

Langages



- 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 de programmation

- 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 : 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.
- 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.
- 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 $x \geqslant y$

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 (1re décomposition).
- Synopsis :
  - Mot clef « **Idée** » suivi de l'énumération des étapes.

Exemple : calcul de la superficie d'un champ

Idée

(i) acquérir la longueur et la largeur du champ ;

(ii) calculer la superficie ;

(iii) 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.
- 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 :

  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

- 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

- 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.
- Synopsis :
  - Mot clef « **Algorithme** » suivi de la suite des actions élémentaires.

Exemple : superficie d'un champ

Algorithme

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

Explications

- « lire » signifie que la valeur est donnée par l'utilisateur (ex. : au clavier).
- « écrire » affiche le texte et/ou les valeurs qui suivent à l'écran.
- L'*affectation* est indiquée par « ← » :
  - affecte la valeur résultant de l'évaluation de l'expression à droite de la flèche dans la variable placée à gauche ;
  - « *truc* ← *machin* » peut se lire « la variable *truc* reçoit la valeur *machin* » ;
  - attention 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* ← 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	40	*largeur*	?	*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*	?

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

(i) acquérir la longueur et la largeur du champ ;
(ii) calculer la superficie ;
(iii) 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 ← lire
largeur ← lire
surface ← *longueur* × *largeur*
écrire *surface*

Exemple : en Java...

Langage de programmation utilisé : Java

```
import java.util.*;

class Champ {
    static final Scanner input = new Scanner(System.in);

    public static void main(String[] args) {
        double longueur;
        double largeur;
        double surface;

        longueur = input.nextDouble();
        largeur = input.nextDouble();
        surface = longueur * largeur;
        System.out.println(surface);
    }
}
```

Exemple : en Java...

Langage de programmation utilisé : Java

```
import java.util.*;

class Champ {
    static final Scanner input = new Scanner(System.in);

    public static void main(String[] args) {
        double longueur;
        double largeur;
        double surface;

        System.out.print("Longueur du champ ? ");
        longueur = input.nextDouble();
        System.out.print("Largeur du champ ? ");
        largeur = input.nextDouble();
        surface = longueur * largeur;
        System.out.println("La superficie du champ est : " + surface);
    }
}
```

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 de comparaison : $\leqslant$, $\geqslant$, $<$, $>$, $=$, $\neq$
- 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
- 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 Java.

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

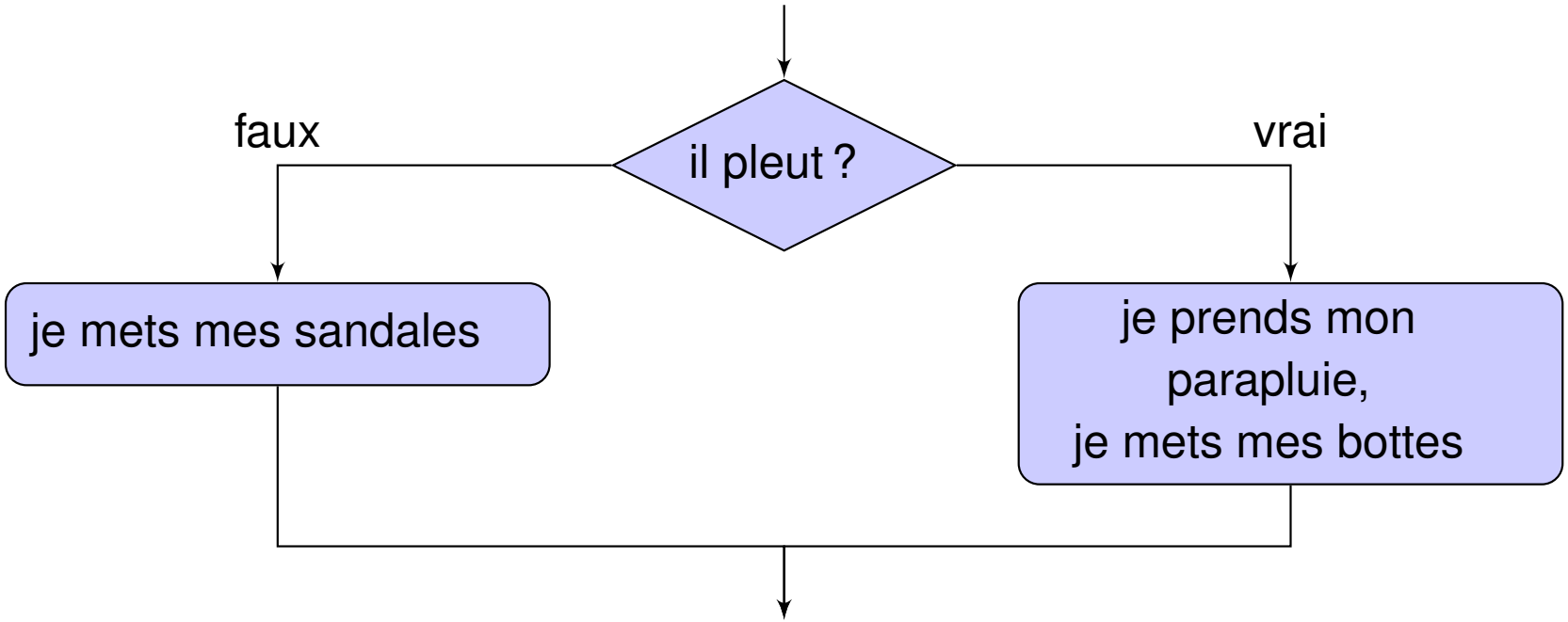
Lexique des constantes

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*.

Exemples

Exemple 1 : un exemple simple

Lexique des variables

âge (entier) âge de l'utilisateur INTERMÉDIAIRE

Algorithme

écrire "`Quel est votre âge ?`"
âge ← lire
si *âge* $<$ 18 **alors**
  écrire "`Vous êtes mineur.`"
sinon
  écrire "`Vous êtes majeur.`"
fsi

Exemples

Exemple 2 : mettre des valeurs dans l'ordre

Lexique des variables

a	(entier)	un entier	DONNÉE
b	(entier)	un autre entier	DONNÉE
tmp	(entier)	une variable temporaire	INTERMÉDIAIRE

Algorithme

si $a < b$ **alors**
  `// quand` $a < b$`, on échange les valeurs des variables`
  $tmp \leftarrow a$
  $a \leftarrow b$
  $b \leftarrow tmp$
fsi
`// ici, on a toujours` $a \geqslant b$
...

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.

Enchaînement de « si... alors... sinon... »

- Lorsque plusieurs structures « si... alors... sinon... » s'enchaînent comme ceci...

si $\langle condition_1 \rangle$ **alors**
| ...
sinon
| **si** $\langle condition_2 \rangle$ **alors**
| | ...
| **sinon**
| | ...
| | **si** $\langle condition_n \rangle$ **alors**
| | | ...
| | **sinon**
| | | ...
| | **fsi**
| **fsi**
fsi

... alors on s'autorisera une notation plus compacte :

si $\langle condition_1 \rangle$ **alors**
| ...
sinon si $\langle condition_2 \rangle$ **alors**
| ...
...
sinon si $\langle condition_n \rangle$ **alors**
| ...
sinon
| ...
fsi

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.

Exemple : jours de la semaine (si... alors...)

Algorithme

val ← lire
si *val* = 1 **alors**
  écrire "C'est un lundi"
sinon si *val* = 2 **alors**
  écrire "C'est un mardi"
sinon si *val* = 3 **alors**
  écrire "C'est un mercredi"
sinon si *val* = 4 **alors**
  écrire "C'est un jeudi"
sinon si *val* = 5 **alors**
  écrire "C'est un vendredi"
sinon si *val* = 6 **alors**
  écrire "C'est un samedi"
sinon si *val* = 7 **alors**
  écrire "C'est un dimanche"
sinon
  écrire "valeur invalide:", *val*
fsi

Exemple : jours de la semaine (selon que)

Algorithme

val ← lire
selon que *val* **est**
cas *1* **:** écrire "`C'est un lundi`"
cas *2* **:** écrire "`C'est un mardi`"
cas *3* **:** écrire "`C'est un mercredi`"
cas *4* **:** écrire "`C'est un jeudi`"
cas *5* **:** écrire "`C'est un vendredi`"
cas *6* **:** écrire "`C'est un samedi`"
cas *7* **:** écrire "`C'est un dimanche`"
défaut : écrire "`valeur invalide:`", *val*
fselon

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.

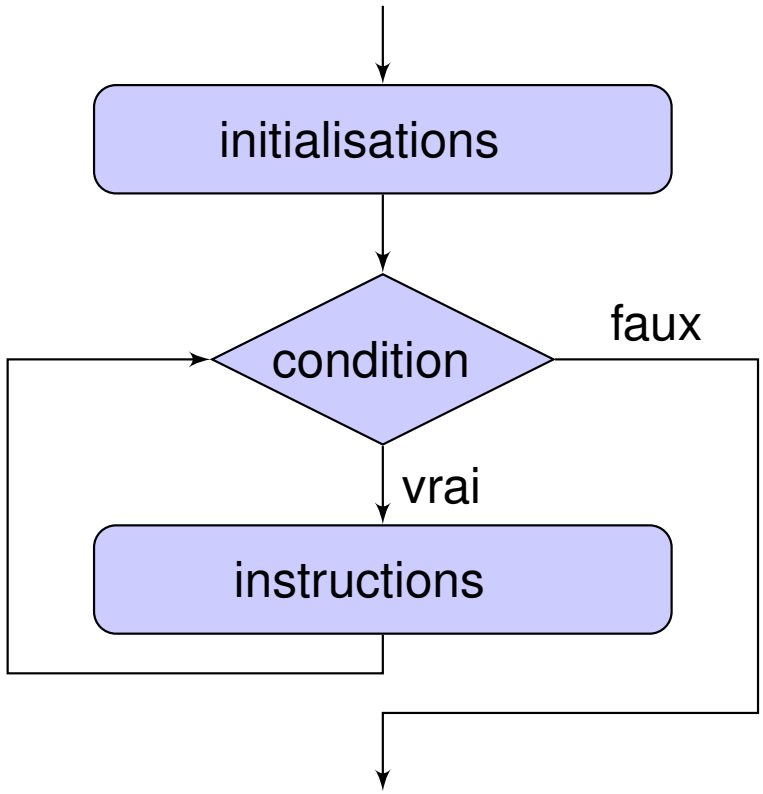
  pour :
  - répète des instructions un nombre *connu* de fois ;
  - la condition porte uniquement sur le nombre d'itérations à effectuer.

  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 !

Exemples

Exemple 2 : conversion de °F en °C

Algorithme

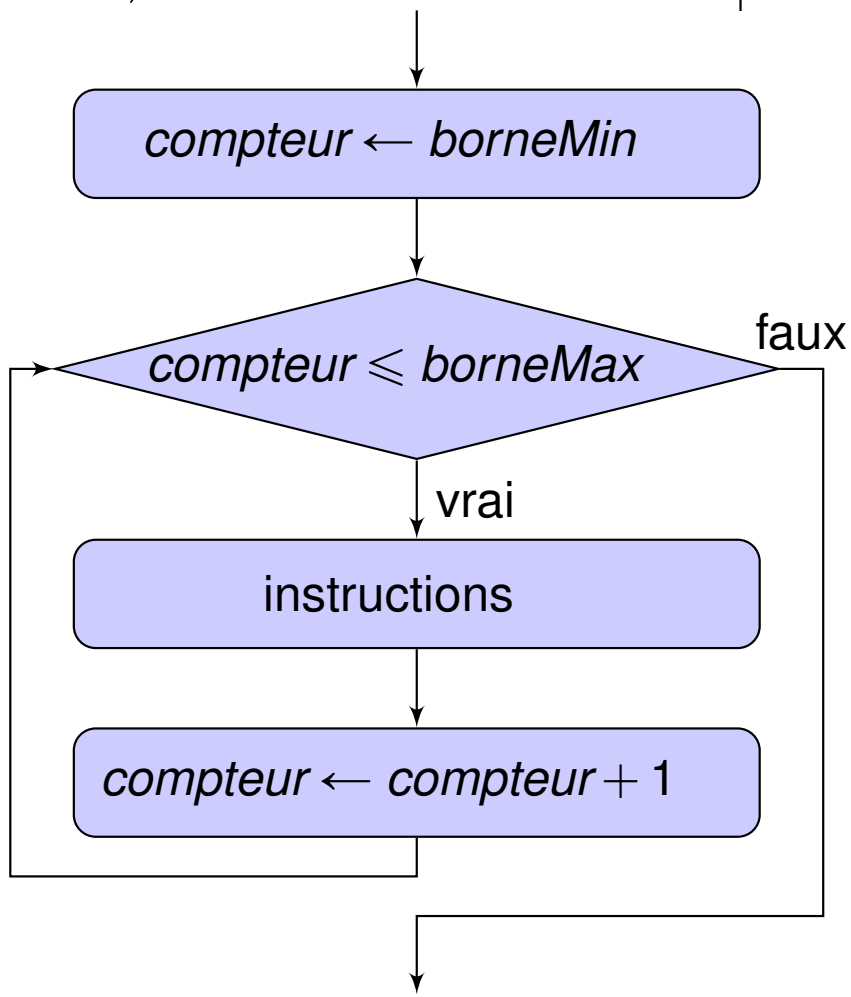
nbTemp ← lire
i ← 1 // initialisation de la boucle
tant que *i* ⩽ *nbTemp* **faire** // condition d'exécution de la boucle
  fahr ← lire
  celsius ← (*fahr* − 32)/9
  écrire "La température ", *fahr*, " en °F est ", *celsius*, " en °C"
  // instruction de modification éventuelle de la condition
  i ←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

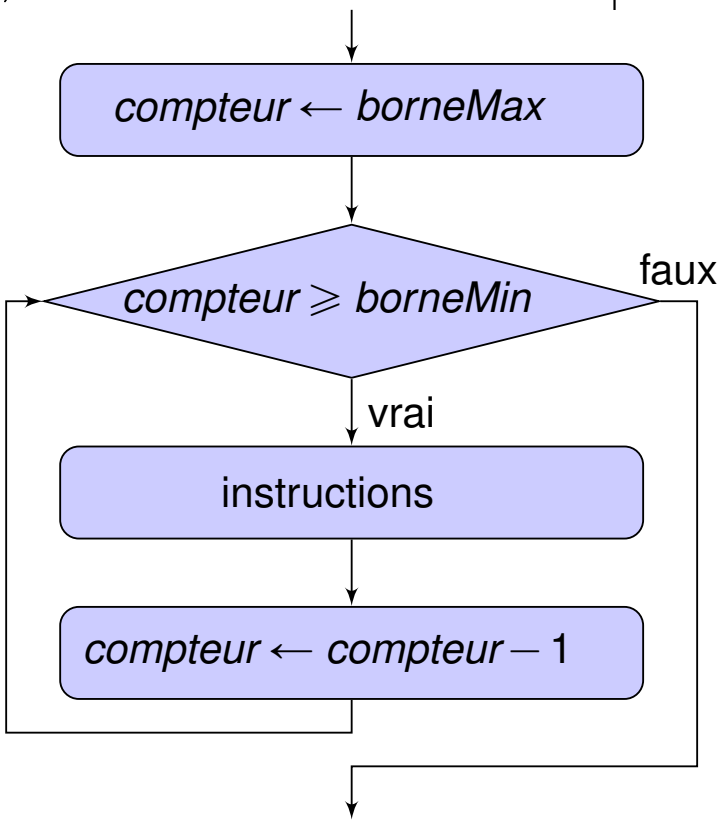
$n \leftarrow$ lire
$fact \leftarrow 1$
pour i **de** 1 **à** n **faire**
    $fact \leftarrow fact \times i$
    écrire $fact$
fpour

- 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.

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

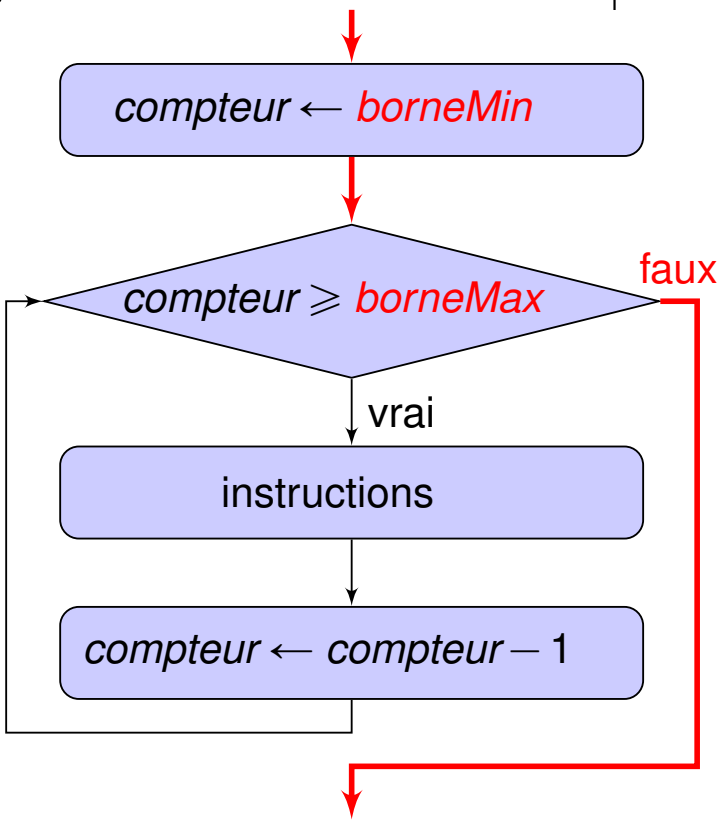


- On n'entre pas dans la boucle si ses bornes sont dans l'ordre inverse de son sens !

Sens du pour

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

pour ⟨*compteur*⟩ **de** ⟨*borneMin*⟩ **à** ⟨*borneMax*⟩ **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

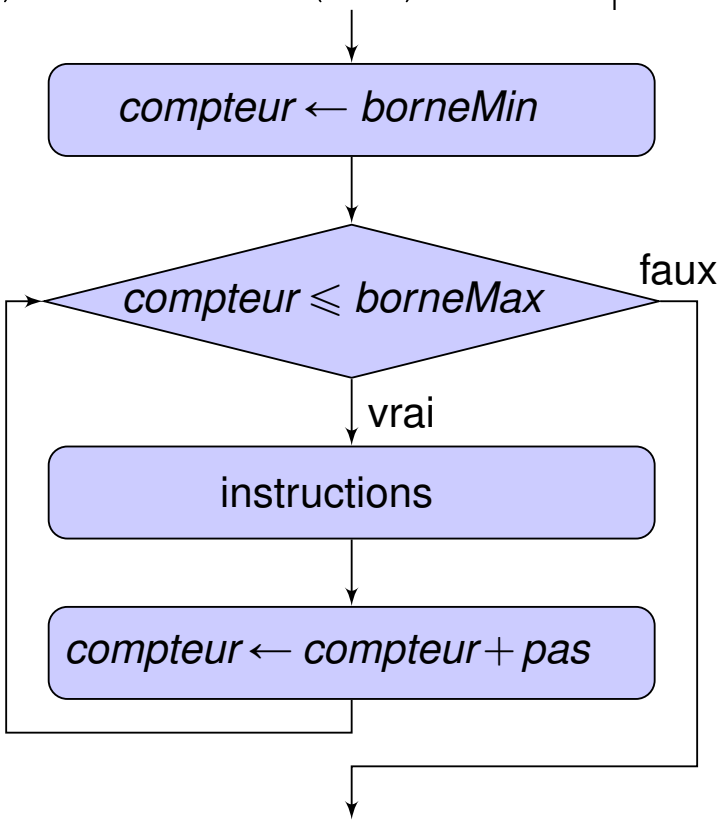
Algorithme

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

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

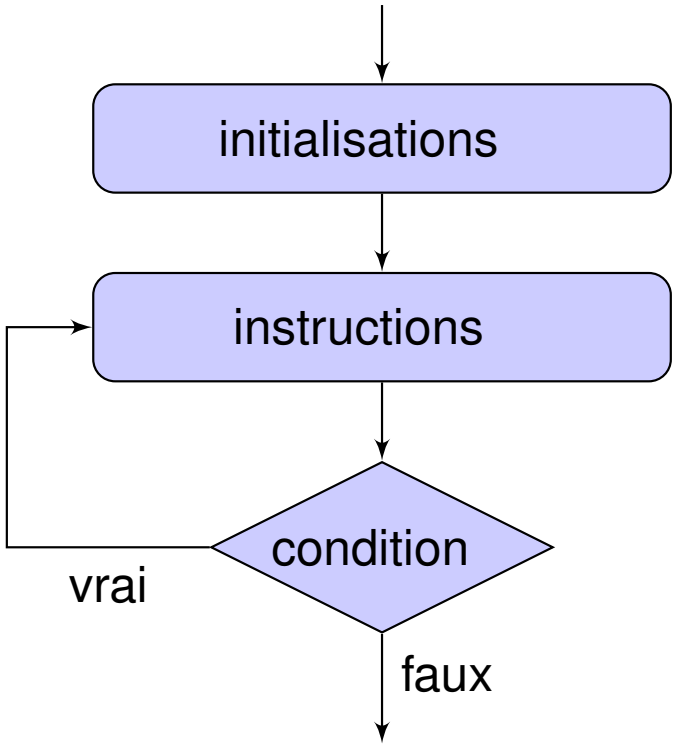
Algorithme

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 ← lire
tant que $\neg(1 \leqslant nombre \wedge nombre \leqslant 100)$
// ici, $1 \leqslant nombre \leqslant 100$

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
```

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
```

- 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 :

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

- 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, $C_n^0 = 1$, et que...

Exemple : triangle de Pascal

Recherche d'idée (suite)

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

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

- 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

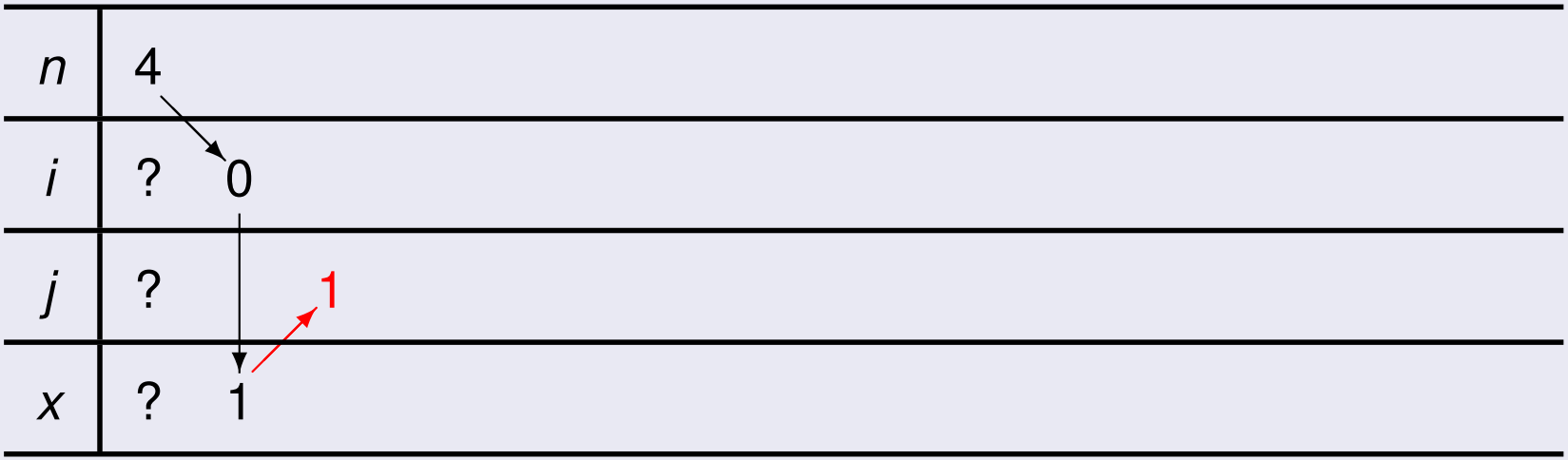
1_

Algorithme

```
pour i de 0 à n − 1 faire
    x ← 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

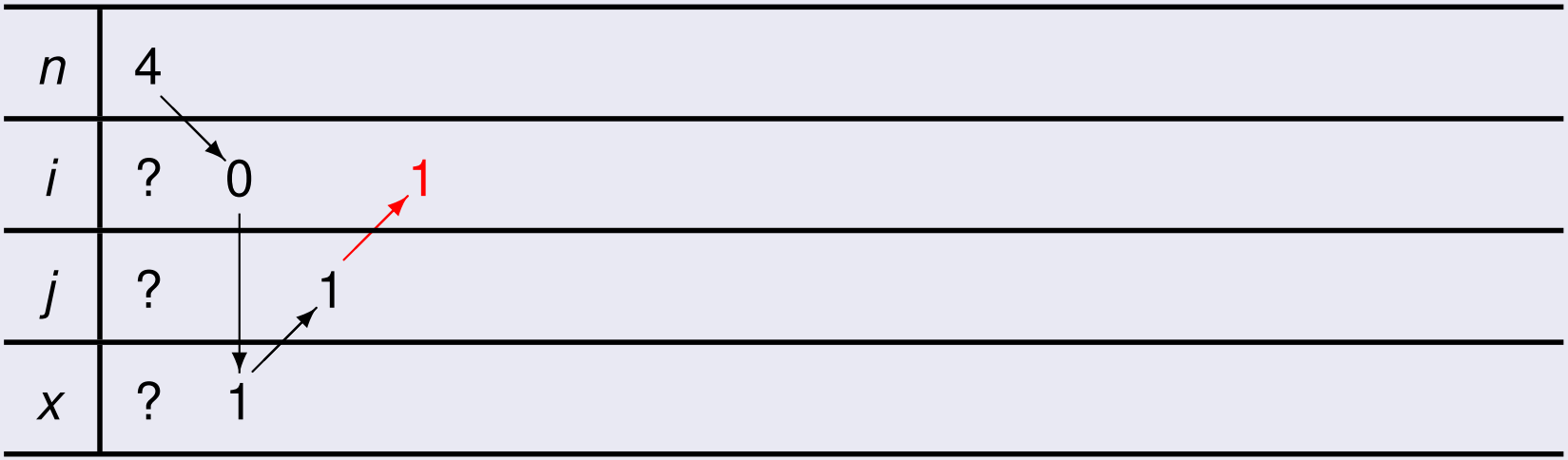
```
1

_
```

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

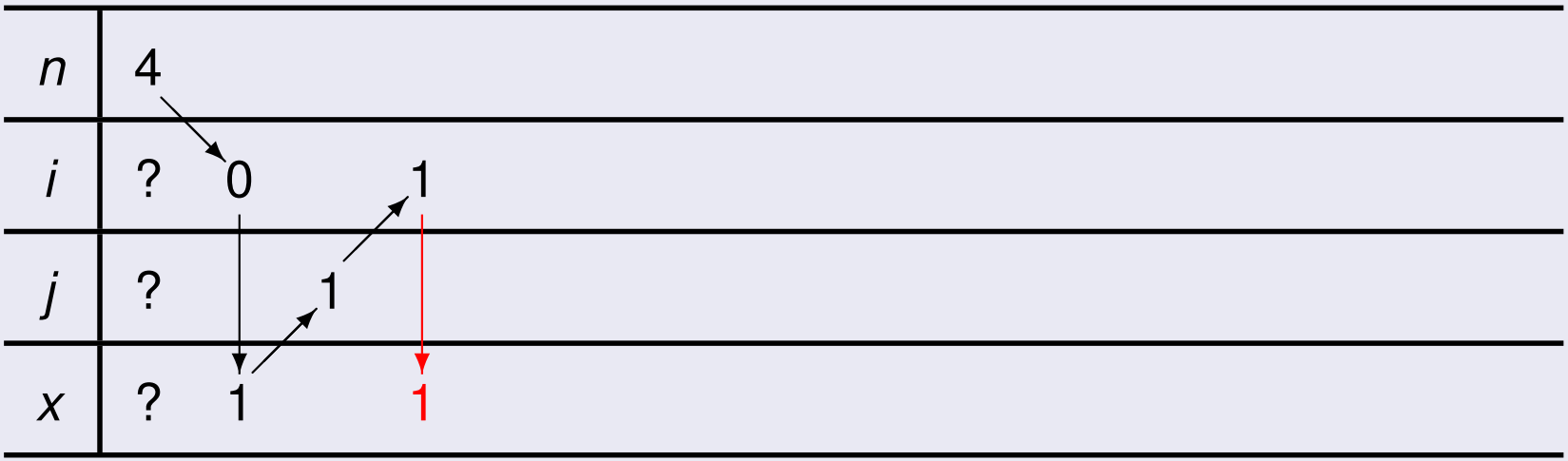
```
1

_
```

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

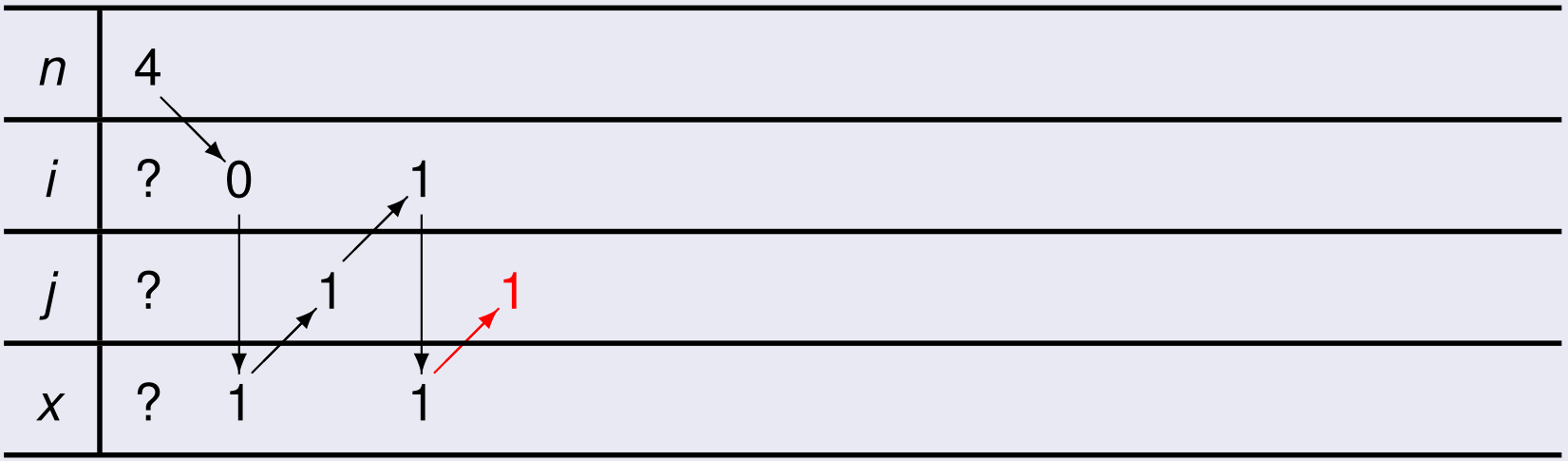
1_

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

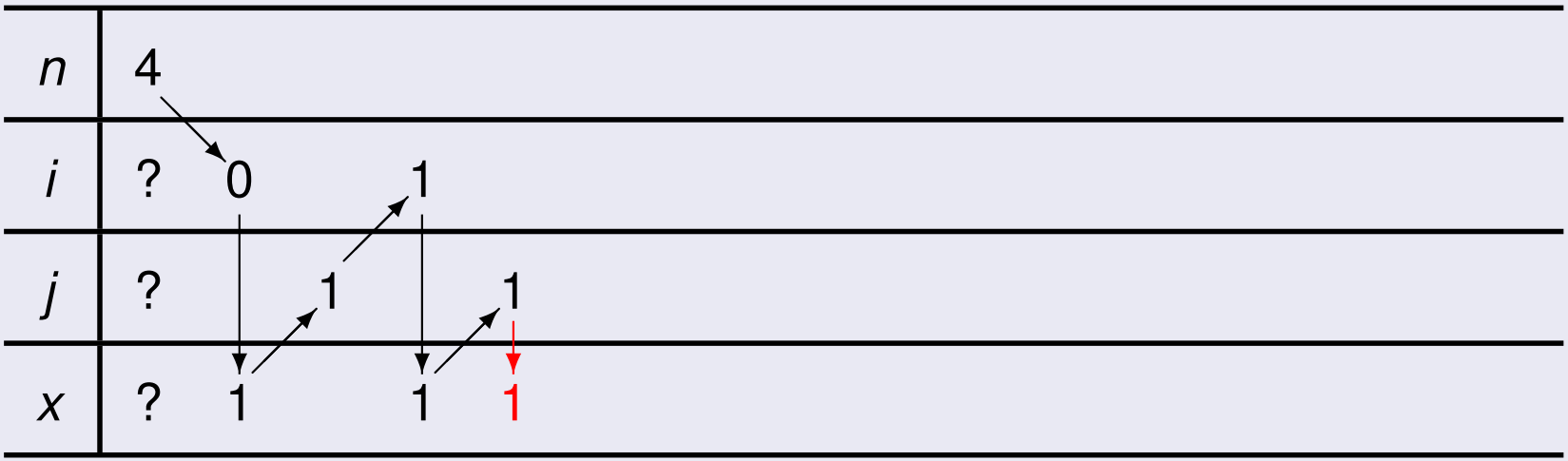
```
1
1_
```

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

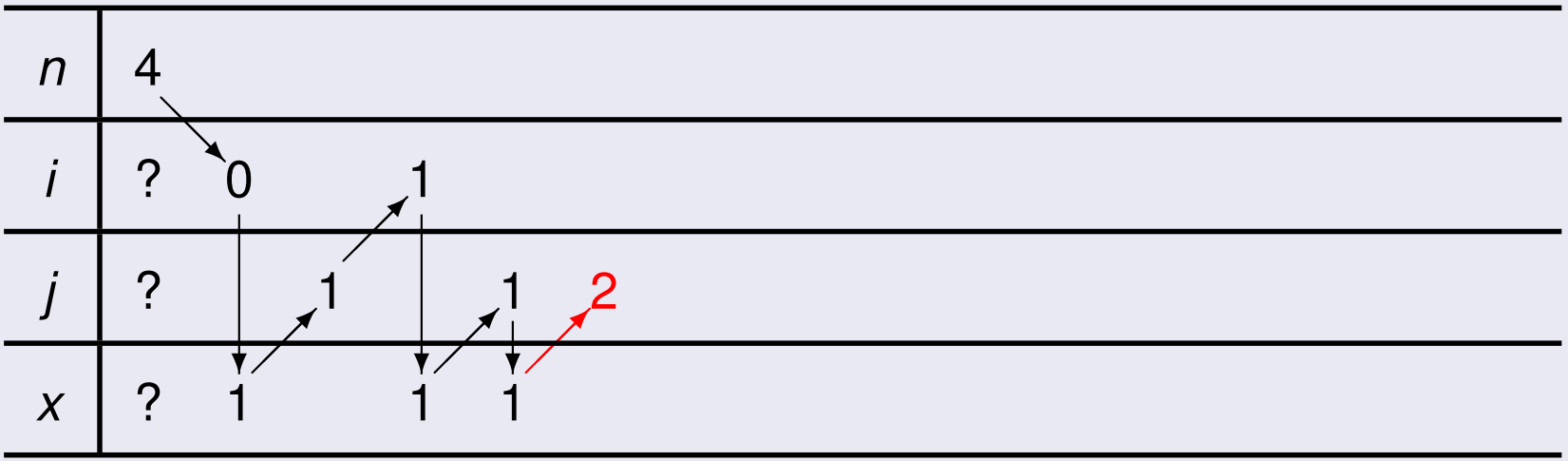
```
1
1 1
```

Algorithme

```
pour i de 0 à n − 1 faire
    x ← 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

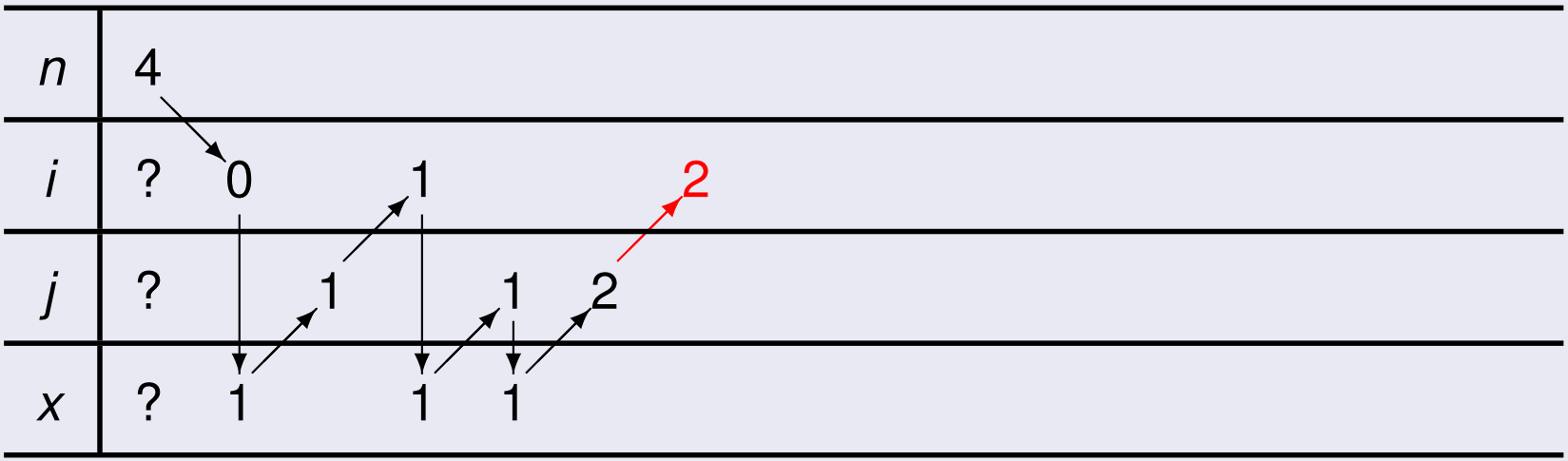
```
1
1 1
_
```

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

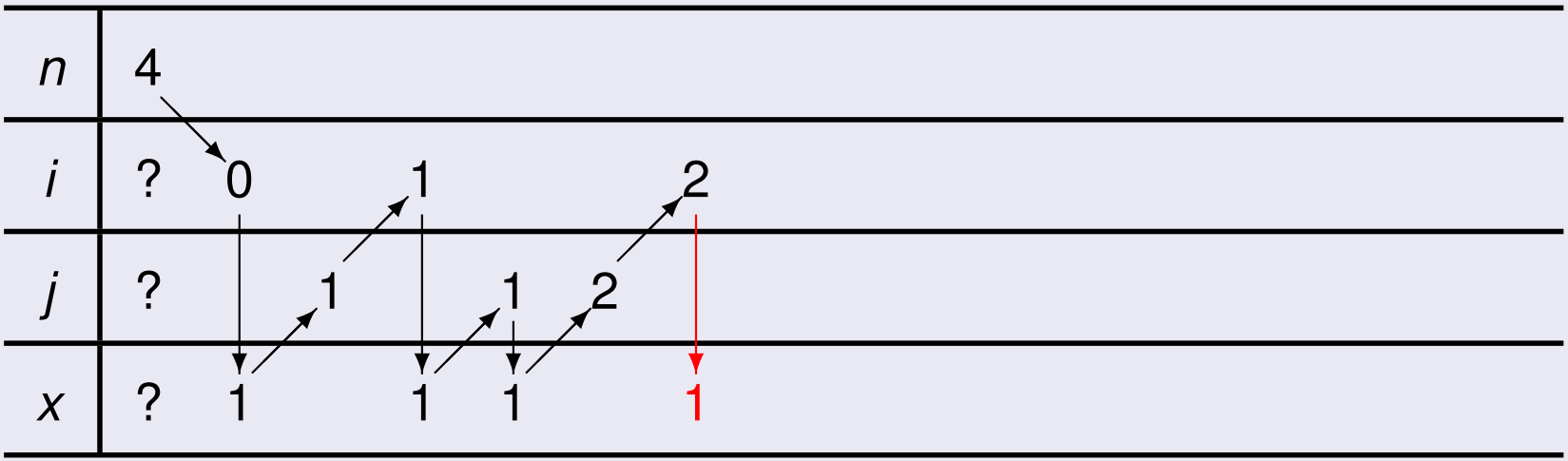
```
1
1 1
_
```

Algorithme

```
pour i de 0 à n − 1 faire
    x ← 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

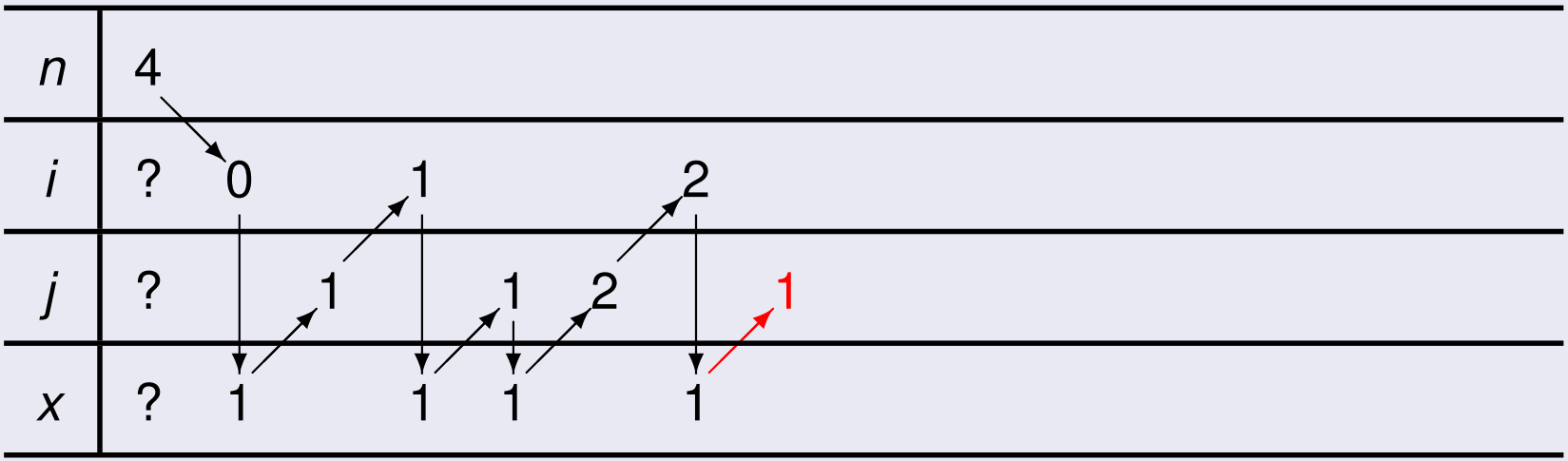
```
1
1 1
1
```

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

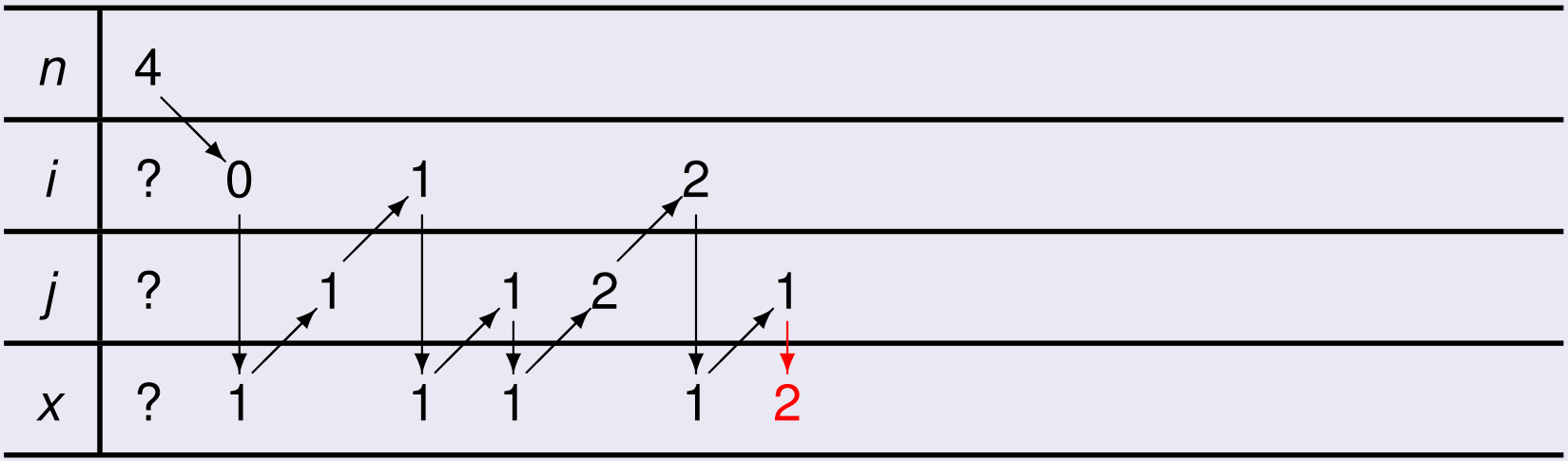
```
1
1 1
1
```

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

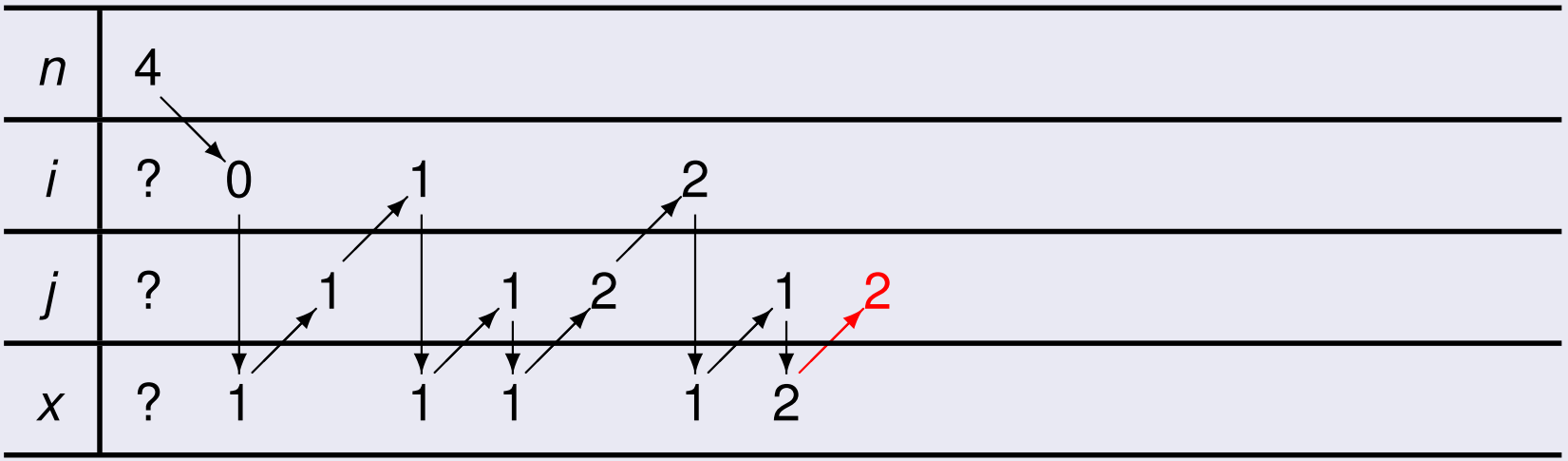
```
1
1 1
1 2
```

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

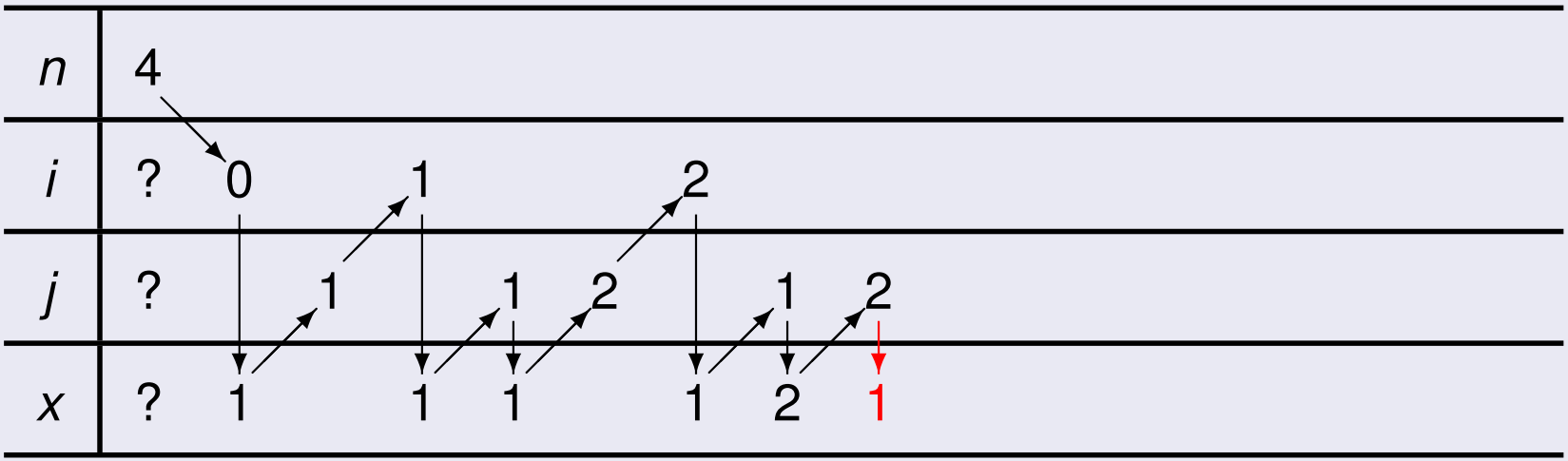
```
1
1 1
1 2
```

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

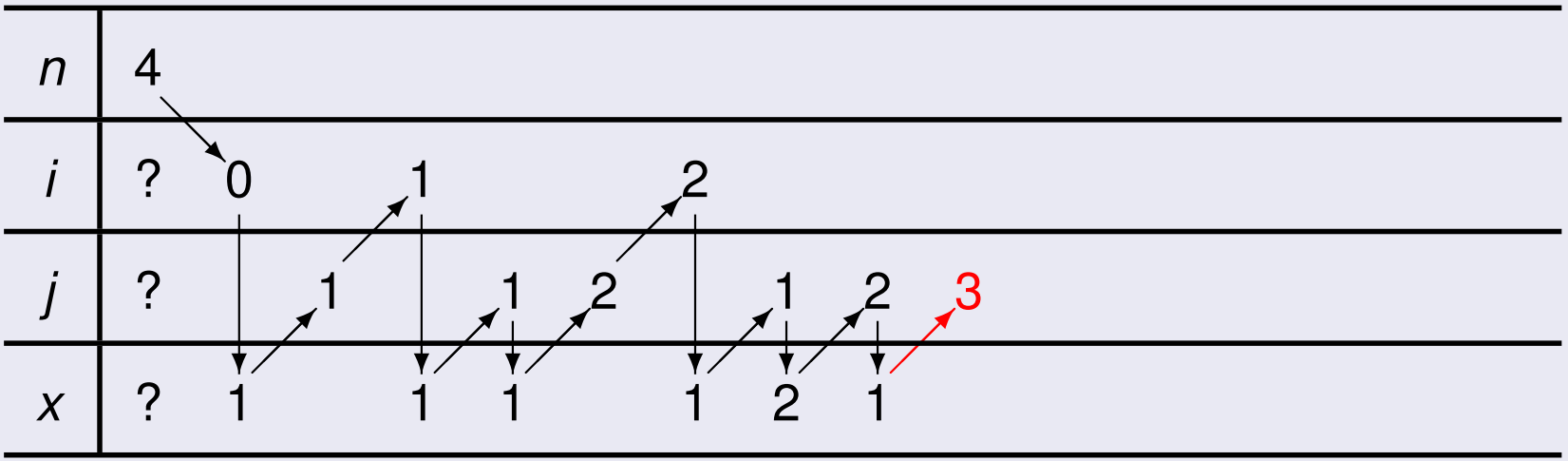
```
1
1 1
1 2 1
```

Algorithme

```
pour i de 0 à n − 1 faire
    x ← 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

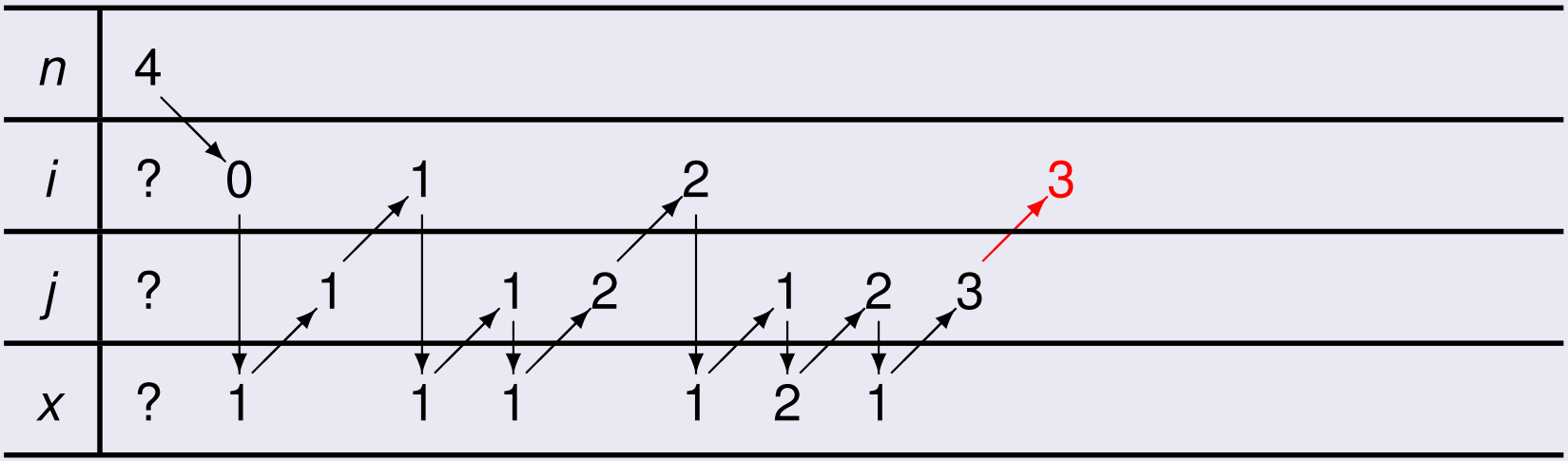
```
1
1 1
1 2 1
_
```

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

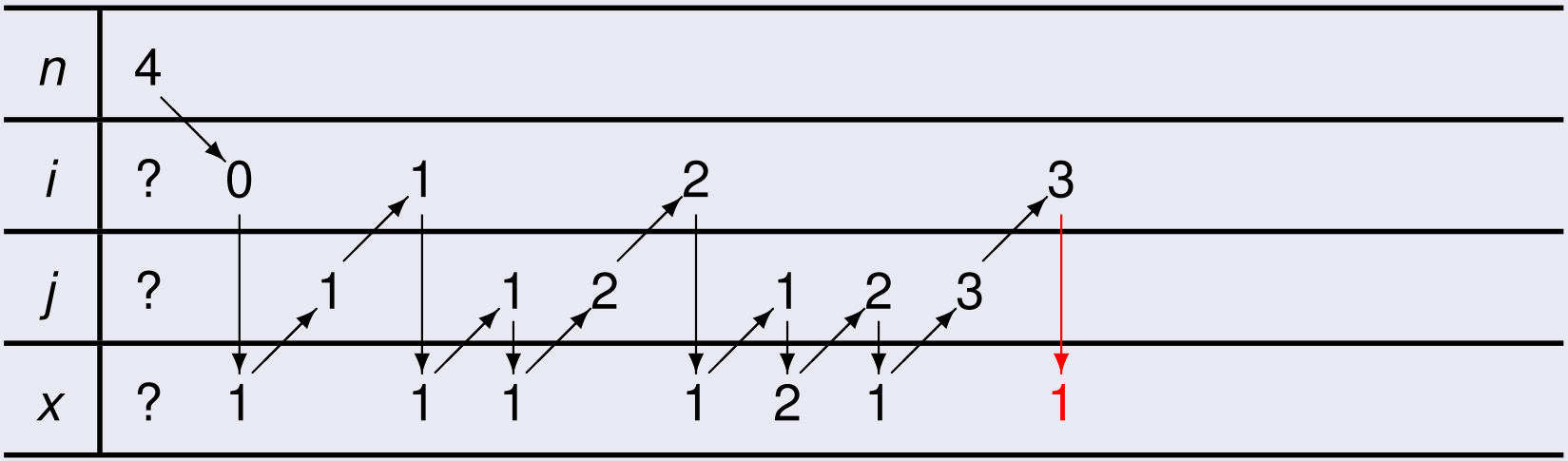
```
1
1 1
1 2 1
```

Algorithme

```
pour i de 0 à n − 1 faire
    x ← 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

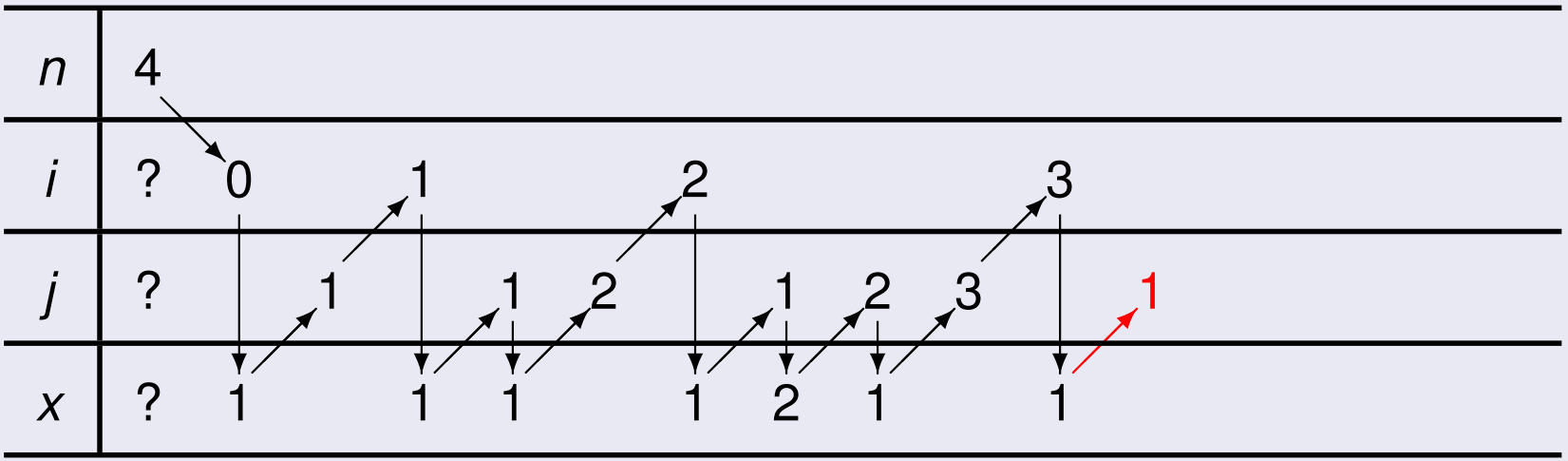
```
1
1 1
1 2 1
1
```

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

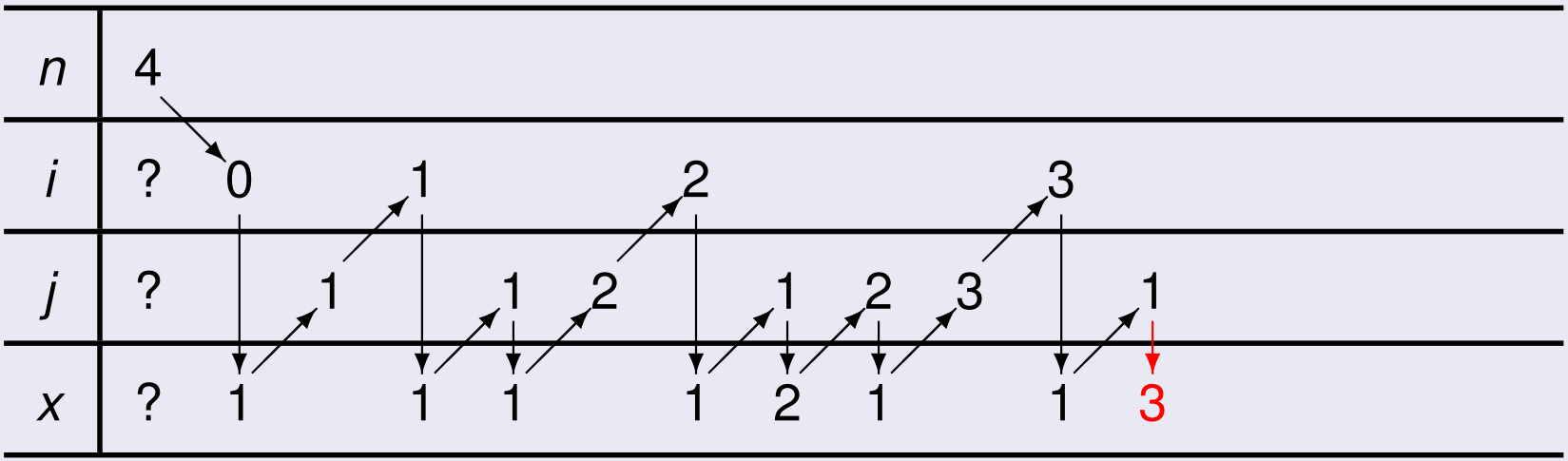
```
1
1 1
1 2 1
1
```

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

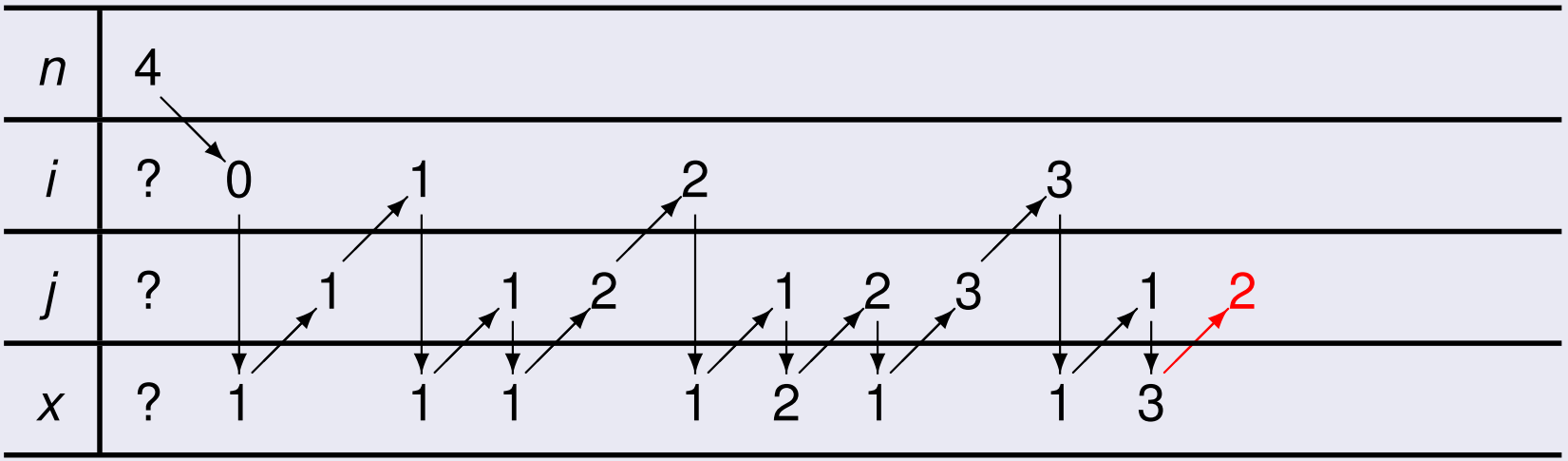
```
1
1 1
1 2 1
1 3
```

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

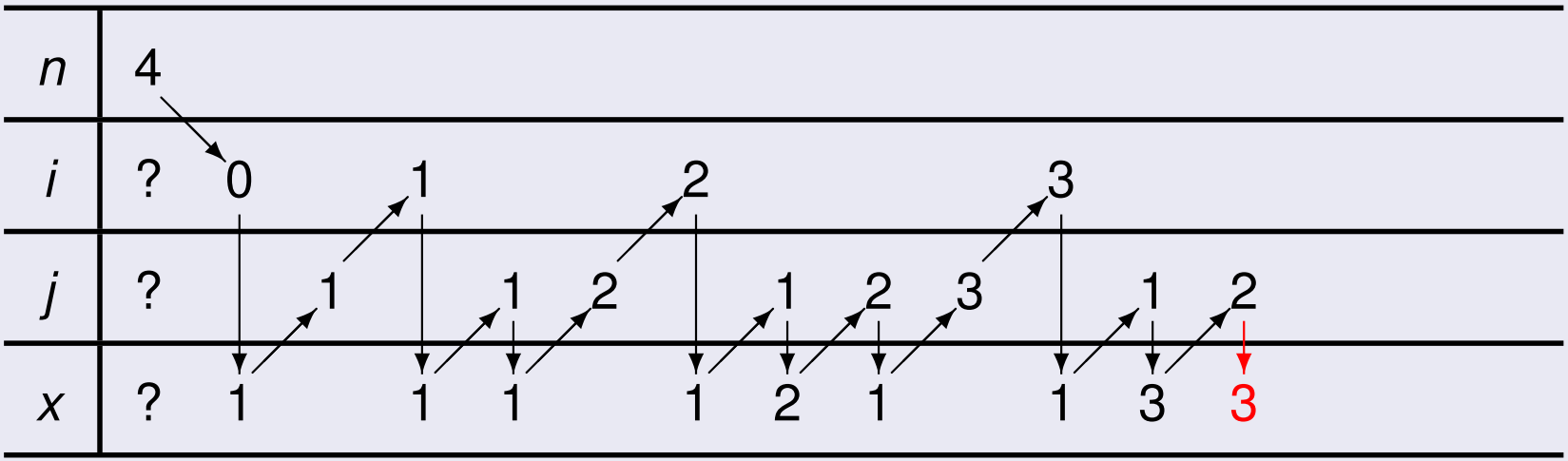
```
1
1 1
1 2 1
1 3
```

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

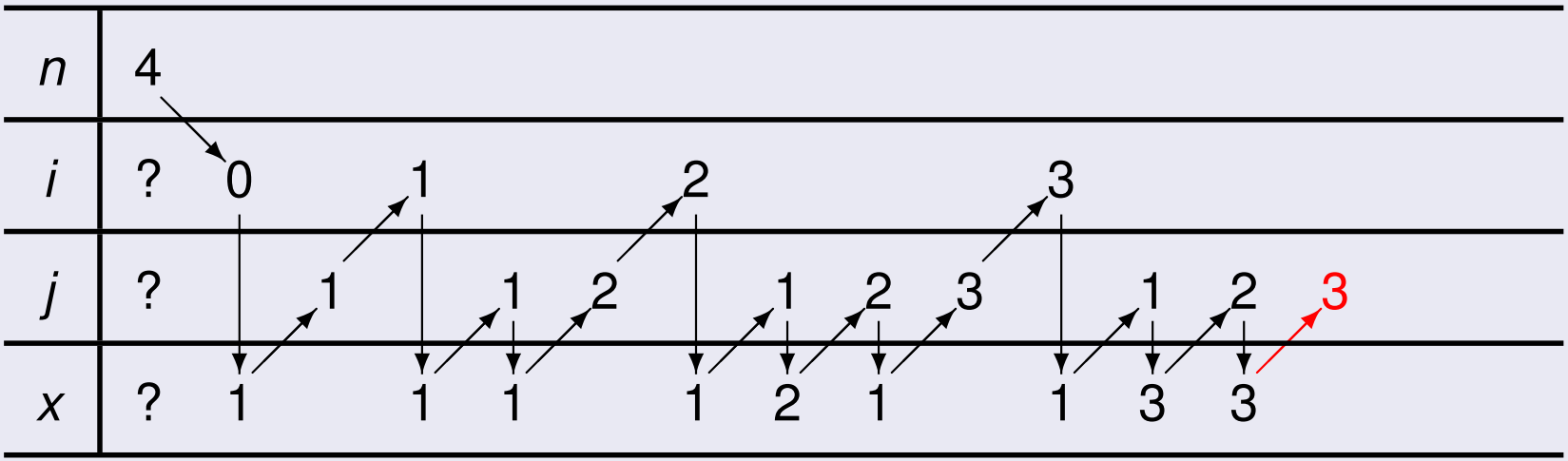
```
1
1 1
1 2 1
1 3 3
```

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

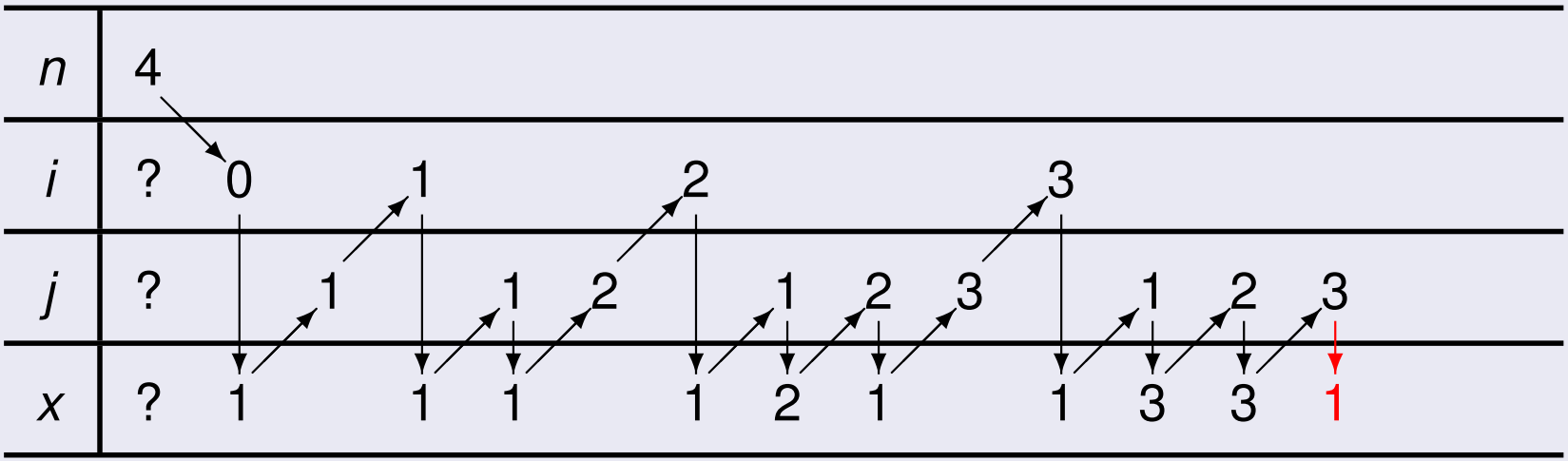
```
1
1 1
1 2 1
1 3 3
```

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

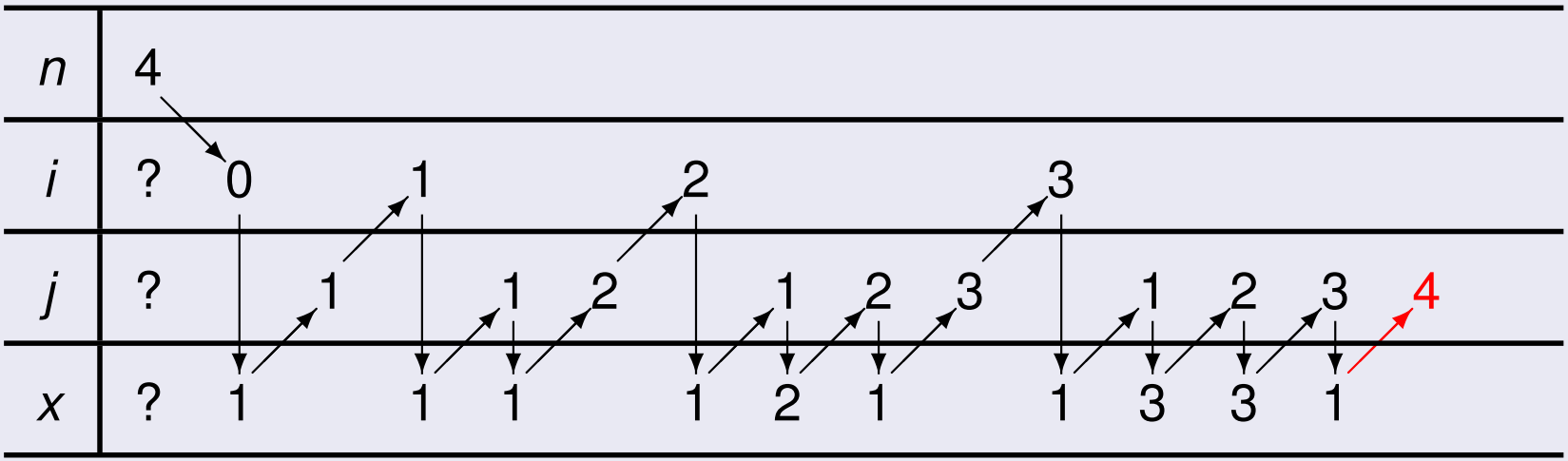
```
1
1 1
1 2 1
1 3 3 1
```

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

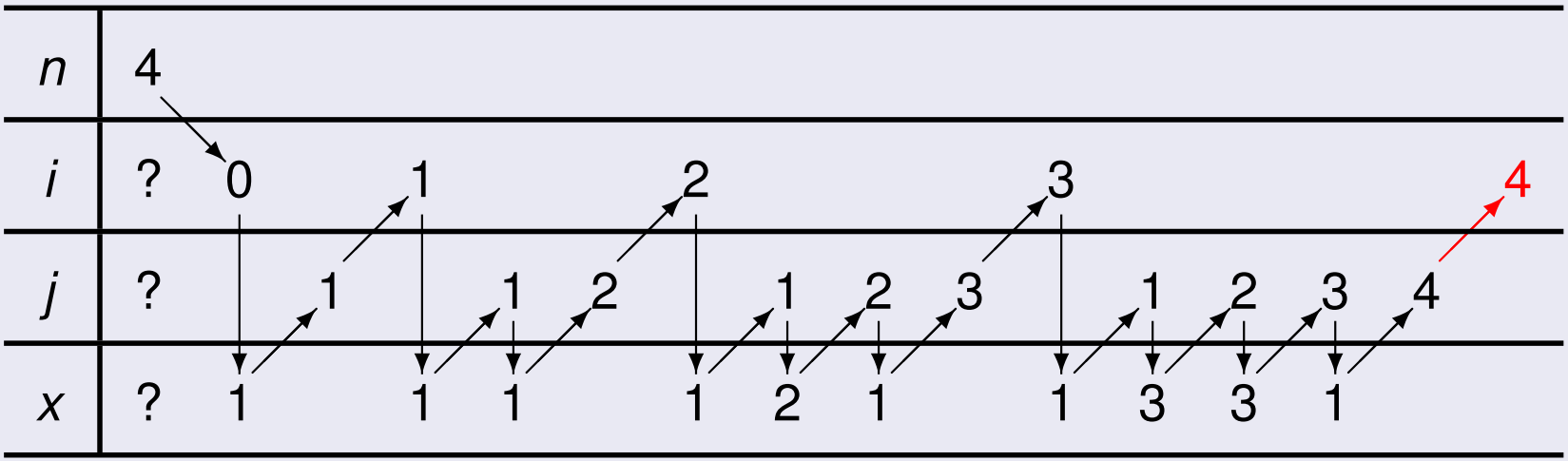
```
1
1 1
1 2 1
1 3 3 1
```

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

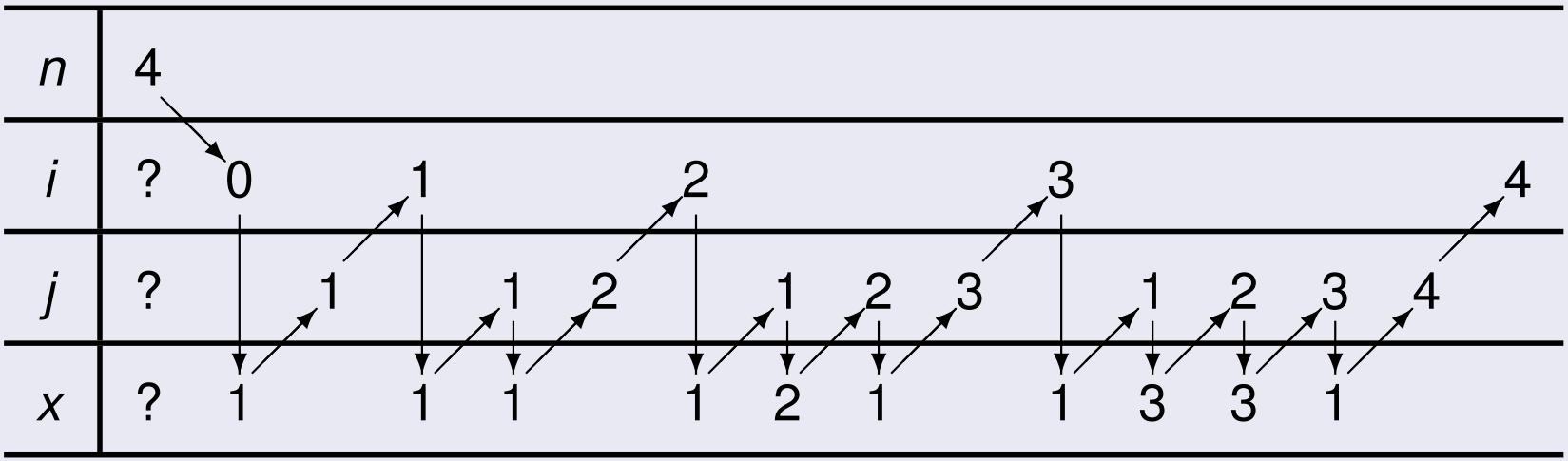
```
1
1 1
1 2 1
1 3 3 1

```

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

```
class Pascal {

    public static void main(String[] args) {
        /*--- déclaration des variables ---*/
        int n;
        int i;
        int j;
        int x;

        /*--- on renseigne les données ---*/
        n = 8;

        /*--- début de l'algorithme ---*/
        for (i = 0 ; i < n ; i++) {
            x = 1;
            System.out.print(x);
            for (j = 1 ; j <= i ; j++) {
                x = x * (i - j + 1) / j;
                System.out.print(" " + x);
            }
            System.out.println ();
        }
        /*--- fin de l'algorithme ---*/
    }
}
```

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.

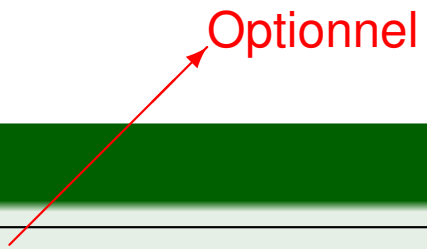
nomParam identifiant du paramètre dans la fonction

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.
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

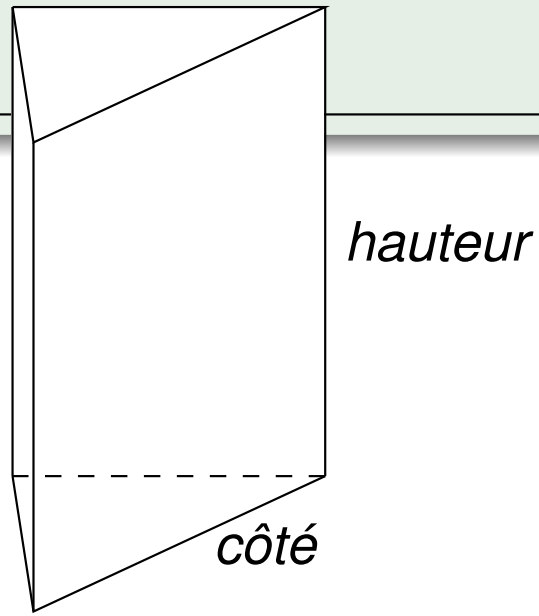
hauteurTri	(réel)	hauteur du triangle formant la base
surfTri	(réel)	surface du triangle formant la base

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	2	*i*	2	*côtéPri*	30	*hauteurPri*	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**

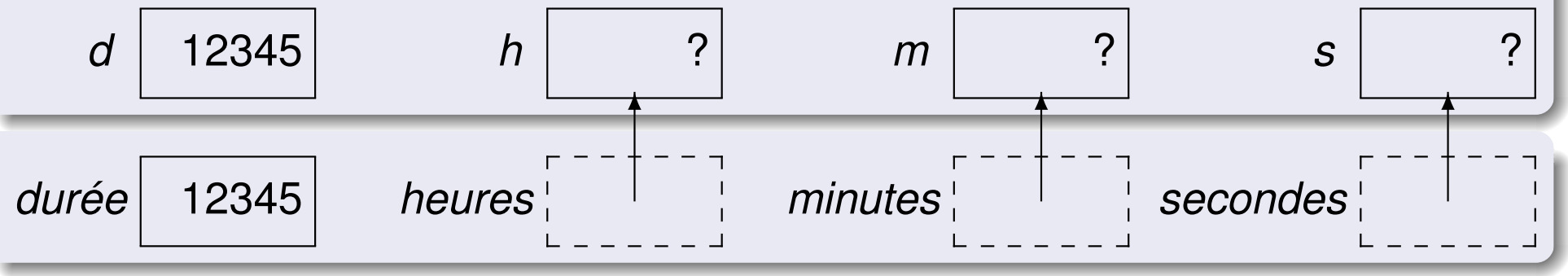
Algorithme principal

$d \leftarrow 12345$
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**

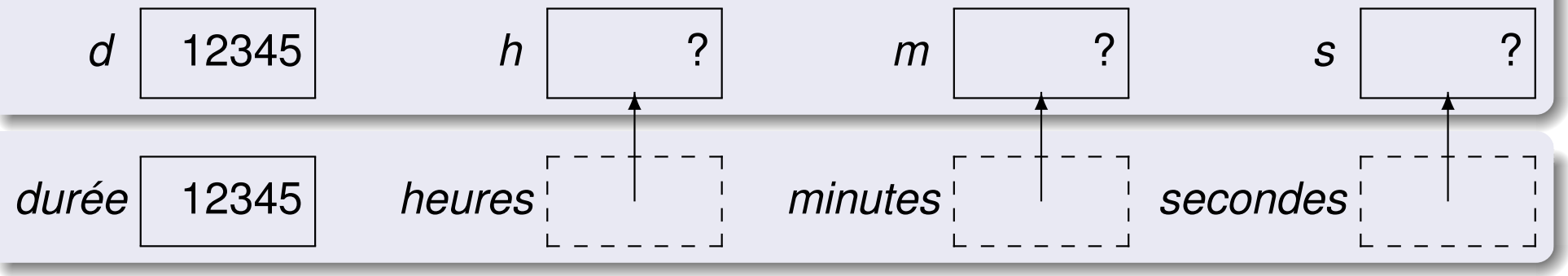
Algorithme principal

$d \leftarrow 12345$
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**

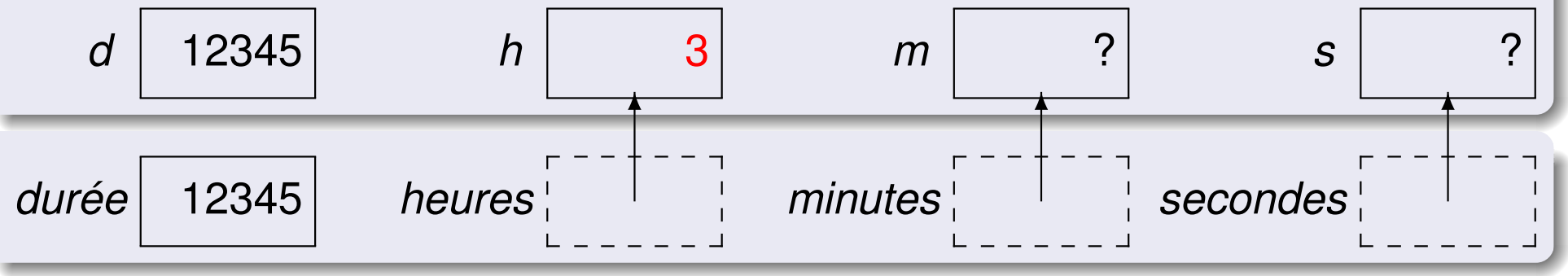
Algorithme de hms

heures ← *durée*/3600
minutes ← (*durée* mod 3600)/60
secondes ← *durée* mod 60

Sorties

Déroulement de l'exemple

Variables



---

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

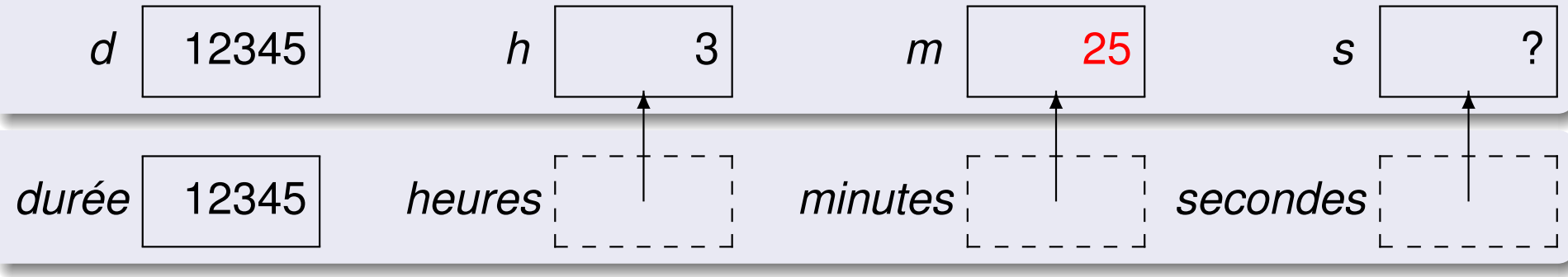
Algorithme de hms

heures ← *durée*/3600
minutes ← (*durée* mod 3600)/60
secondes ← *durée* mod 60

Sorties

Déroulement de l'exemple

Variables



---

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

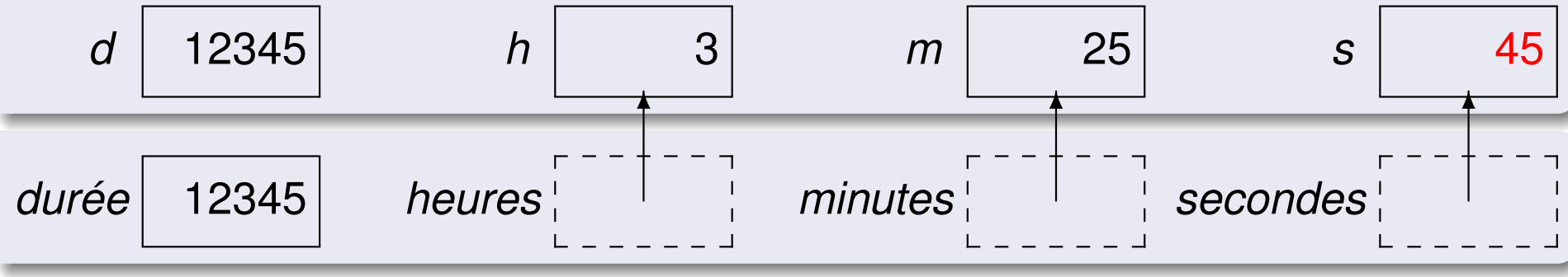
Algorithme de hms

heures ← *durée*/3600
minutes ← (*durée* mod 3600)/60
secondes ← *durée* mod 60

Sorties

Déroulement de l'exemple

Variables



---

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

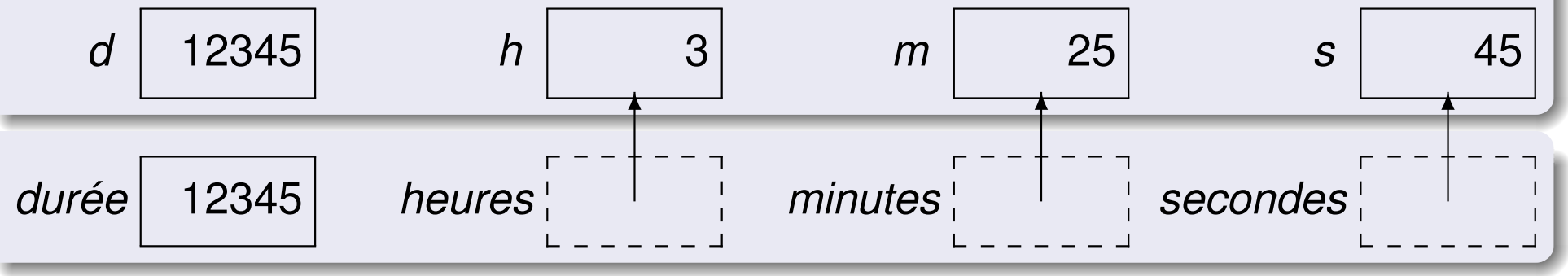
Algorithme de hms

heures ← *durée*/3600
minutes ← (*durée* mod 3600)/60
secondes ← *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 ← *durée*/3600
minutes ← (*durée* mod 3600)/60
secondes ← *durée* mod 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

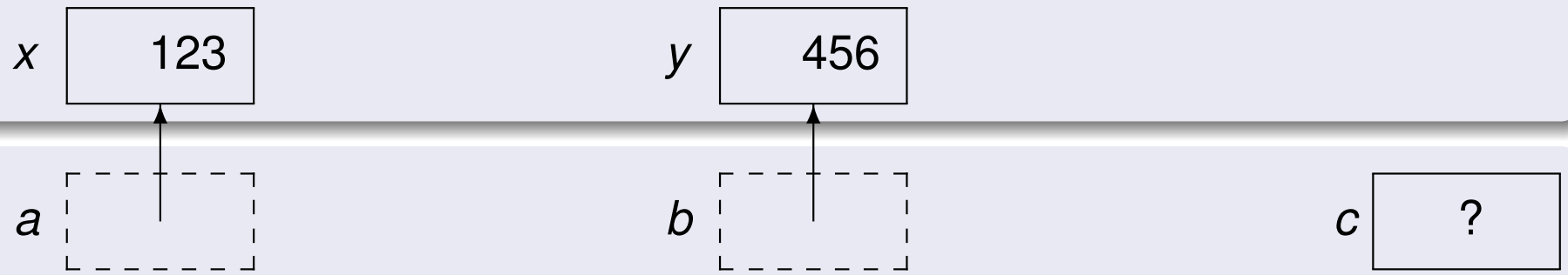
$x \leftarrow 123$
$y \leftarrow 456$
écrire "`avant : x =`", x, "`et y =`", y
échanger(x, y)
écrire "`après : x =`", x, "`et y =`", y

Sorties

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

Déroulement de l'exemple

Variables



Algorithme de échanger

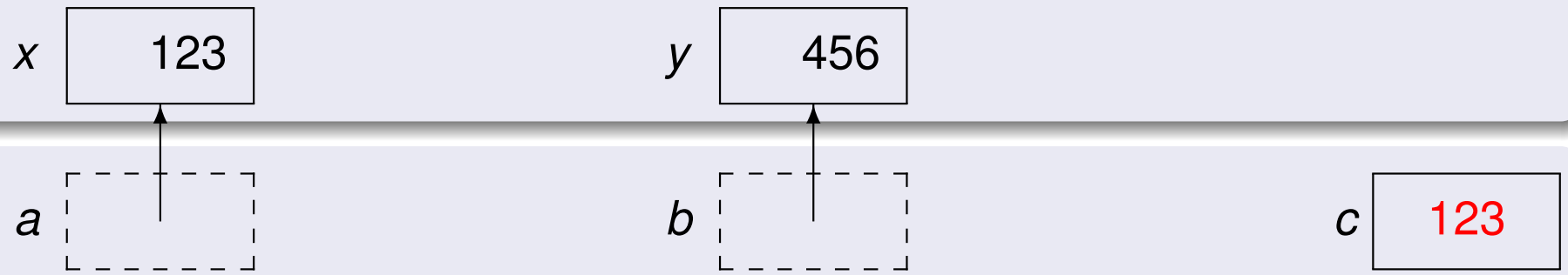
$c \leftarrow a$
$a \leftarrow b$
$b \leftarrow c$

Sorties

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

Déroulement de l'exemple

Variables



Algorithme de échanger

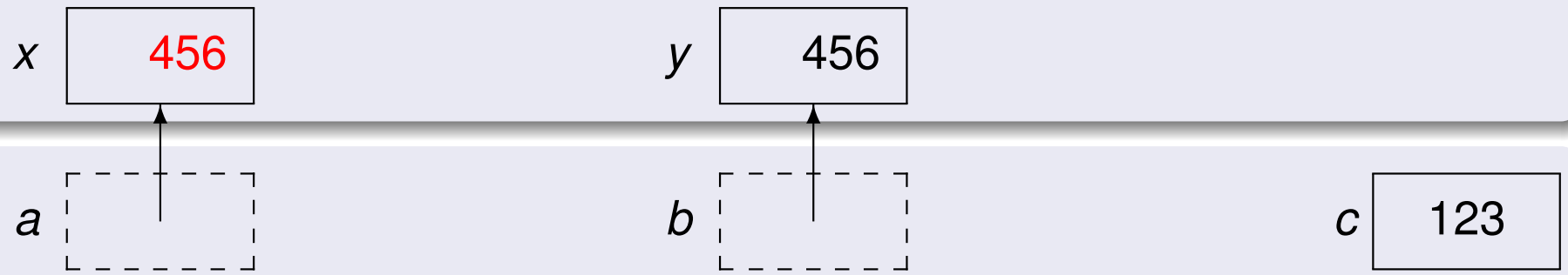
$c \leftarrow a$
$a \leftarrow b$
$b \leftarrow c$

Sorties

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

Déroulement de l'exemple

Variables



Algorithme de échanger

$c \leftarrow a$

$a \leftarrow b$

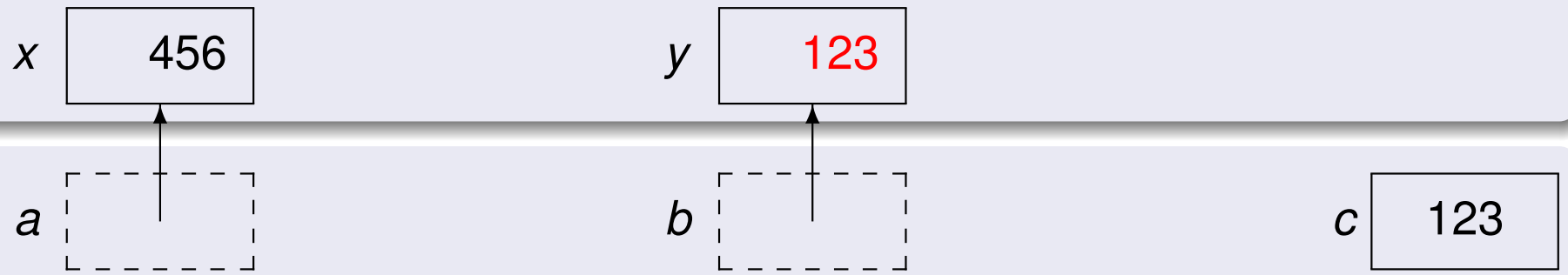
$b \leftarrow c$

Sorties

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

Déroulement de l'exemple

Variables



Algorithme de échanger

$c \leftarrow a$

$a \leftarrow b$

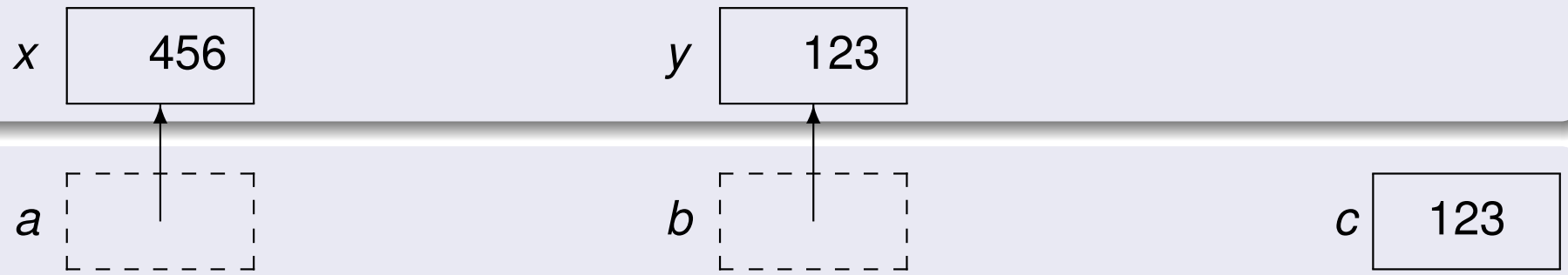
$b \leftarrow c$

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**
  résultat $\leftarrow 1$
sinon
  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)

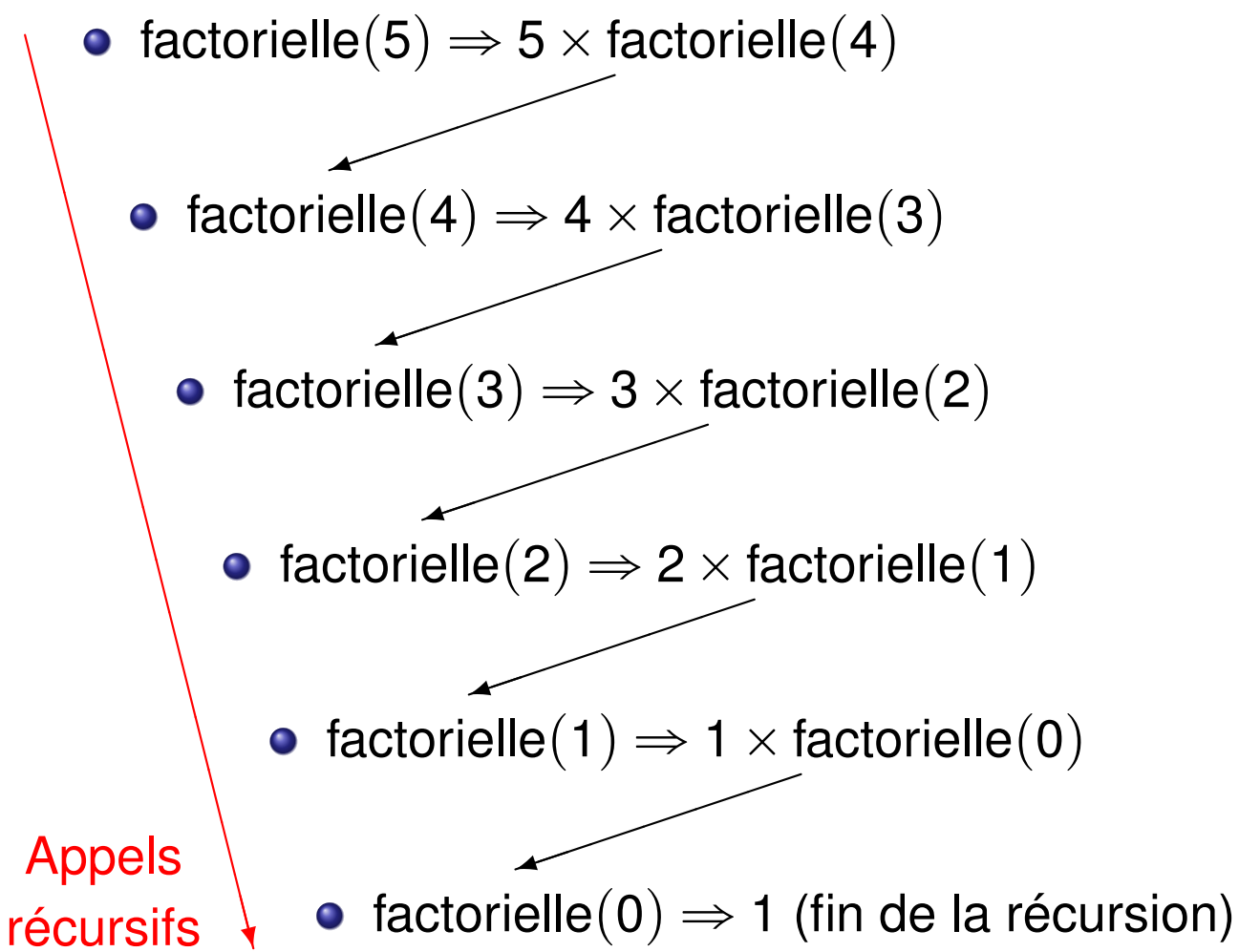
Exemple d'exécution

- 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)

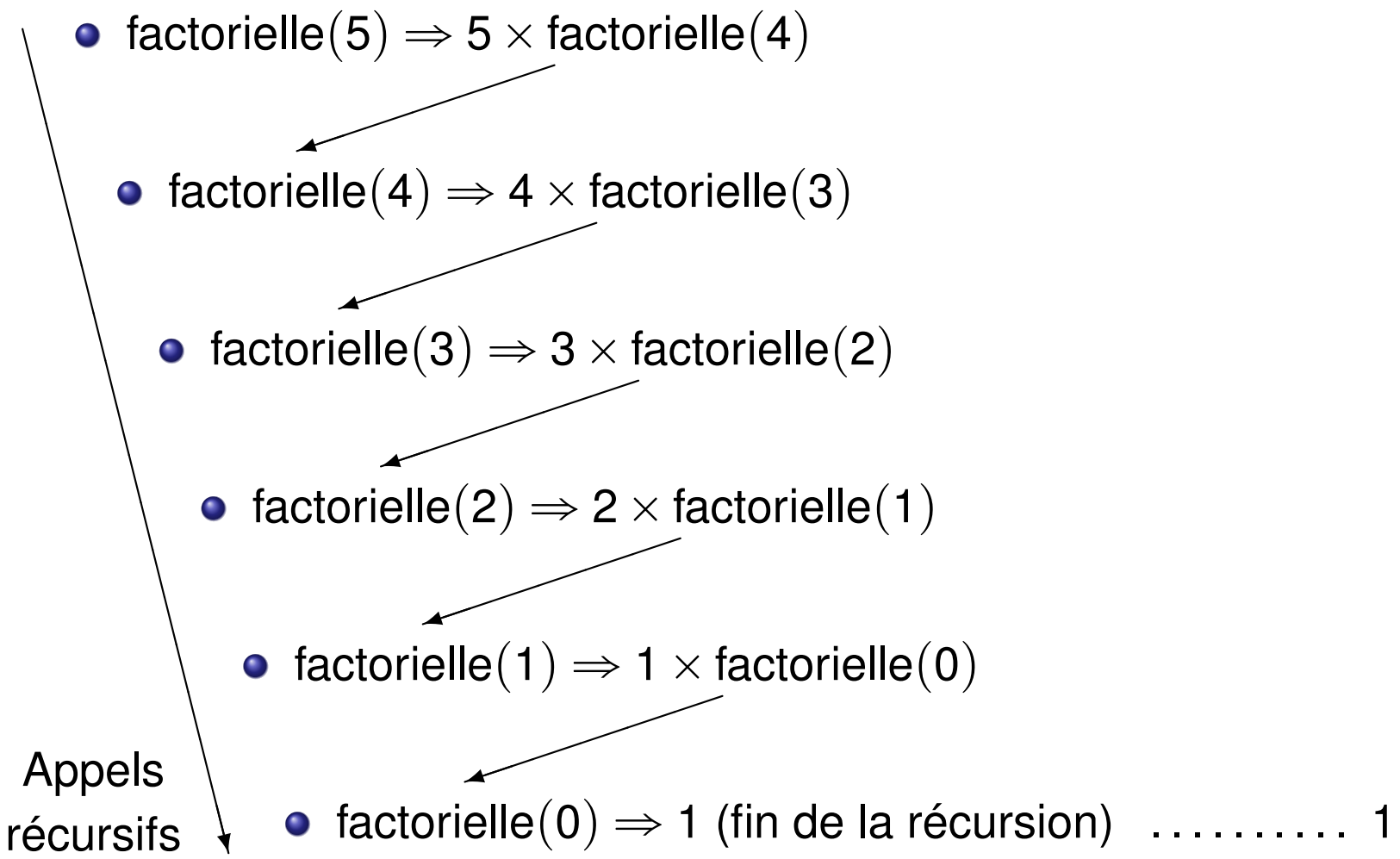
Exemple d'exécution

- 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)

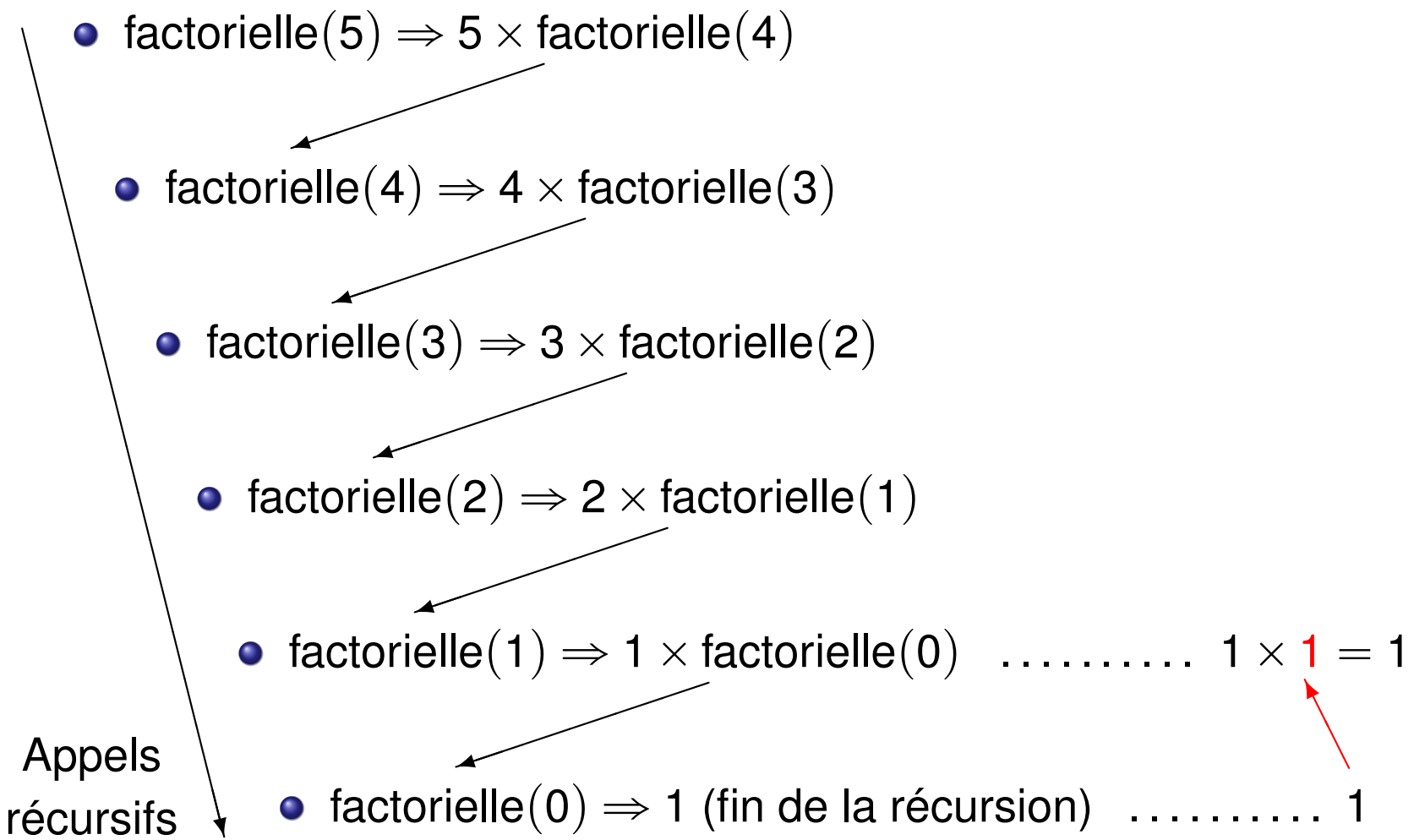
Exemple d'exécution



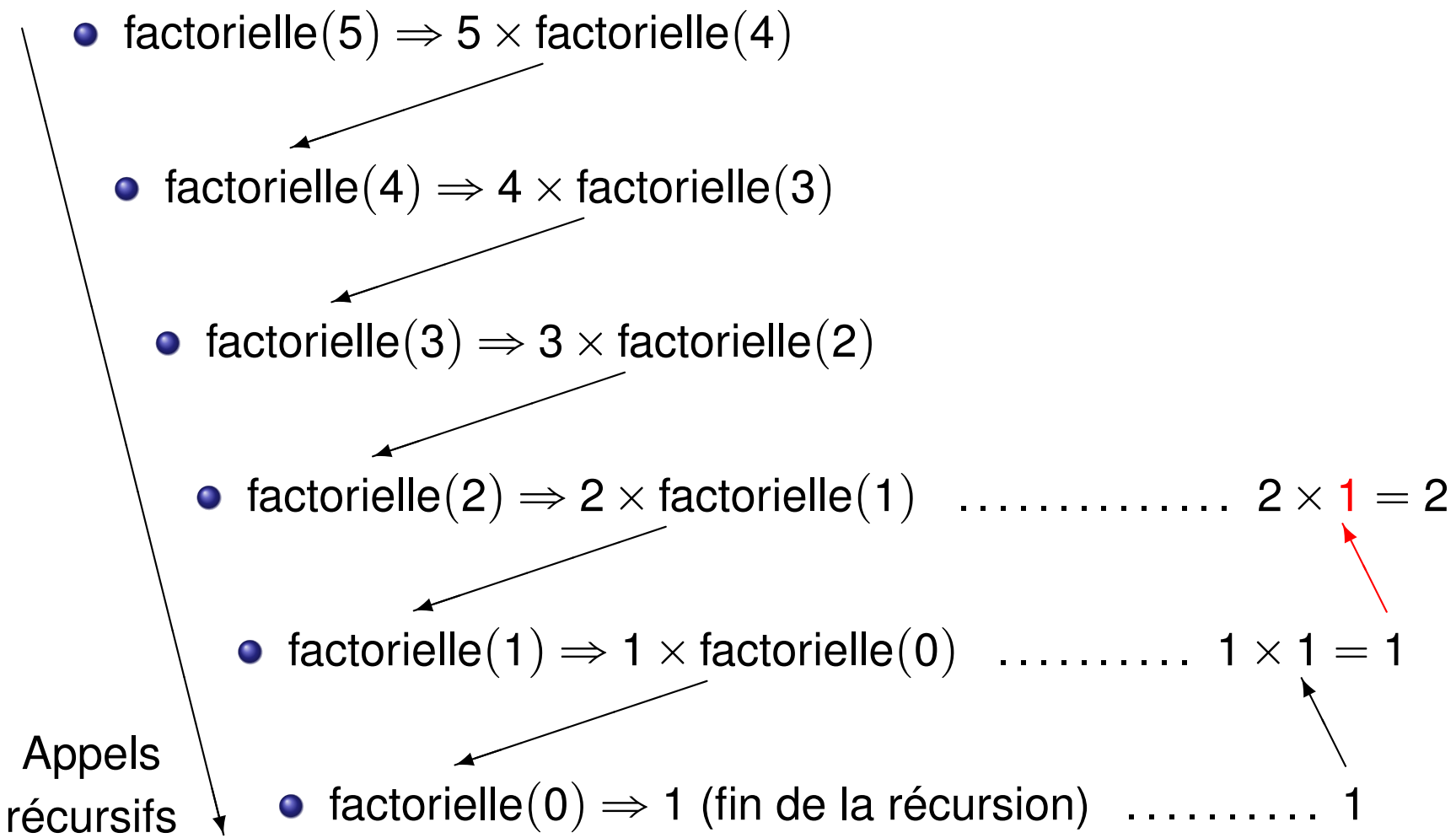
Exemple d'exécution



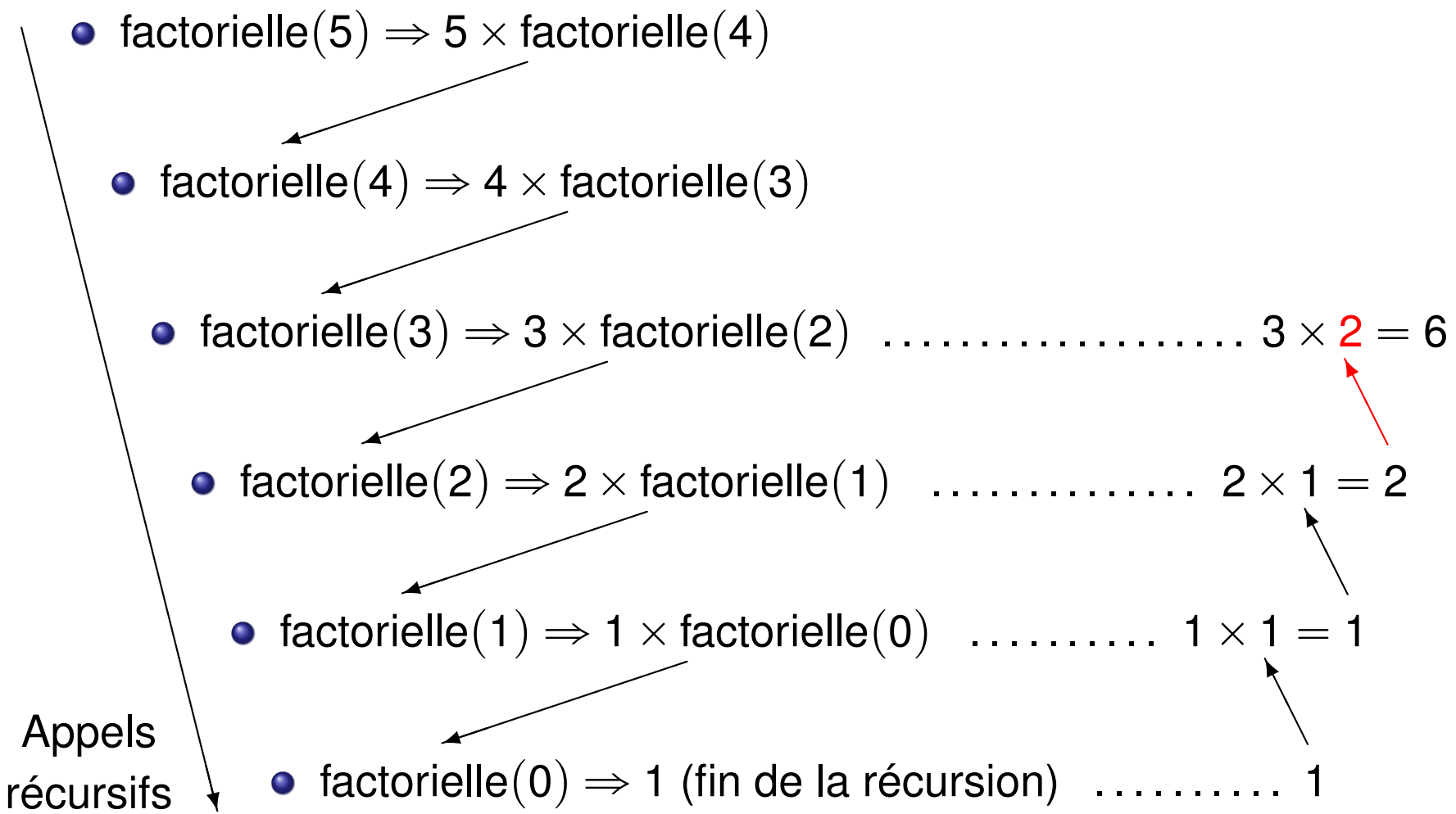
Exemple d'exécution



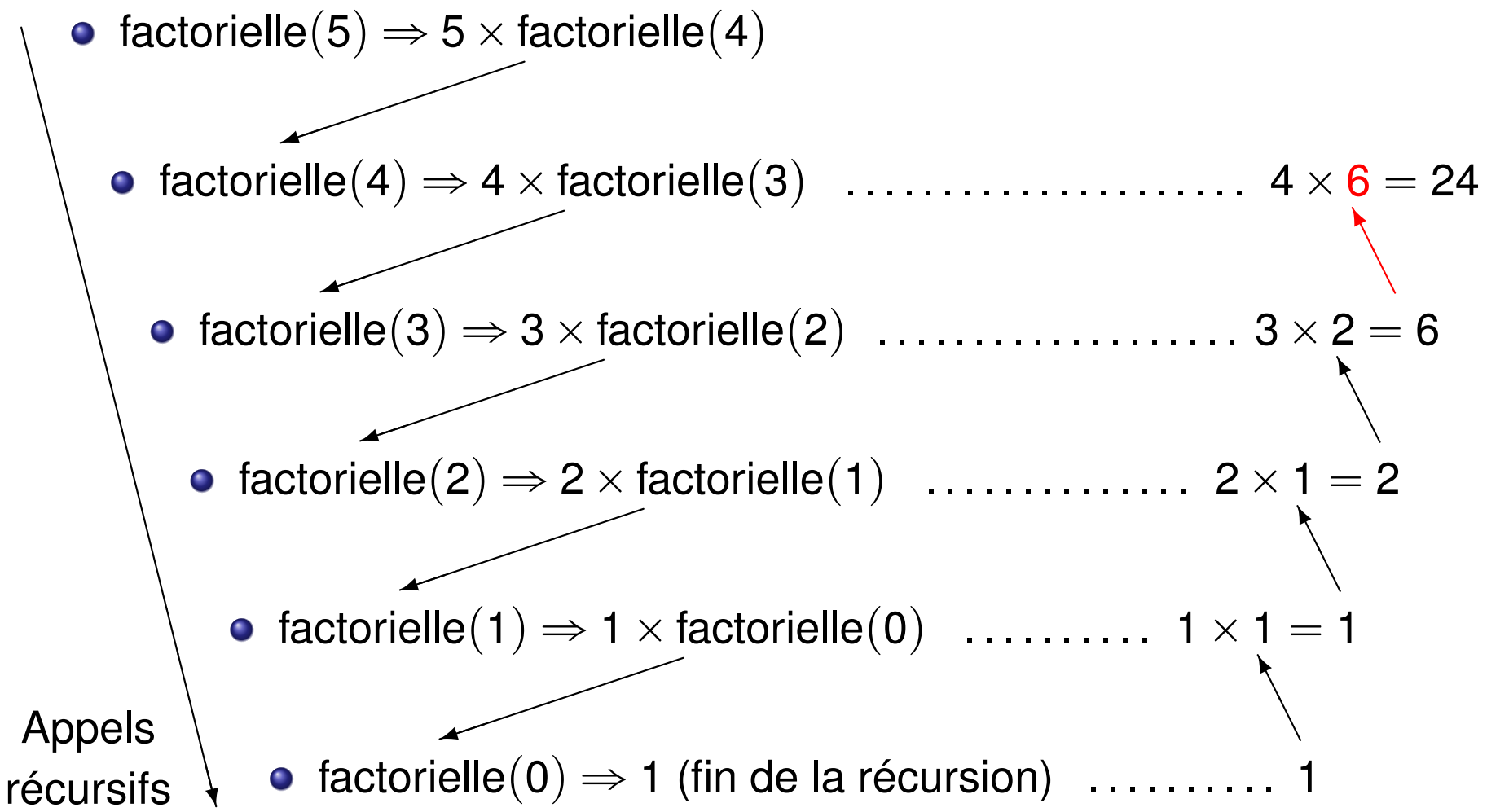
Exemple d'exécution



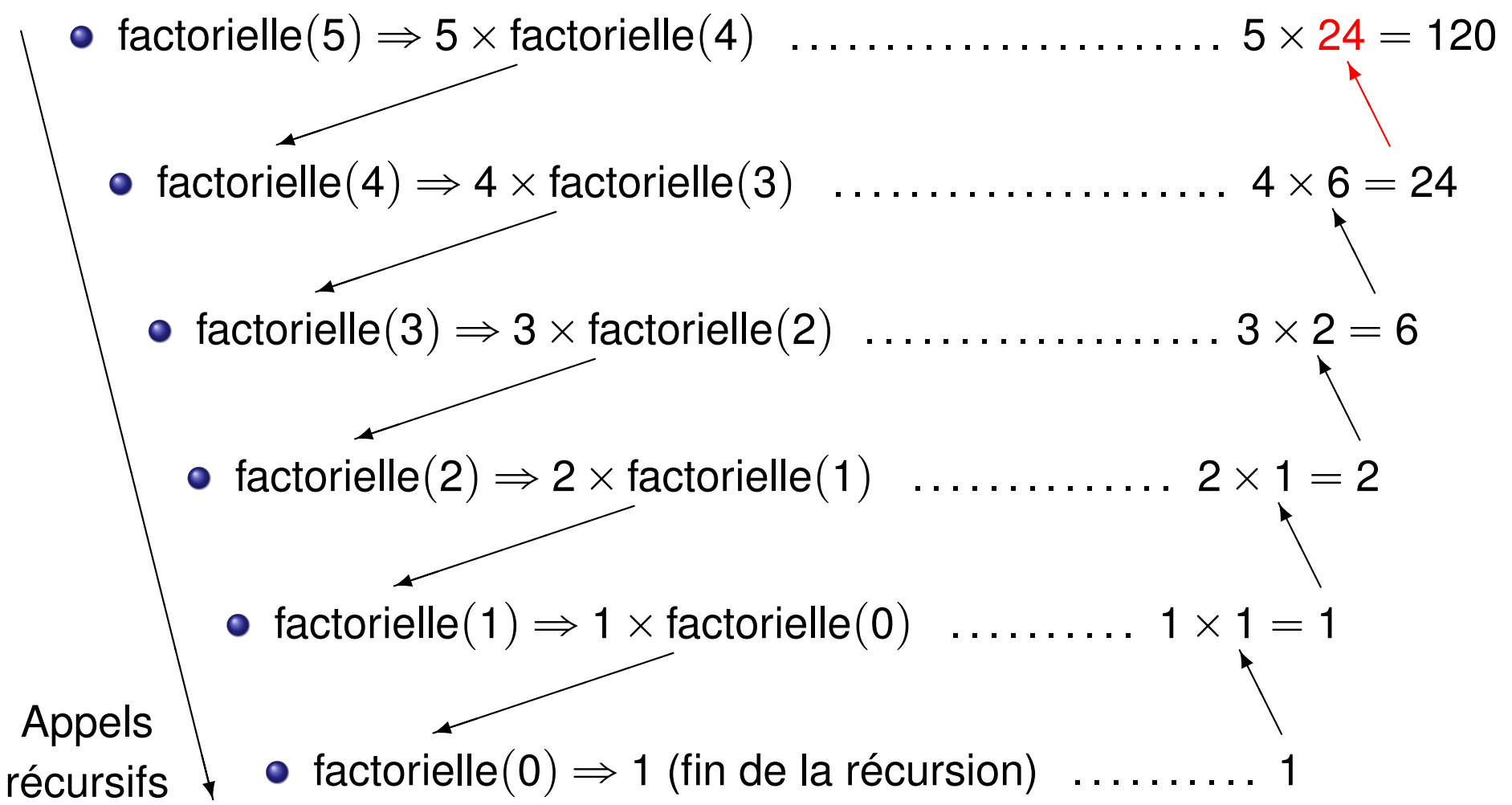
Exemple d'exécution



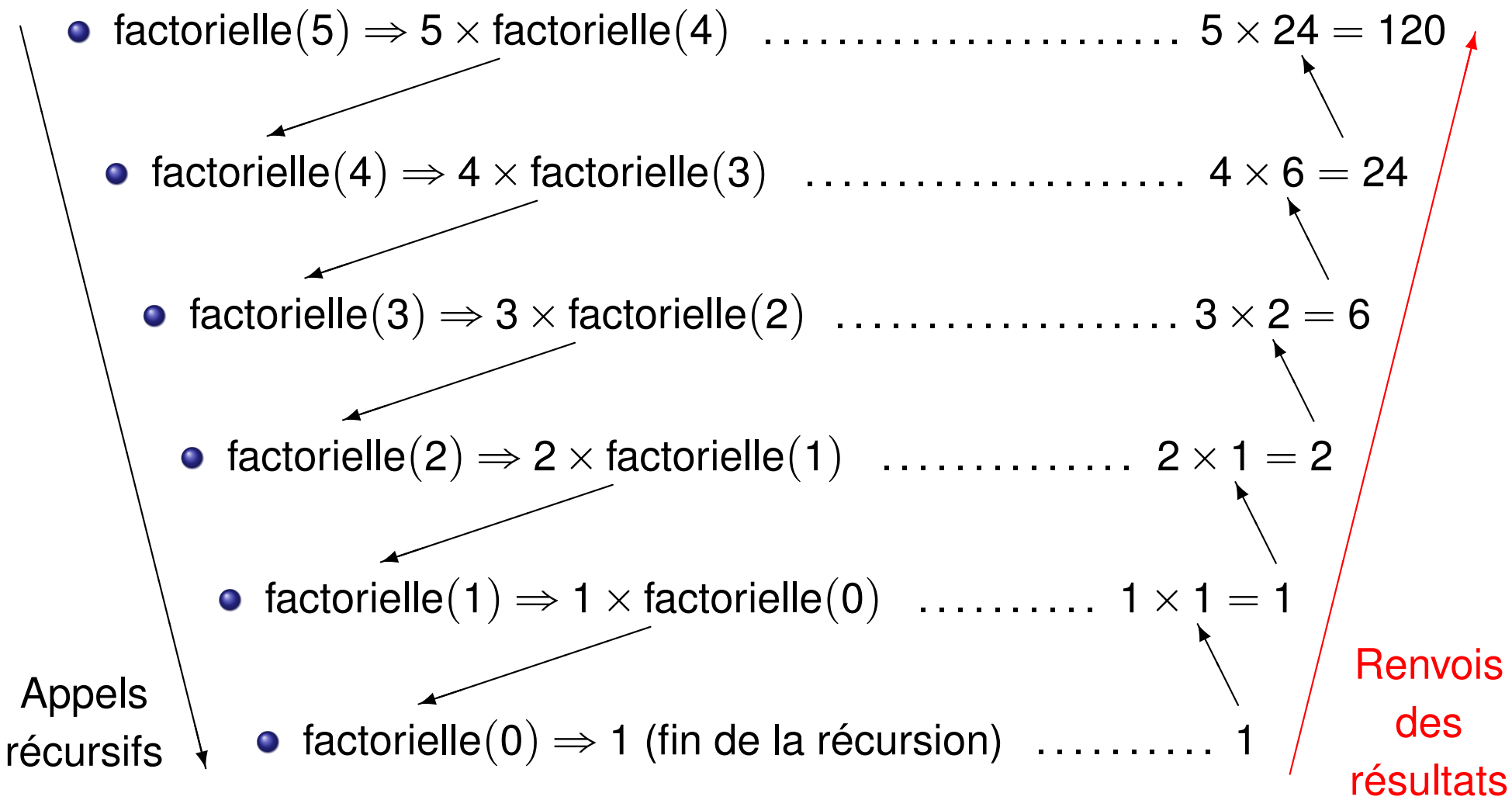
Exemple d'exécution



Exemple d'exécution



Exemple d'exécution



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 ← 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 ← 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 ← 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 ← lire
  litÉcritInv$(nb-1)$
  écrire *valeur*
fsi

- La récursivité n'est pas terminale.

Autre exemple

- Calcul du coefficient binomial C_n^k :

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

Calcul de C_n^k

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

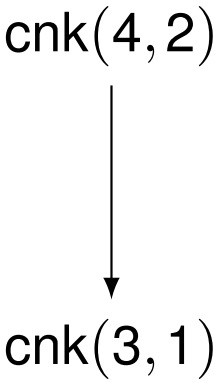
Algorithme de cnk

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

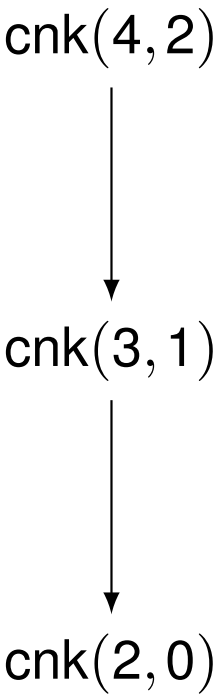
Exécution de cnk$(4,2)$

cnk$(4,2)$

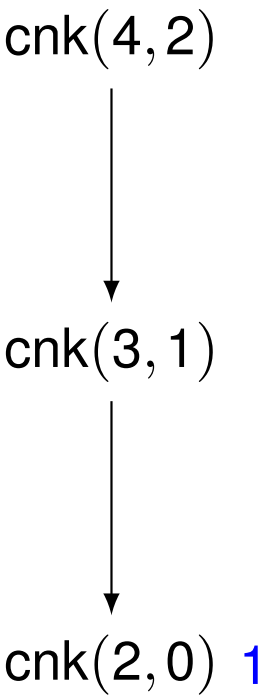
Exécution de cnk$(4,2)$



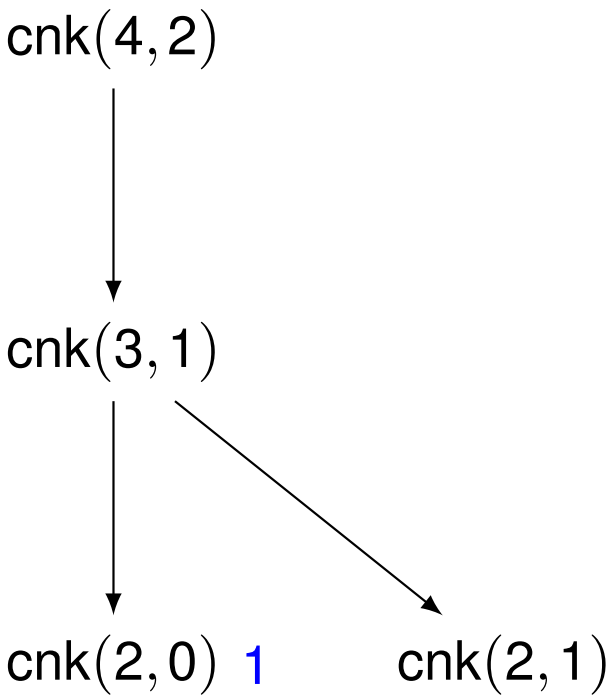
Exécution de cnk$(4,2)$



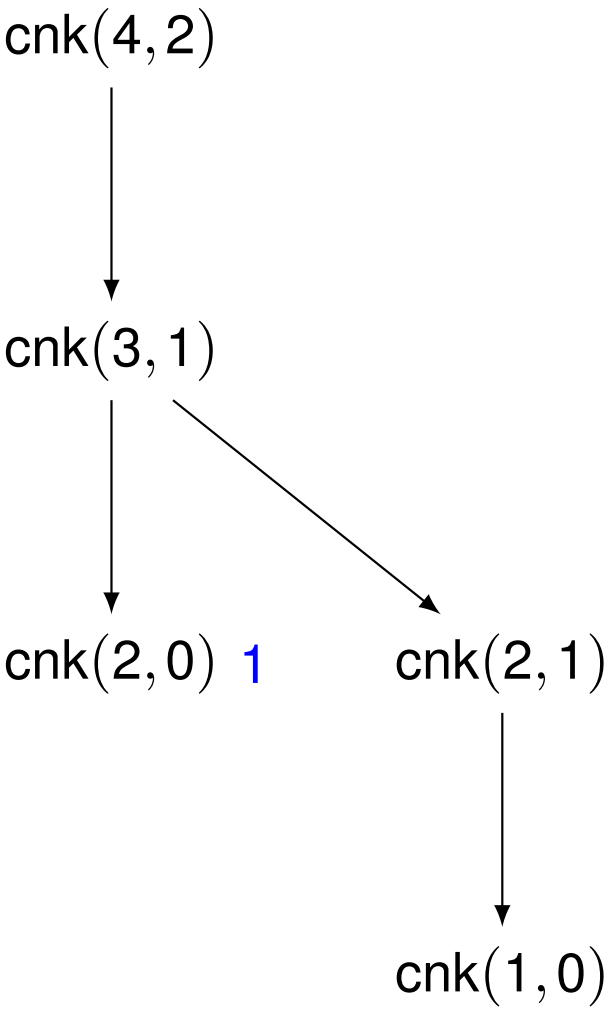
Exécution de cnk$(4,2)$



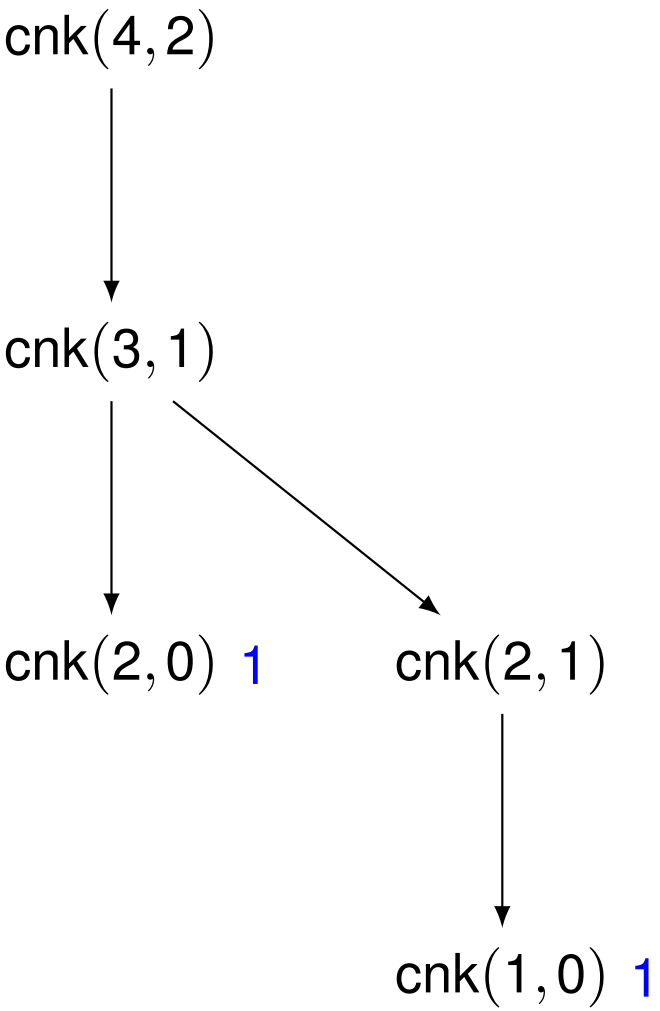
Exécution de cnk(4,2)



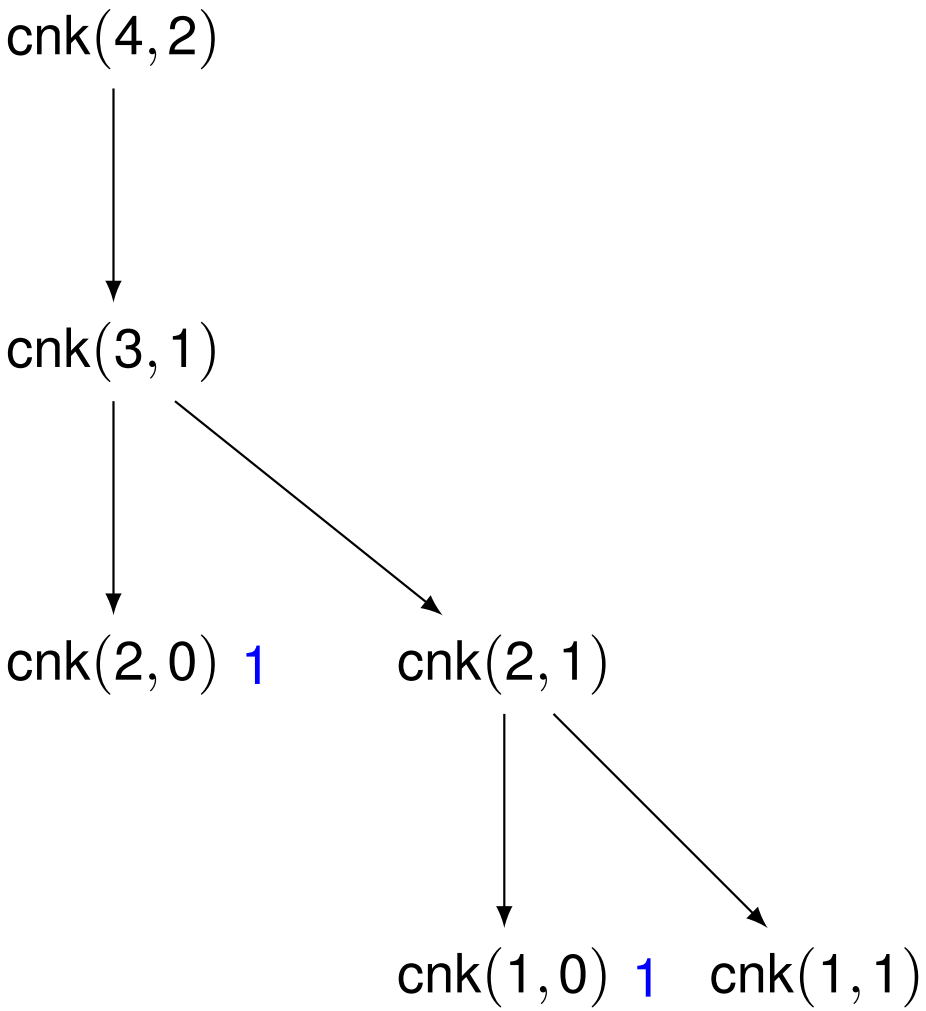
Exécution de cnk(4,2)



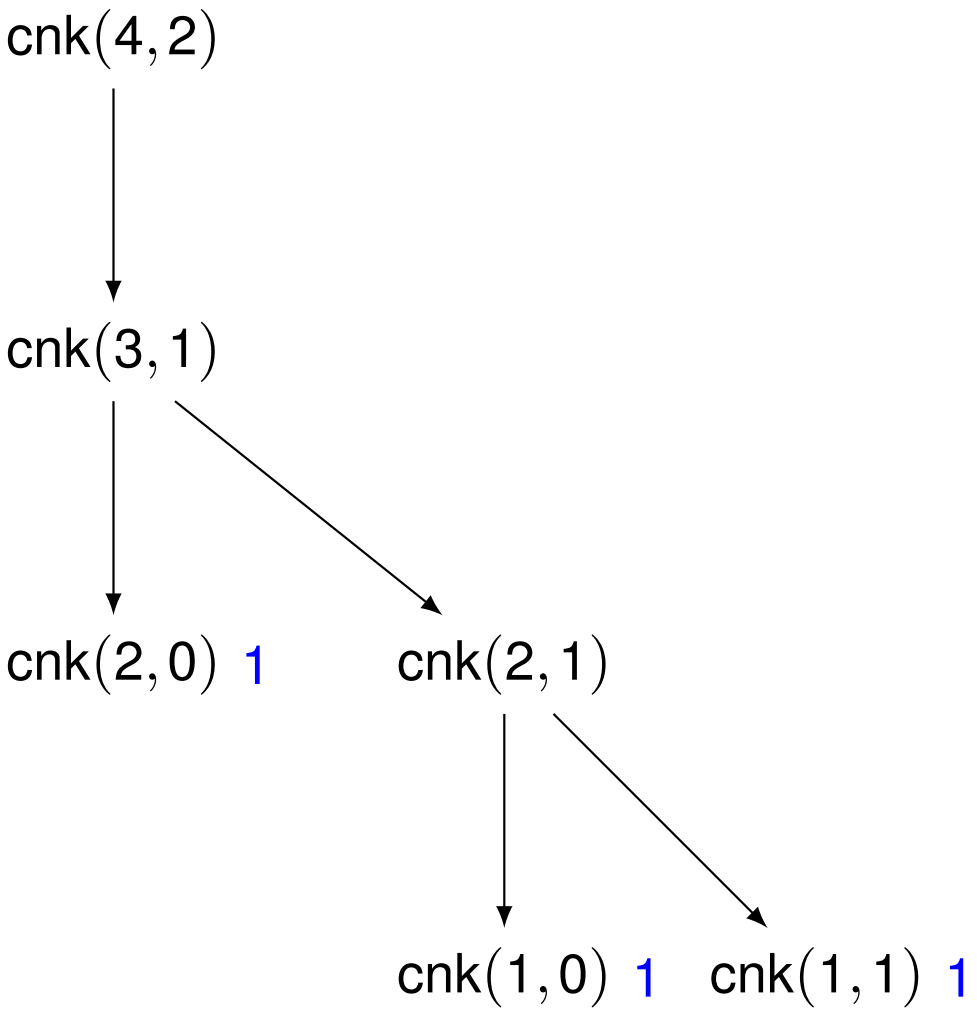
Exécution de cnk(4,2)



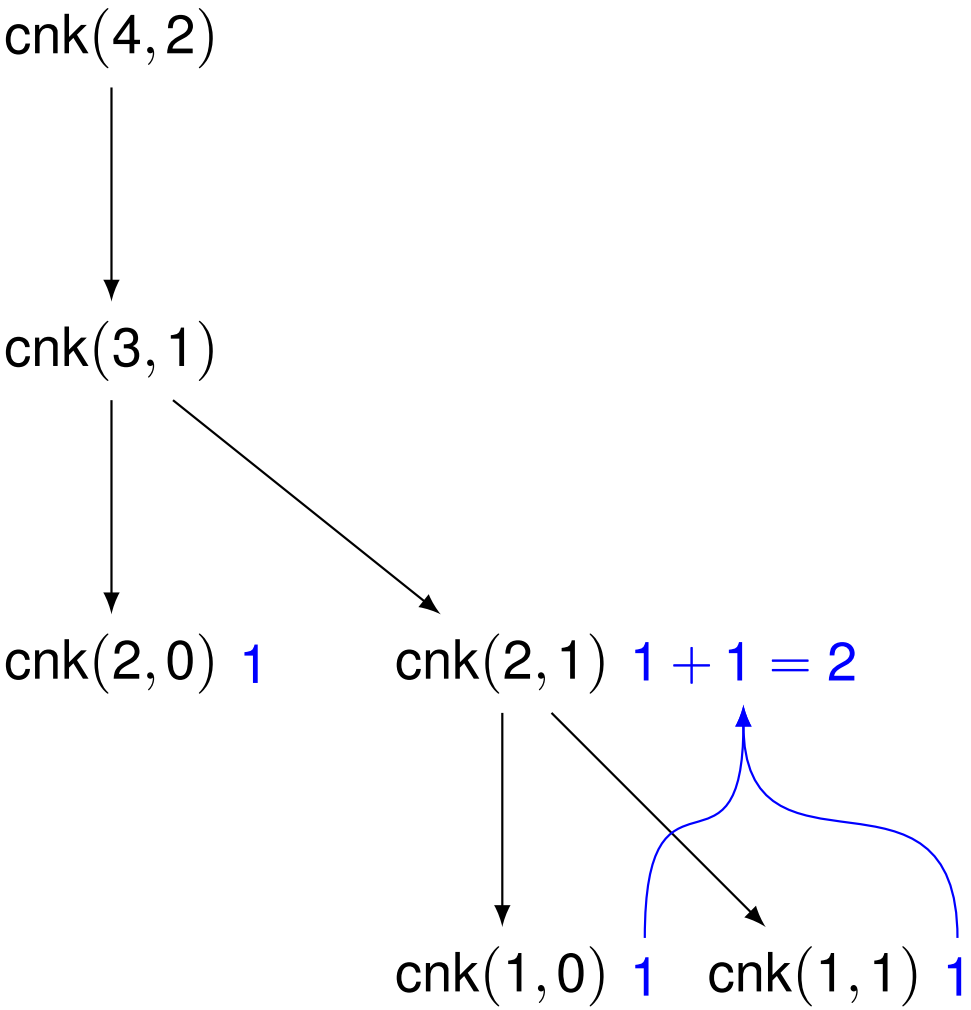
Exécution de cnk(4,2)



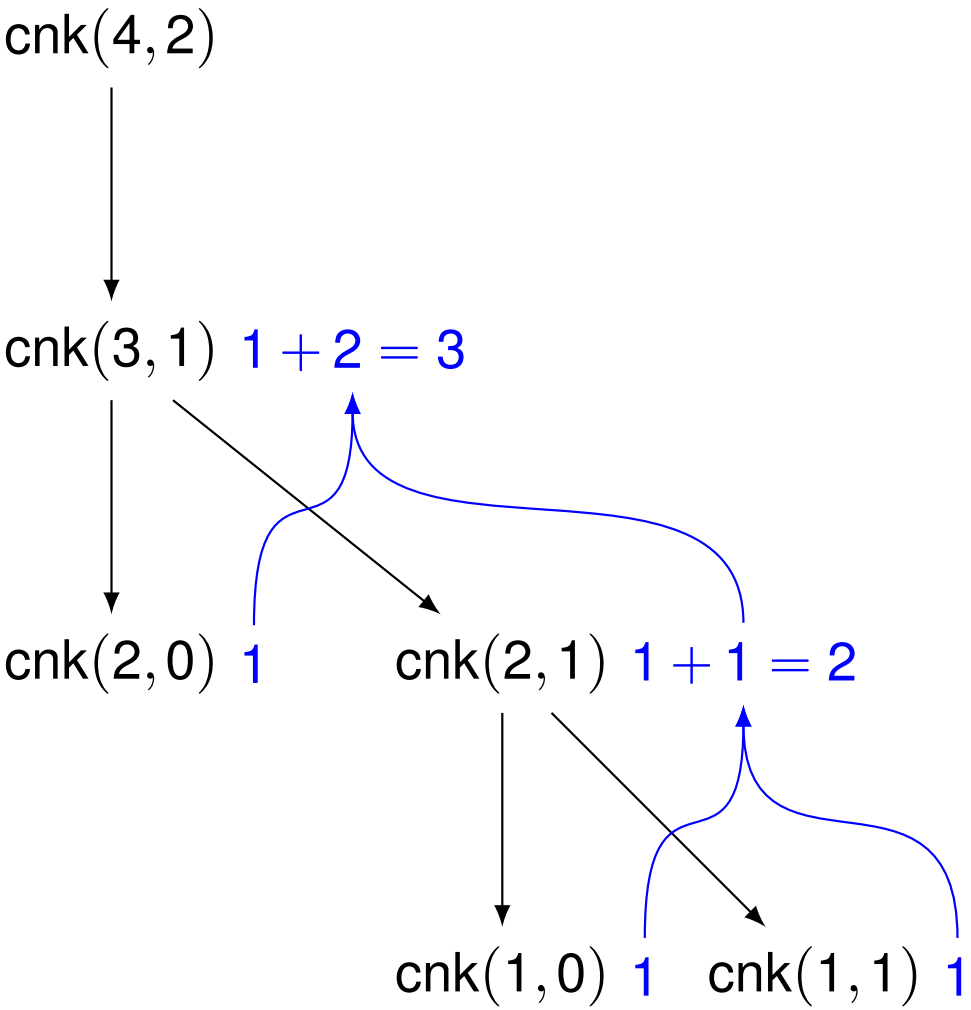
Exécution de cnk(4,2)



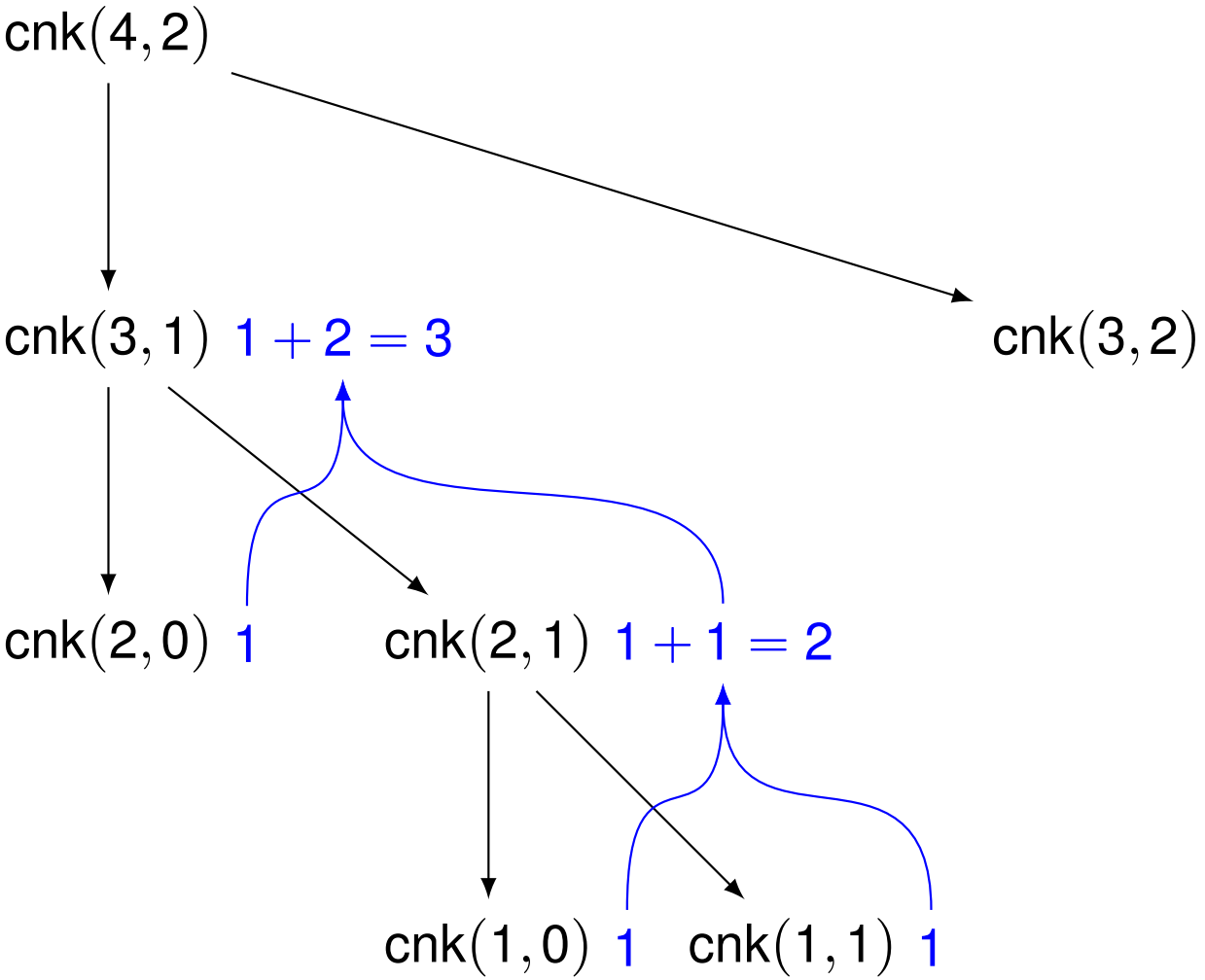
Exécution de cnk(4,2)



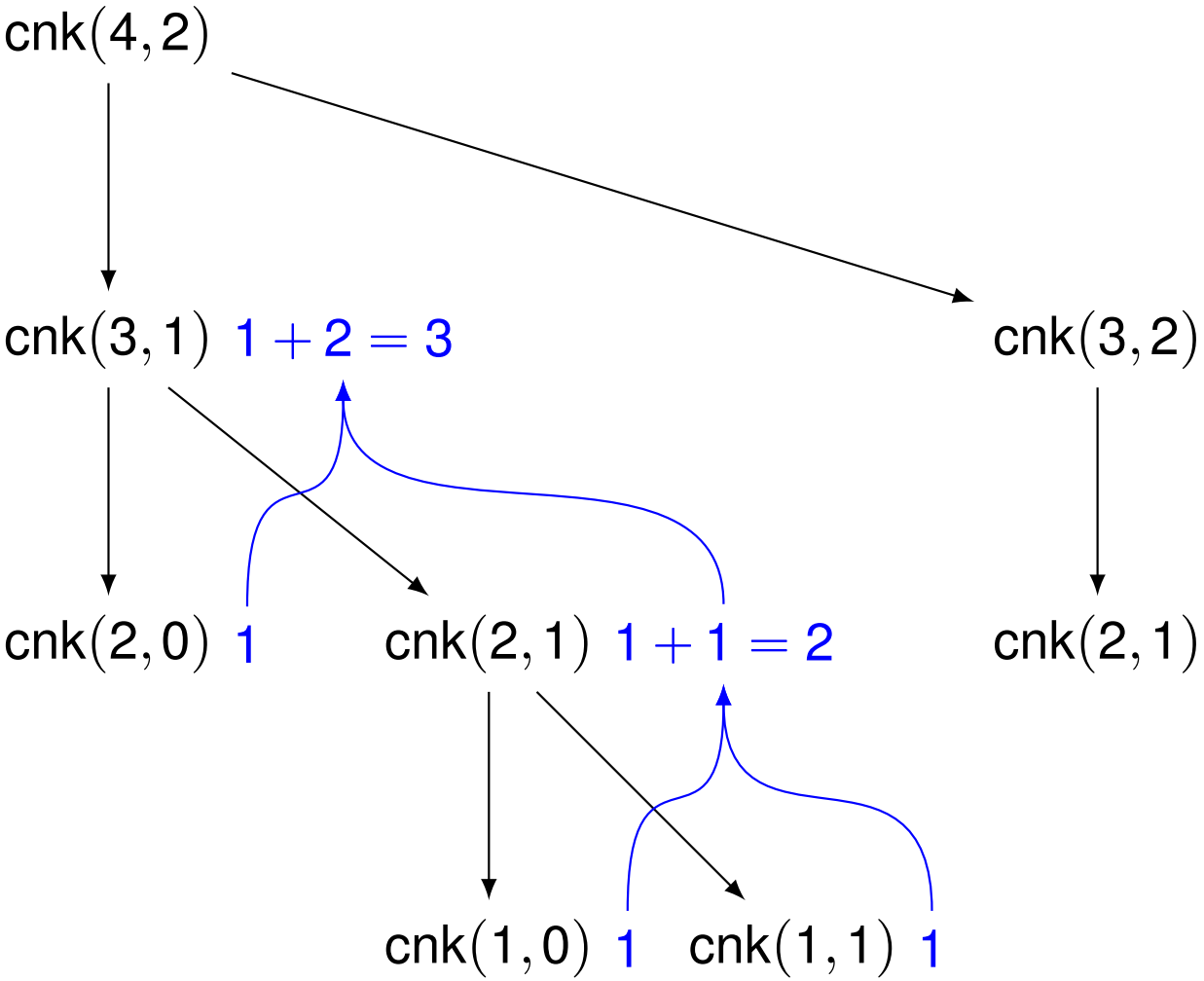
Exécution de cnk(4,2)



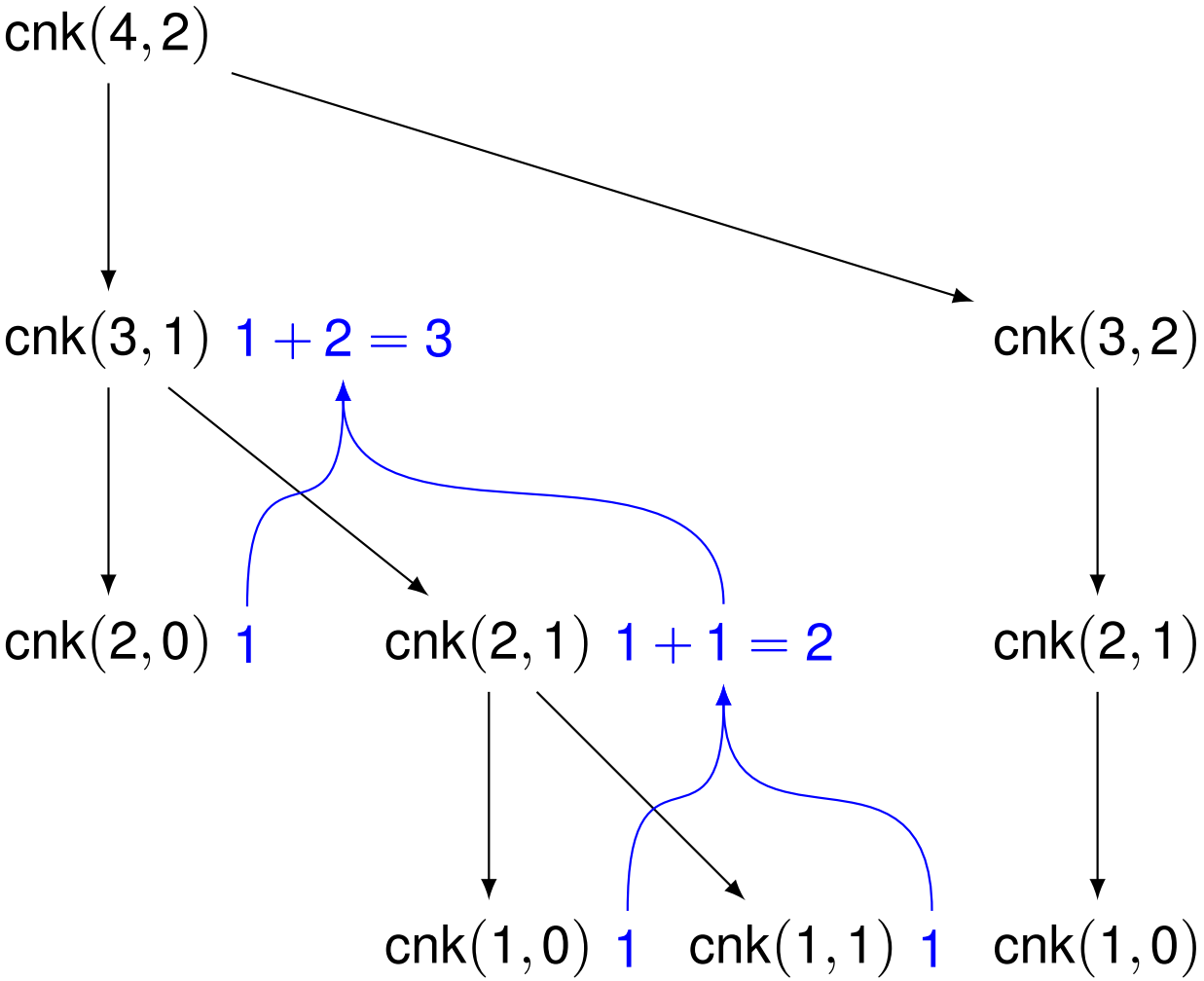
Exécution de cnk(4,2)



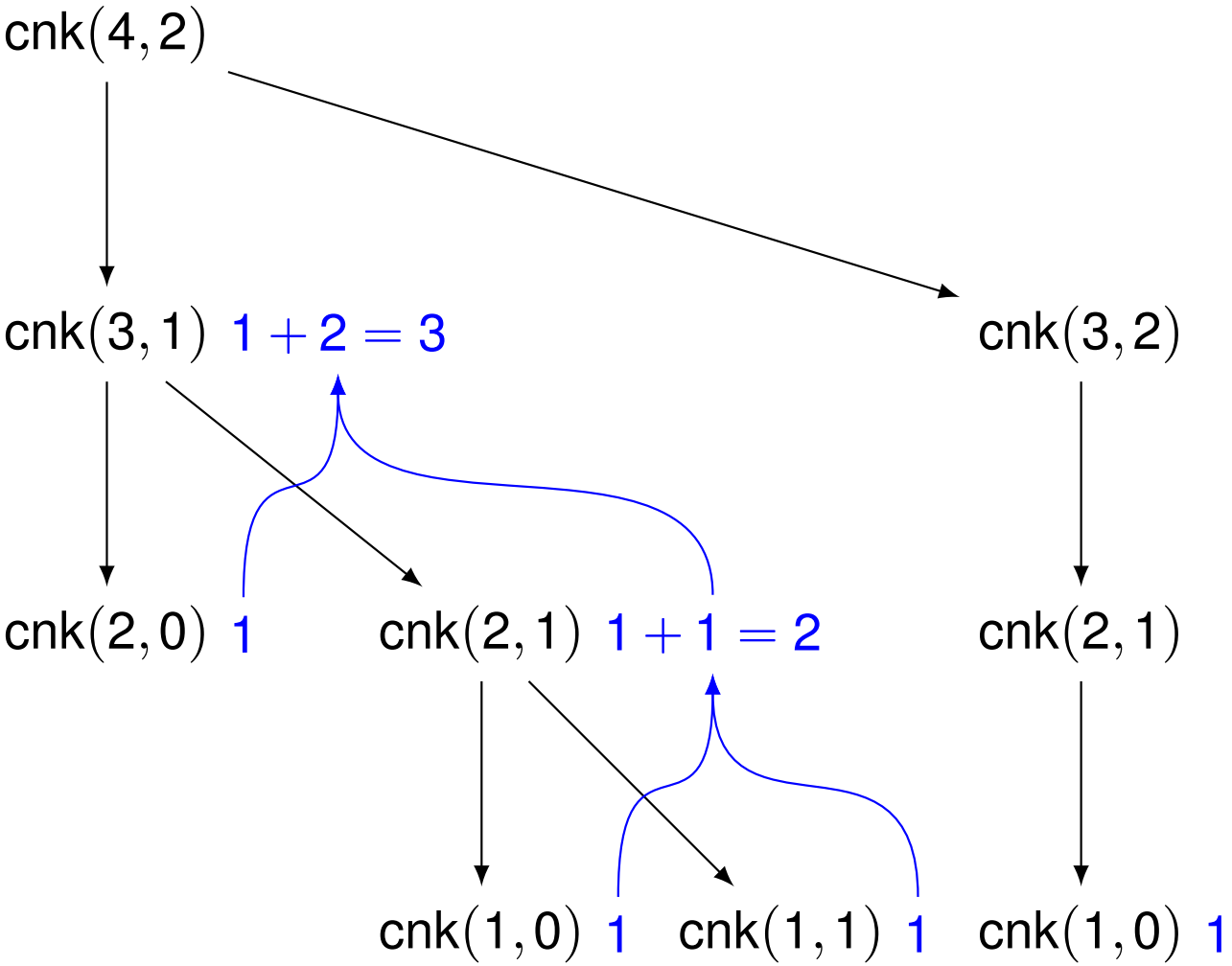
Exécution de cnk(4,2)



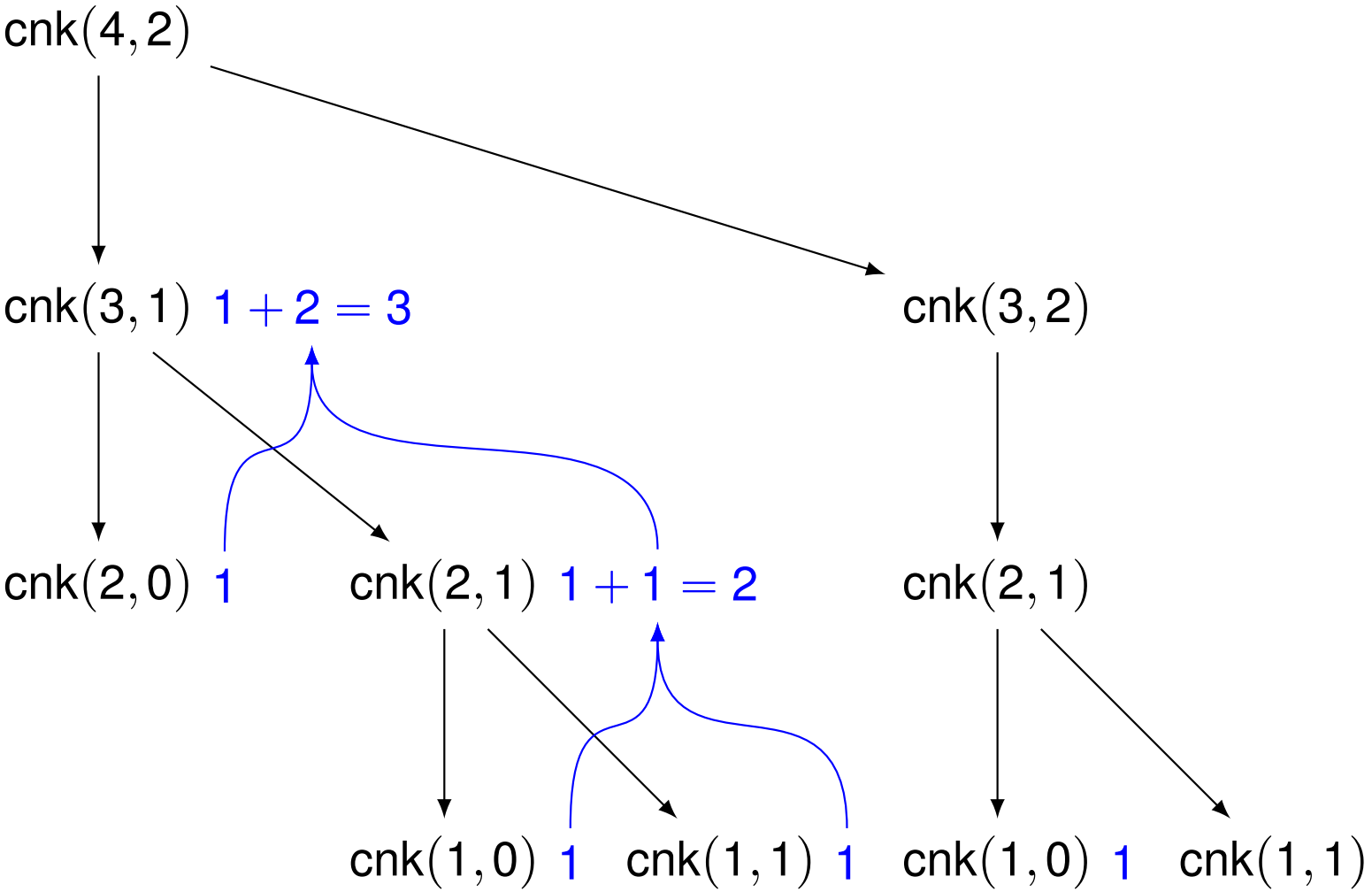
Exécution de cnk(4,2)



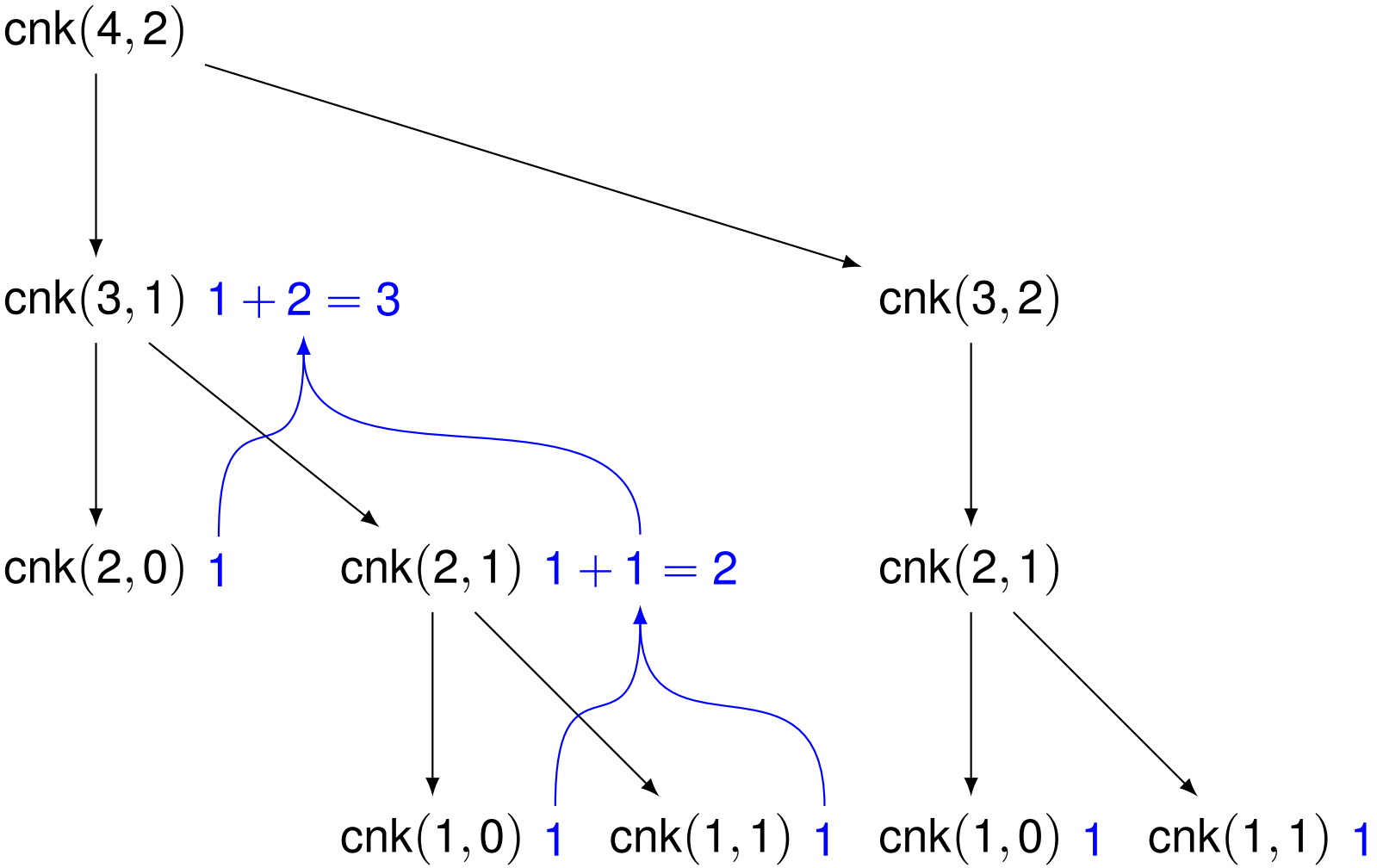
Exécution de cnk(4,2)



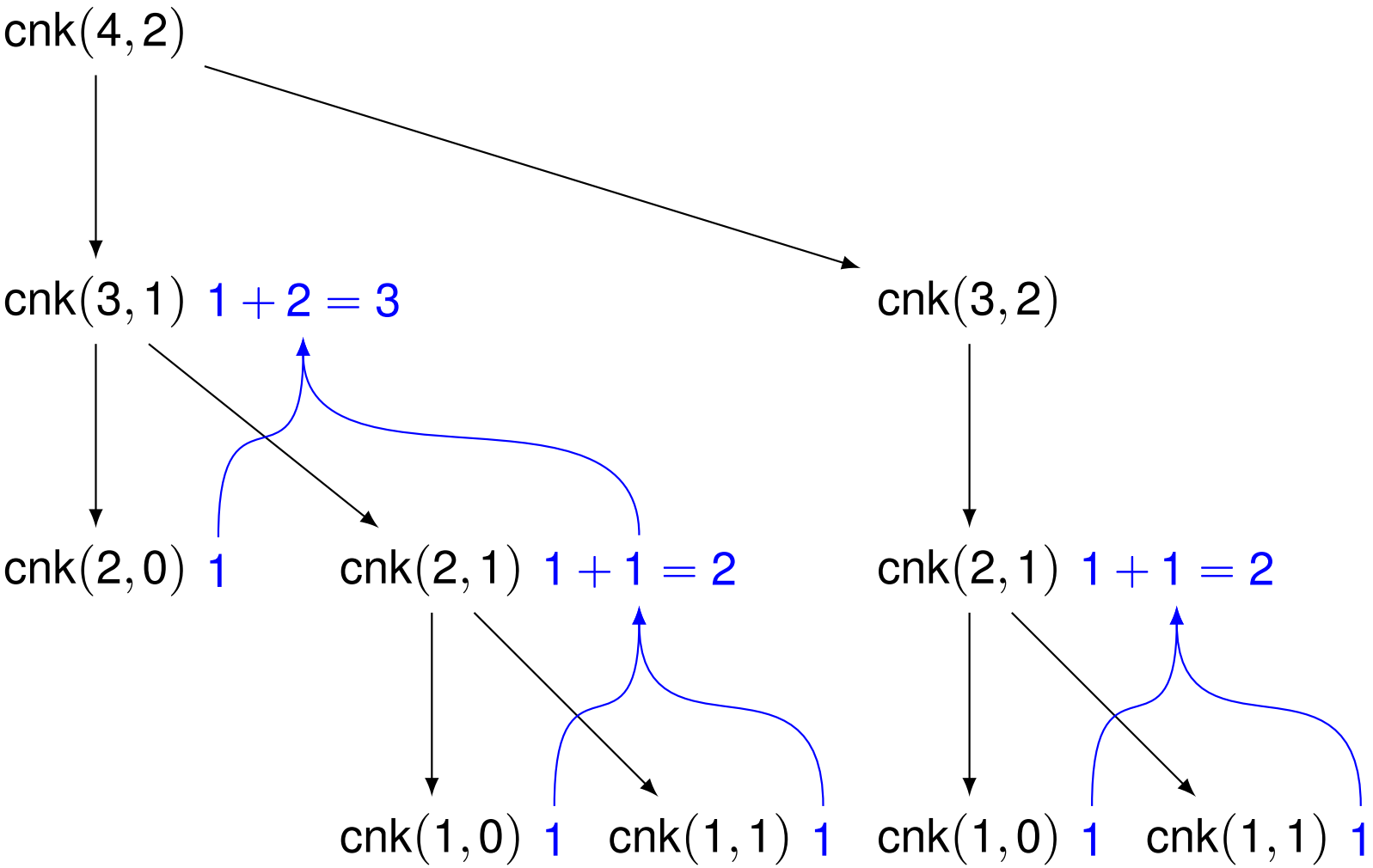
Exécution de cnk(4,2)



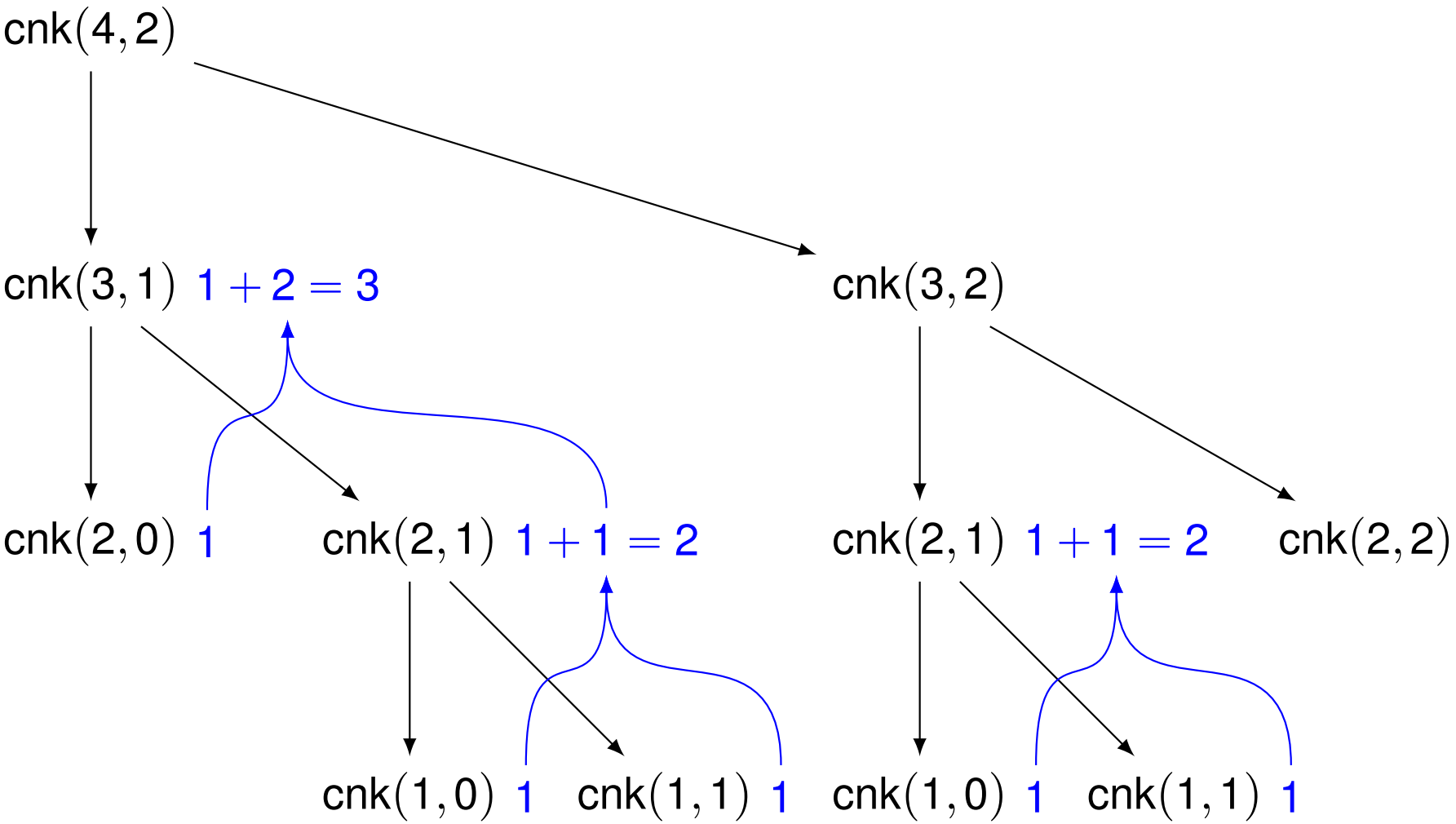
Exécution de cnk(4,2)



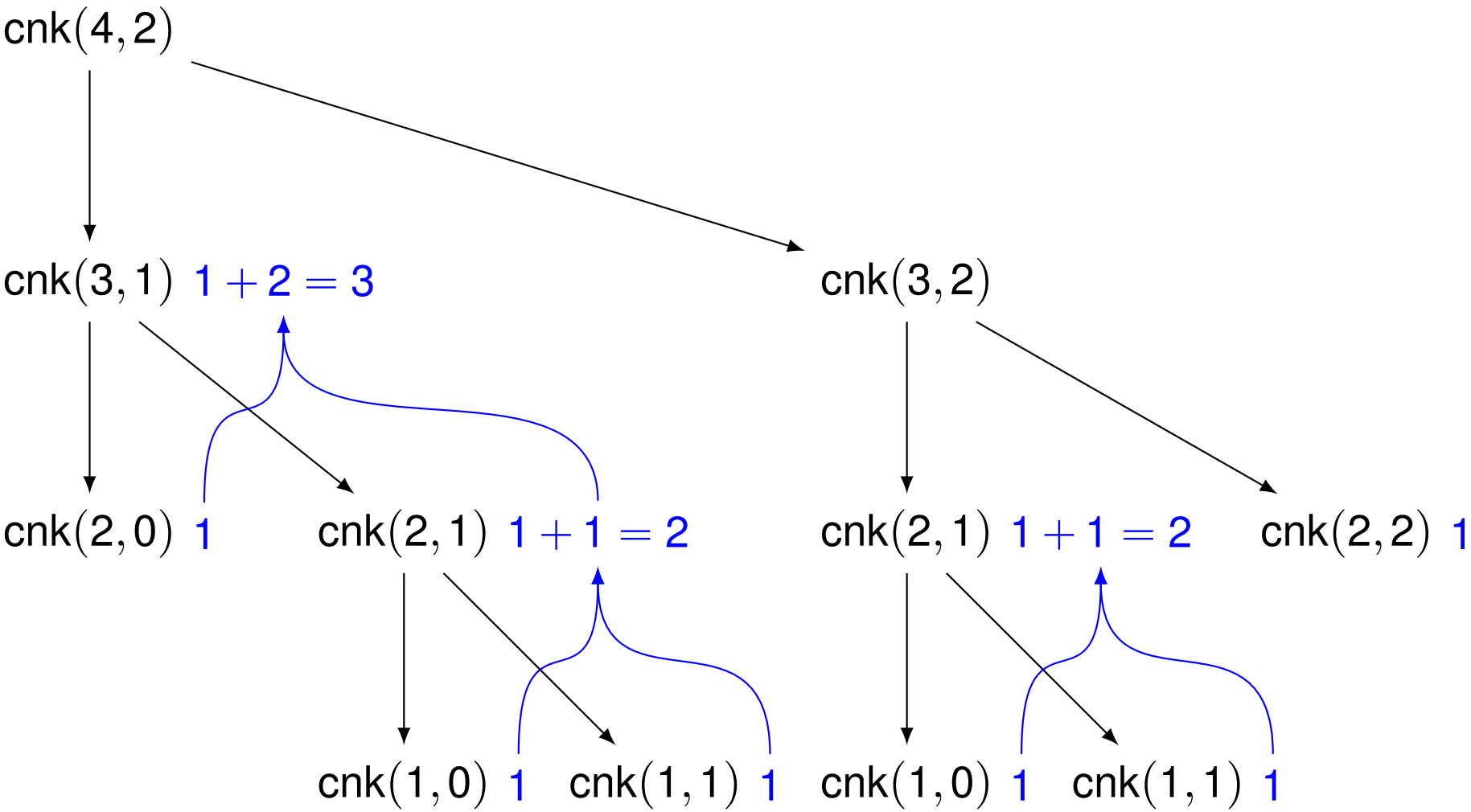
Exécution de cnk(4,2)



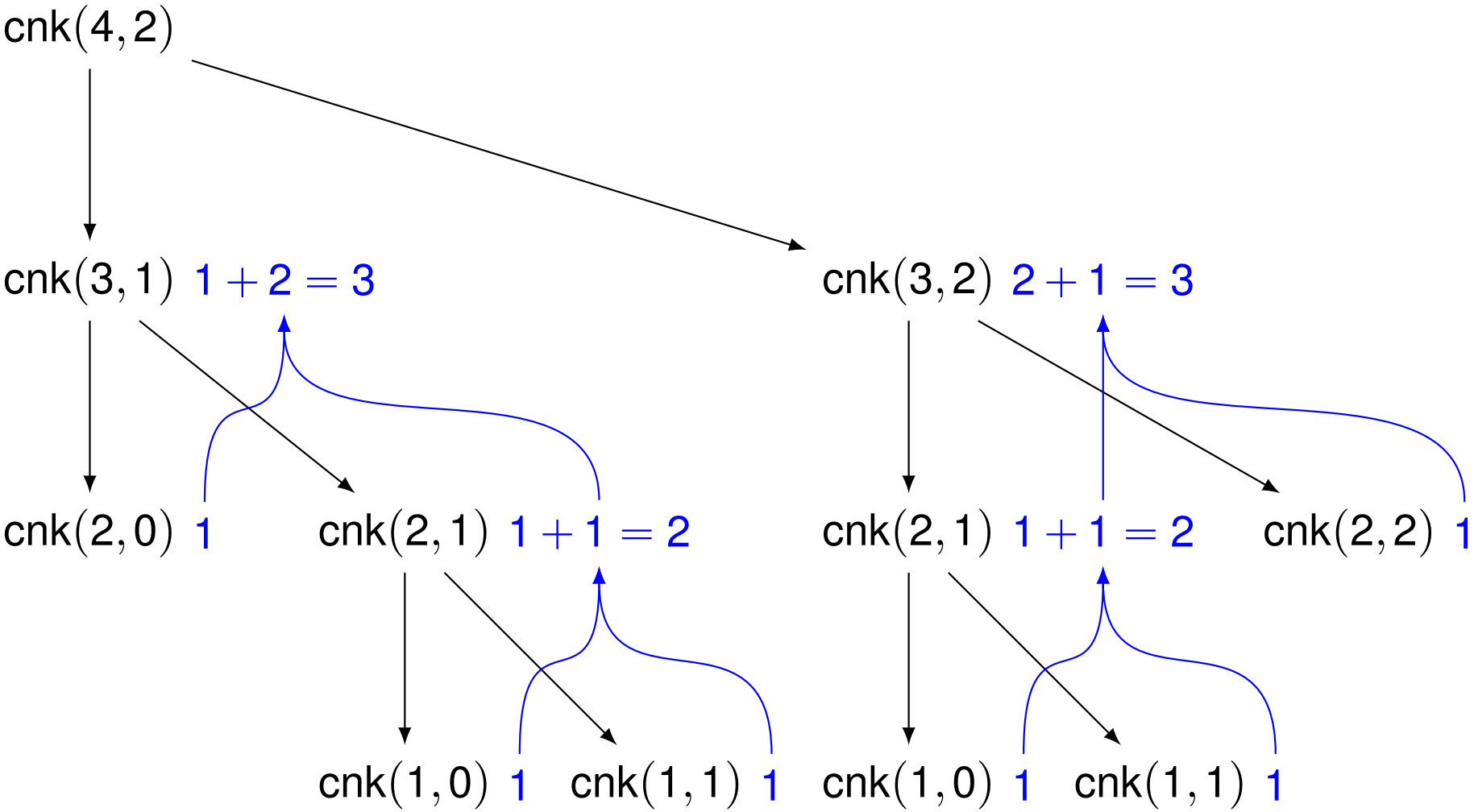
Exécution de cnk(4,2)



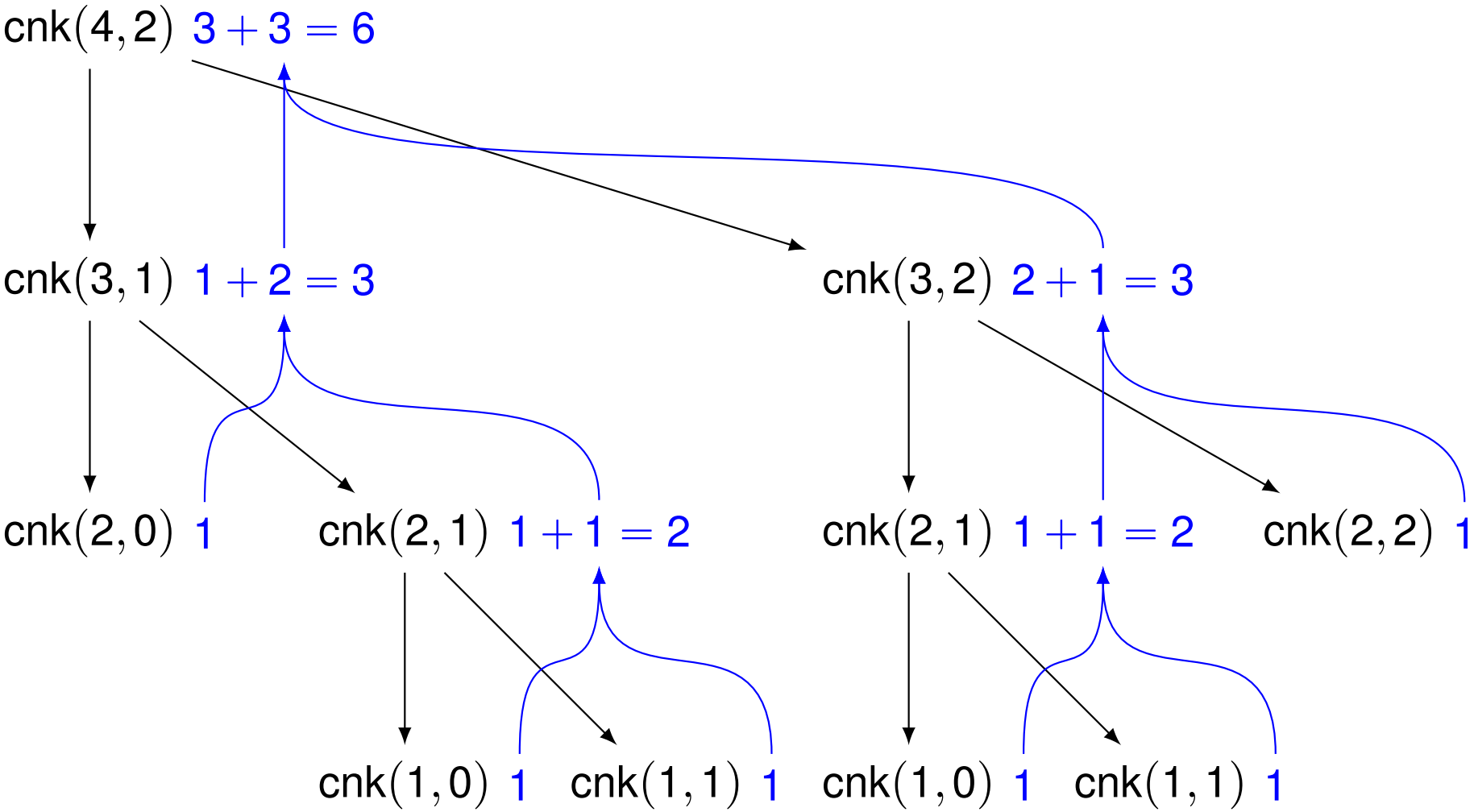
Exécution de cnk(4,2)



Exécution de cnk(4,2)



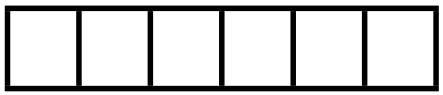
Exécution de cnk(4,2)



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

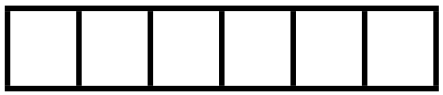
Déclaration (exemple)

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

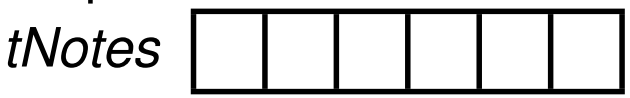
tNotes	(tableau [6] de réels)	Liste de notes	INTERMÉDIAIRE

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 :



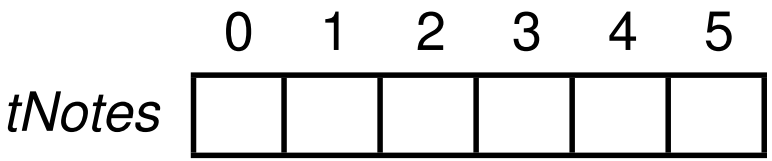
un tableau de 6 réels

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 !).



- 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 :

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

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

...

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 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**
  $tab[i] \leftarrow$ 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

Tableaux en Java

Déclaration

- Forme générale : `<type>[] <nom>;`
- Exemple : `int[] tMesValeurs;`

Tableaux en Java

Déclaration

- Forme générale : `<type>[] <nom>;`
- Exemple : `int[] tMesValeurs;`

Création/initalisation

- Avant de pouvoir utiliser un tableau, il faut le créér (allocation mémoire) :
  Exemple : `int[] tMesValeurs = new int[8];`
- On peut aussi combiner allocation et initialisation (variante) :
  Exemple : `int[] tMesValeurs = {5, 9, -4, 2, 12, 17, 2, 7};`

Tableaux en Java

Déclaration

- Forme générale : `<type>[] <nom>;`
- Exemple : `int[] tMesValeurs;`

Création/initalisation

- Avant de pouvoir utiliser un tableau, il faut le créér (allocation mémoire) :
  Exemple : `int[] tMesValeurs = new int[8];`
- On peut aussi combiner allocation et initialisation (variante) :
  Exemple : `int[] tMesValeurs = {5, 9, -4, 2, 12, 17, 2, 7};`

Utilisation

- Similaire au langage algorithmique.
- Les indices sont placés entre crochets, et vont de 0 à *taille* − 1.
- Exemple : `a = tMesValeurs[5] + 6; tMesValeurs[6] = a + a;`
- Taille d'un tableau : attribut `.length` (ex. : `tMesValeurs.length`)
- Attention à ne pas *sortir* des tableaux !

Plan

Les caractères

- Notés entre apostrophes :
  Exemples : '`a`', '`A`', '`3`', '`{`' sont des caractères
- On utilise un seul jeu de caractères à la fois
- Il suit certaines conventions :
  - il contient tous les caractères utilisables (lettres, chiffres, ponctuations) ;
  - chaque caractère est repéré par sa position dans le jeu de caractères ;
    $\Rightarrow$ notion d'ordre entre les caractères
  - les chiffres sont contigus et dans l'ordre ;
  - les lettres minuscules sont contiguës et dans l'ordre ;
  - les lettres majuscules sont contiguës et dans l'ordre ;
  - l'ordre entre ces différents sous-ensembles peut être quelconque.

Opérations sur les caractères

- Addition / soustraction d'un entier :

$$\textit{caractère} \pm \textit{entier} \Rightarrow \textit{caractère}$$

  - Effectue un décalage dans le jeu de caractères, du nombre de positions spécifié à partir du caractère donné :
    - vers la droite avec l'addition ;
    - vers la gauche avec la soustraction.
  - Exemples : `'a'` $+ 1 \Rightarrow$ `'b'`, `'5'` $- 3 \Rightarrow$ `'2'`, `'G'` $+ 0 \Rightarrow$ `'G'`

Exemple de jeu de caractères

									0	1	2	3	4	5	6	7	8	9	
			A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
R	S	T	U	V	W	X	Y	Z						a	b	c	d	e	f
g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Opérations sur les caractères

- Différence entre deux caractères :

  $$\textit{caractère} - \textit{caractère} \Rightarrow \textit{entier}$$

  - Renvoie le décalage relatif entre les deux caractères :
    le déplacement nécessaire pour arriver au 1er car en partant du 2e.
  - Exemples : 'b' − 'a' ⇒ 1, '3' − '5' ⇒ −2, 'G' − 'G' ⇒ 0, 'A' − 'a' ⇒ ...

Exemple de jeu de caractères

									0	1	2	3	4	5	6	7	8	9	
			A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
R	S	T	U	V	W	X	Y	Z						a	b	c	d	e	f
g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Opérations sur les caractères

- Comparaisons :

$$caractère \{ < = > \leqslant \geqslant \neq \} caractère \Rightarrow booléen$$

  - Compare deux caractères selon leurs positions dans le jeu de caractères : un caractère est inférieur à un autre s'il est placé avant
  - Exemples : 'a' < 'b' ⇒ vrai, '5' > '9' ⇒ faux, 'G' ≠ 'I' ⇒ vrai

- Il n'est pas nécessaire de connaître les positions réelles des caractères dans la table.

Exemple de jeu de caractères

									0	1	2	3	4	5	6	7	8	9	
			A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
R	S	T	U	V	W	X	Y	Z						a	b	c	d	e	f
g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Les chaînes de caractères

- Une chaîne de caractères est une suite de caractères.
- Différentes implémentations possibles, en général basées sur des tableaux de caractères.

Déclaration

Lexique des variables

maChaîne (chaîne) Une chaîne ...

- Lecture d'une chaîne :
  - directement avec l'instruction lire
    Exemple : *maChaîne* ← lire
  - Les caractères sont rangés dans l'ordre de lecture
- La lecture d'une chaîne s'arrête sur le premier séparateur rencontré :
  - espace et retour à la ligne

Manipulation de chaînes

- Plusieurs opérations sont possibles sur les chaînes :
  - déterminer la longueur d'une chaîne ;
  - copier une chaîne dans une autre ;
  - concaténer deux chaînes (l'une à la suite de l'autre) ;
  - comparer deux chaînes (ordre lexicographique) ;
  - rechercher une sous-chaîne dans une chaîne ;
  - interpréter le contenu d'une chaîne (entier, réel, ...) ;
  - etc.
- Ce sont en général des opérations coûteuses.

Plan

Recherche dans un tableau

- Trouver l'indice d'un élément particulier dans le tableau.
- Une première solution est de parcourir tout le tableau et de s'arrêter dès que l'on trouve la valeur cherchée.
- Il faut prévoir de s'arrêter à la fin du tableau si la valeur n'est pas présente.

Algorithme :

fonction recherche(**in** *tab* : tableau d'entiers, **in** *taille* : entier, **in** *val* : entier) : **ret** entier
Renvoie l'indice de *val* dans *tab* de taille *taille* si elle y est. Renvoie -1 sinon.

Lexique local des variables

i	(entier)	suite des indices des cases du tableau
trouvé	(entier)	indice de *val* dans *tab*

Algorithme simple

Algorithme (suite) :

Algorithme de recherche

$i \leftarrow 0$
$trouvé \leftarrow -1$
tant que $trouvé = -1 \wedge i < taille$ **faire**
  si $tab[i] = val$ **alors**
    $trouvé \leftarrow i$
  fsi
  $i \leftarrow i + 1$
ftant
retourner *trouvé*

Recherche dans un sous-tableau

Algorithme :

fonction recherche(**in** *tab* : tableau d'entiers, **in** *deb* : entier, **in** *fin* : entier, **in** *val* : entier) : **ret** entier

Renvoie l'indice de *val* dans *tab* entre les indices *deb* et *fin* si elle y est, ou -1 sinon.

Lexique local des variables

i	(entier)	suite des indices des cases du tableau
trouvé	(entier)	indice de *val* dans *tab*

Algorithme de recherche

$i \leftarrow deb$
$trouvé \leftarrow -1$
tant que $trouvé = -1 \wedge i \leqslant fin$ **faire**
  si $tab[i] = val$ **alors**
    $trouvé \leftarrow i$
  fsi
  $i \leftarrow i + 1$
ftant
retourner *trouvé*

Recherche dichotomique

- Ne s'applique qu'à des tableaux triés dont on connaît le sens du tri.
- Principe : réduire l'intervalle de recherche
  - choisir un élément dans le tableau : $t[i]$

Recherche dichotomique

- Ne s'applique qu'à des tableaux triés dont on connaît le sens du tri.
- Principe : réduire l'intervalle de recherche
  - choisir un élément dans le tableau : $t[i]$

  - si la valeur recherchée est égale à $t[i]$, on a trouvé !

Recherche dichotomique

- Ne s'applique qu'à des tableaux triés dont on connaît le sens du tri.
- Principe : réduire l'intervalle de recherche
  - choisir un élément dans le tableau : $t[i]$

éléments $\leqslant t[i]$	$t[i]$	éléments $\geqslant t[i]$

  - si la valeur recherchée est égale à $t[i]$, on a trouvé !

éléments $\leqslant t[i]$	$t[i]$	éléments $\geqslant t[i]$

  - si la valeur recherchée est $> t[i]$, on continue la recherche dans la partie des éléments $\geqslant t[i]$

éléments $\leqslant t[i]$	$t[i]$	éléments $\geqslant t[i]$

Recherche dichotomique

- Ne s'applique qu'à des tableaux triés dont on connaît le sens du tri.
- Principe : réduire l'intervalle de recherche
  - choisir un élément dans le tableau : $t[i]$

éléments $\leqslant t[i]$	$t[i]$	éléments $\geqslant t[i]$

  - si la valeur recherchée est égale à $t[i]$, on a trouvé !

éléments $\leqslant t[i]$	$t[i]$	éléments $\geqslant t[i]$

  - si la valeur recherchée est $> t[i]$, on continue la recherche dans la partie des éléments $\geqslant t[i]$

éléments $\leqslant t[i]$	$t[i]$	éléments $\geqslant t[i]$

  - si la valeur recherchée est $< t[i]$, on continue la recherche dans la partie des éléments $\leqslant t[i]$

éléments $\leqslant t[i]$	$t[i]$	éléments $\geqslant t[i]$

Recherche dichotomique

- Ne s'applique qu'à des tableaux triés dont on connaît le sens du tri.
- Principe : réduire l'intervalle de recherche
  - choisir un élément dans le tableau : $t[i]$

éléments $\leqslant t[i]$	$t[i]$	éléments $\geqslant t[i]$

  - si la valeur recherchée est égale à $t[i]$, on a trouvé !

éléments $\leqslant t[i]$	$t[i]$	éléments $\geqslant t[i]$

  - si la valeur recherchée est $> t[i]$, on continue la recherche dans la partie des éléments $\geqslant t[i]$

éléments $\leqslant t[i]$	$t[i]$	éléments $\geqslant t[i]$

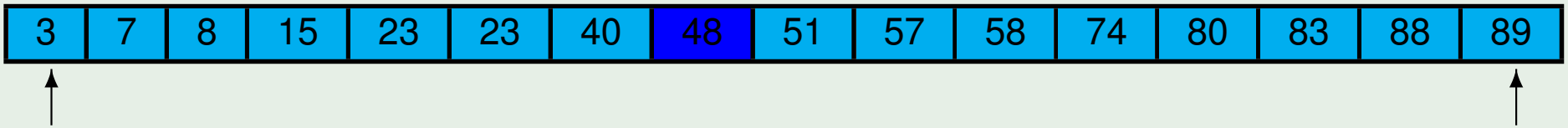
  - si la valeur recherchée est $< t[i]$, on continue la recherche dans la partie des éléments $\leqslant t[i]$

éléments $\leqslant t[i]$	$t[i]$	éléments $\geqslant t[i]$

Recherche dichotomique (suite)

- Arrêt de la recherche :
  - lorsque l'on trouve l'élément recherché ;
  - lorsque l'on arrive sur un intervalle de recherche vide.
- En général, on choisit l'élément du milieu pour la séparation.

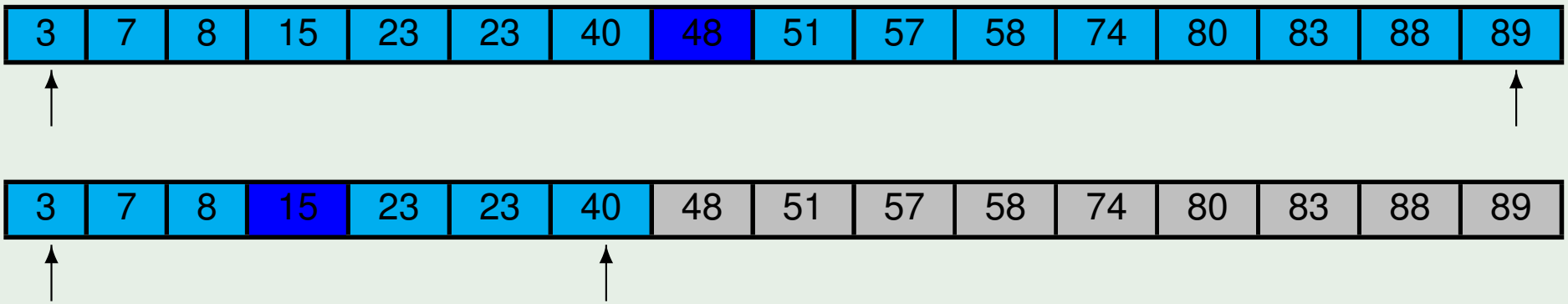
Exemple : recherche de 23



Recherche dichotomique (suite)

- Arrêt de la recherche :
  - lorsque l'on trouve l'élément recherché ;
  - lorsque l'on arrive sur un intervalle de recherche vide.
- En général, on choisit l'élément du milieu pour la séparation.

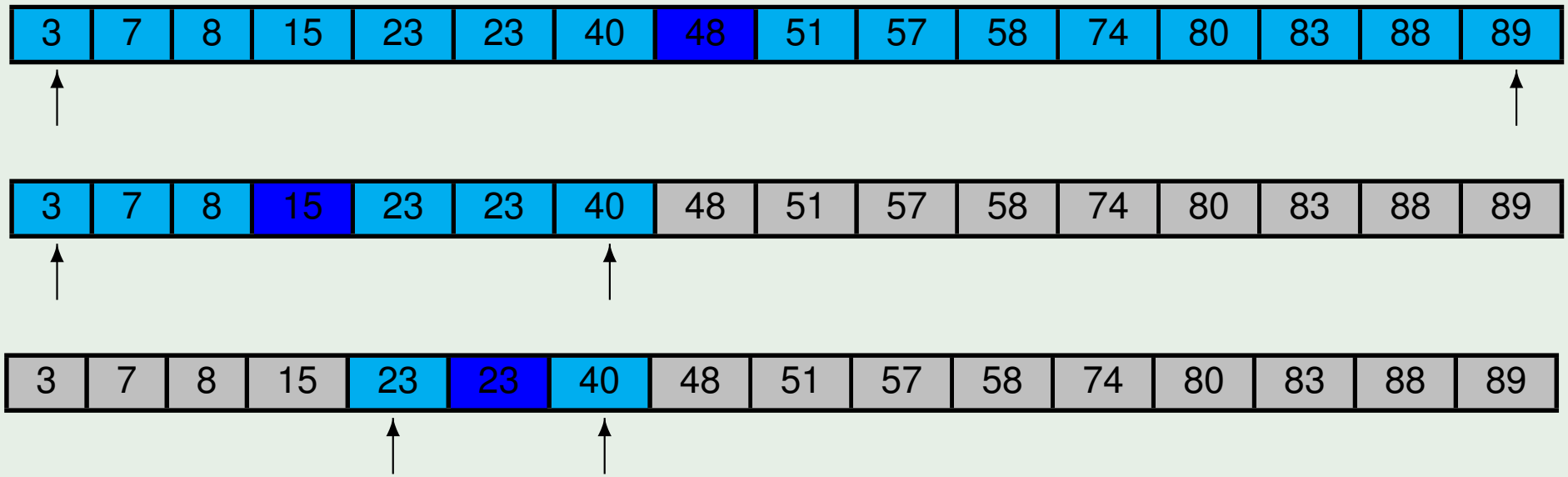
Exemple : recherche de 23



Recherche dichotomique (suite)

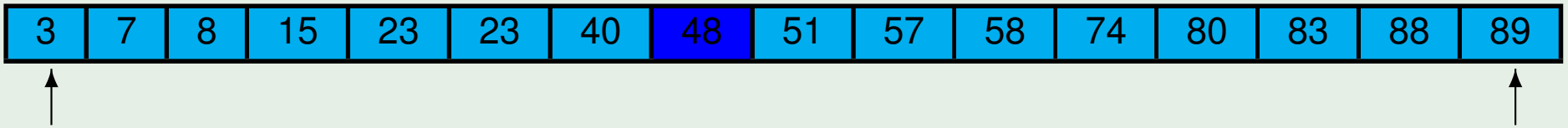
- Arrêt de la recherche :
  - lorsque l'on trouve l'élément recherché ;
  - lorsque l'on arrive sur un intervalle de recherche vide.
- En général, on choisit l'élément du milieu pour la séparation.

Exemple : recherche de 23



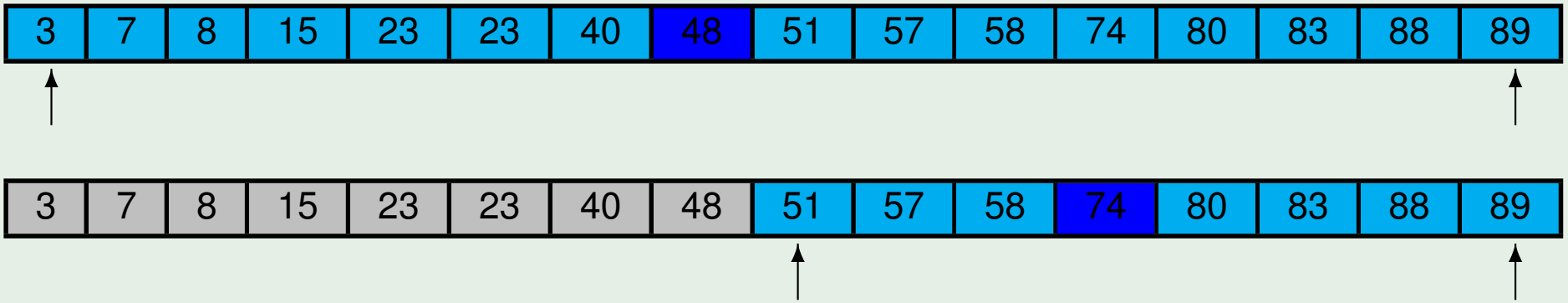
Exemple de recherche dichotomique

Exemple : recherche de 77



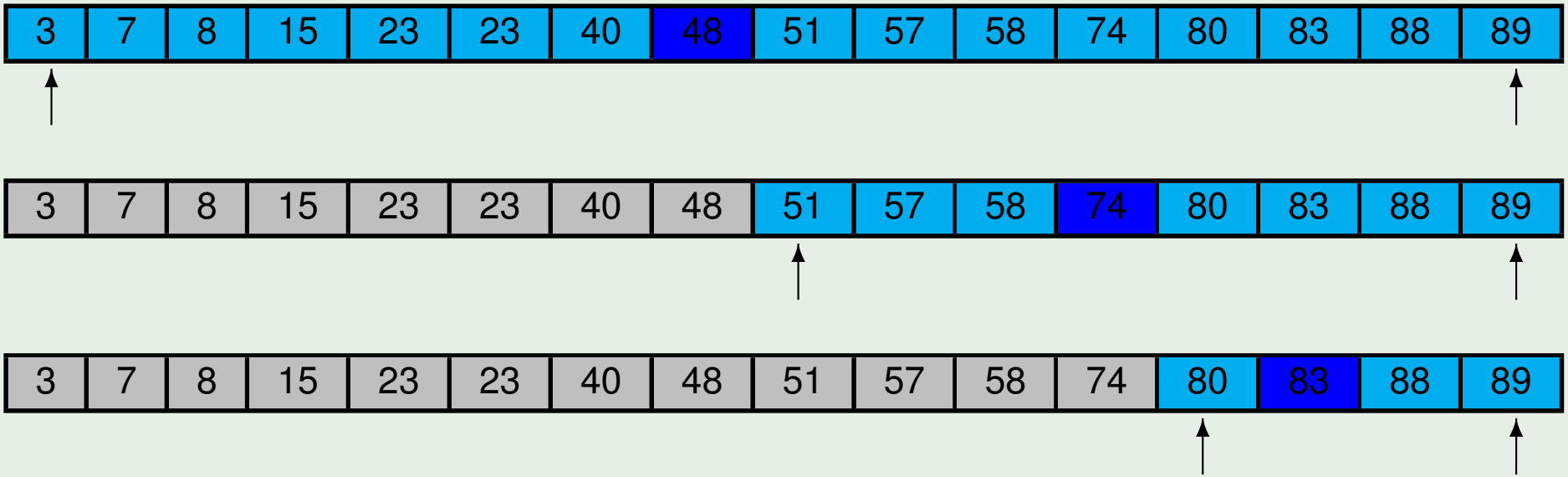
Exemple de recherche dichotomique

Exemple : recherche de 77



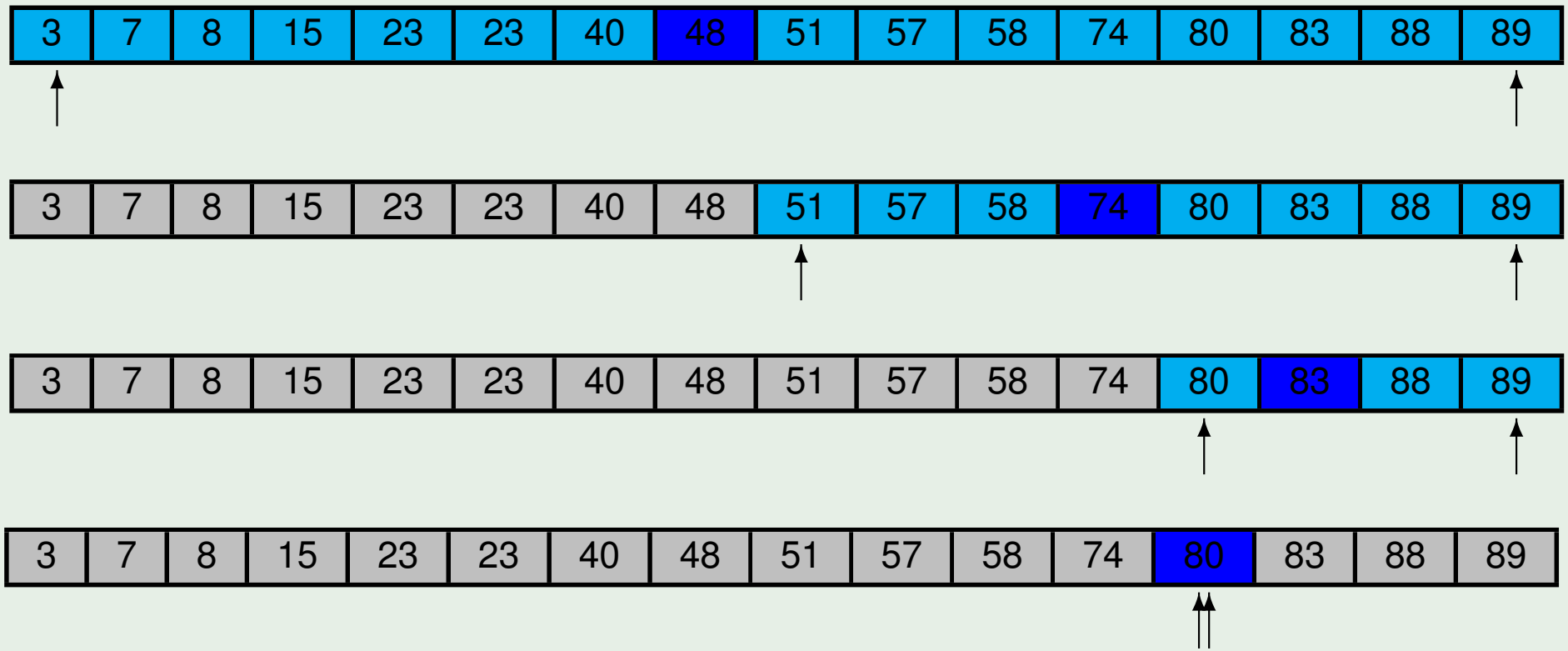
Exemple de recherche dichotomique

Exemple : recherche de 77



Exemple de recherche dichotomique

Exemple : recherche de 77



Écriture récursive

- Idée de l'algorithme :
  - Vérifier si l'on est sur la case recherchée, ou que l'on a terminé la recherche.
  - Si ce n'est pas le cas, regarder de quel côté peut se trouver la valeur.
  - Si on a terminé, on renvoie l'indice si c'est la bonne case, ou -1 sinon.

Algorithme :

fonction rechDichoRec(**in** *tab* : tableau d'entiers, **in** *deb* : entier, **in** *fin* : entier, **in** *val* : entier) : **ret** entier

Renvoie l'indice de *val* dans *tab* entre les indices *deb* et *fin* si elle y est, ou -1 sinon.

Lexique local des variables

milieu	(entier)	milieu de l'intervalle de recherche
trouvé	(entier)	indice de *val* dans *tab*

Algorithme récursif

Algorithme :

Algorithme de rechDichoRec

si $deb > fin$ **alors**
  $trouvé \leftarrow -1$
sinon
  $milieu \leftarrow (deb + fin)/2$
  si $tab[milieu] = val$ **alors**
    $trouvé \leftarrow milieu$
  sinon si $val < tab[milieu]$ **alors**
    $trouvé \leftarrow \text{rechDichoRec}(tab, deb, milieu - 1, val)$
  sinon
    $trouvé \leftarrow \text{rechDichoRec}(tab, milieu + 1, fin, val)$
  fsi
fsi
retourner *trouvé*

Écriture itérative

- Idée de l'algorithme :
  - Vérifier si l'on est sur la case recherchée, ou que l'on a terminé la recherche.
  - Si ce n'est pas le cas, regarder de quel côté peut se trouver la valeur.
  - Si on a terminé, on renvoie l'indice si c'est la bonne case, ou -1 sinon.

Algorithme :

fonction rechDicholter(**in** *tab* : tableau d'entiers, **in** *deb* : entier, **in** *fin* : entier, **in** *val* : entier) : **ret** entier

Renvoie l'indice de *val* dans *tab* entre les indices *deb* et *fin* si elle y est, ou -1 sinon.

Lexique local des variables

min	(entier)	début de l'intervalle de recherche
max	(entier)	fin de l'intervalle de recherche
milieu	(entier)	milieu de l'intervalle de recherche
trouvé	(entier)	indice de *val* dans *tab*

Algorithme itératif

Algorithme :

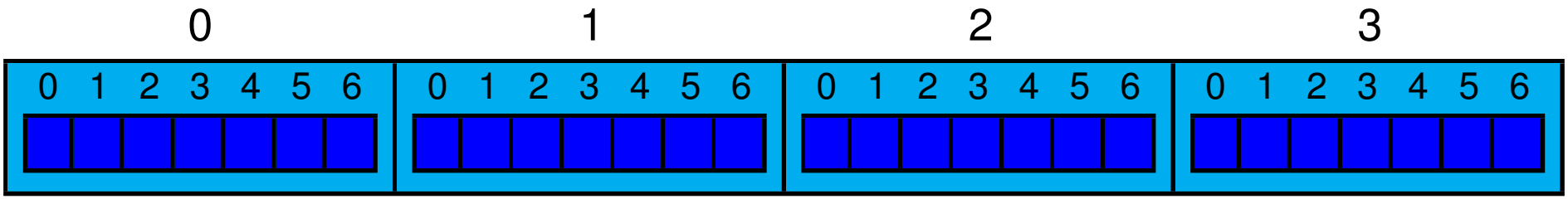
Algorithme de rechDicholter

trouvé $\leftarrow -1$
min $\leftarrow$ *deb*
max $\leftarrow$ *fin*
tant que *trouvé* $= -1 \wedge$ *min* $\leqslant$ *max* **faire**
  milieu $\leftarrow (min + max)/2$
  si *tab*$[milieu] =$ *val* **alors**
    trouvé $\leftarrow$ *milieu*
  sinon si *val* $<$ *tab*$[milieu]$ **alors**
    max $\leftarrow$ *milieu* $- 1$
  sinon
    min $\leftarrow$ *milieu* $+ 1$
  fsi
ftant
retourner *trouvé*

Plan

Tableaux à 2 dimensions

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



- **Exercice :** simuler un tel tableau avec un tableau simple de $4 \times 7 = 28$ éléments. Comment convertir les indices ?

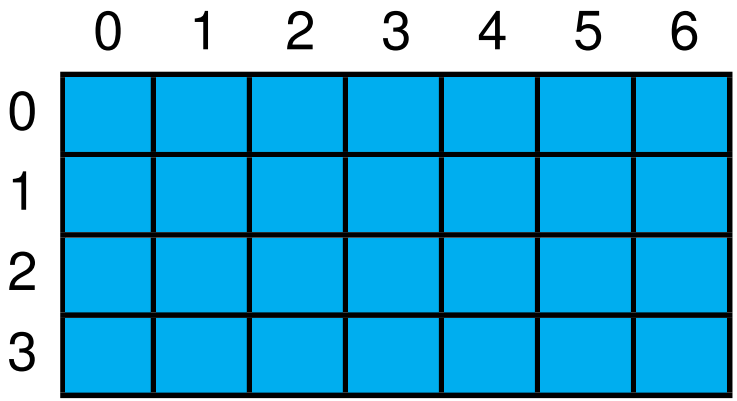
Représentation matricielle

Déclaration (exemple)

Lexique des variables

table (tableau $[4,7]$ d'entiers) Tableau de 4 lignes et 7 colonnes

- Accès à un élément : $nomDuTableau[indice_1, indice_2]$, avec
  - $indice_1$ entre 0 et $dimension_1 - 1$
  - $indice_2$ entre 0 et $dimension_2 - 1$
- Représentation matricielle de *table* :



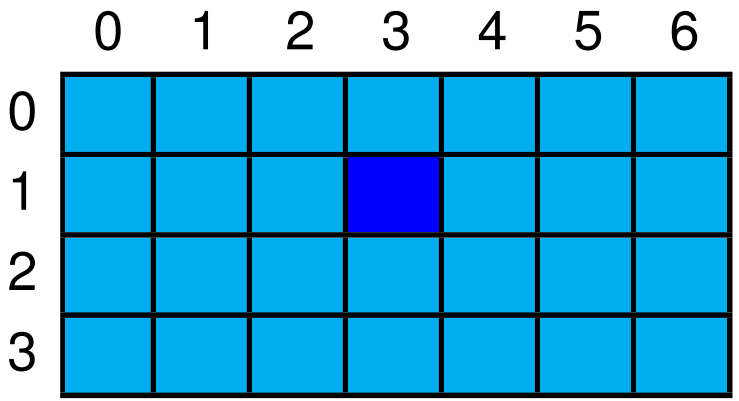
Représentation matricielle

Déclaration (exemple)

Lexique des variables

table (tableau $[4,7]$ d'entiers) Tableau de 4 lignes et 7 colonnes

- Accès à un élément : $nomDuTableau[indice_1, indice_2]$, avec
  - $indice_1$ entre 0 et $dimension_1 - 1$
  - $indice_2$ entre 0 et $dimension_2 - 1$
- Représentation matricielle de *table* :



$table[1,3]$
2^e ligne,
4^e colonne

Tableaux à deux dimensions en Java

Il n'y a pas de tableaux à plusieurs dimensions en Java, uniquement des tableaux de tableaux.

Déclaration

- Forme générale : `<type>[][] <nom>;`
- Exemple : `int[][] table;`

Tableaux à deux dimensions en Java

Il n'y a pas de tableaux à plusieurs dimensions en Java, uniquement des tableaux de tableaux.

Déclaration

- Forme générale : `<type>[][] <nom>;`
- Exemple : `int[][] table;`

Création/initialisation

- Comme pour les tableaux simples, il faut créer le tableau :
  Exemple : `int[][] table = new int[2][4];`
- On peut aussi combiner allocation et initialisation (variante) :
  Exemple :

```
int[][] table = { {5, 9, -4, 2}, {12, 17, 2, 7} };
```

Tableaux à deux dimensions en Java

Il n'y a pas de tableaux à plusieurs dimensions en Java, uniquement des tableaux de tableaux.

Déclaration

- Forme générale : `<type>[][] <nom>;`
- Exemple : `int[][] table;`

Création/initialisation

- Comme pour les tableaux simples, il faut créer le tableau :
  Exemple : `int[][] table = new int[2][4];`
- On peut aussi combiner allocation et initialisation (variante) :
  Exemple :
  ```
  int[][] table = { {5, 9, -4, 2}, {12, 17, 2, 7} };
  ```

Utilisation

- Les indices sont placés entre crochets, et vont de 0 à $taille_k - 1$.
- Exemple : `a = table[0][3] + 6; table[1][2] = a + a;`
- Attention à ne pas *sortir* des tableaux !

Exemple

- On veut stocker et manipuler les notes des étudiants.
  $\Rightarrow$ Utilisation d'un tableau 2D contenant :
  - un étudiant par ligne ;
  - une matière par colonne ;
  - dimensions $NBE \times NBM$ fixées : NBE et NBM définies en constantes.
- Tableau $tNotes[NBE, NBM]$:

	Matière 1 (indice 0)	...	Matière NBM (indice $NBM-1$)
Étudiant 1 (indice 0)		...	
...	...	...	...
Étudiant NBE (indice $NBE-1$)		...	

Exemple (suite)

On utilise deux tableaux de chaînes de caractères pour les noms des matières, et les noms des étudiants.

Lexique des variables

tEtu	(tableau $[NBE]$ de chaînes)	liste des noms de étudiants	...
tMat	(tableau $[NBM]$ de chaînes)	liste des noms des matières	...

Fonctions de remplissage des tableaux

Lexique des fonctions

fonction lireListe(**out** *tab* : tableau de chaînes, **in** *nbElem* : entier) : **vide**
Lit *nbElem* chaînes et les place dans le tableau *tab*.

fonction lireNotes(**out** *tNotes* : tableau de tableaux de réels,
in *tEtu* : tableau de chaînes, **in** *tMat* : tableau de chaînes,
in *nbE* : entier, **in** *nbM* : entier) : **vide**
Lit $nbE \times nbM$ notes et les place dans le tableau *tNotes*.

Fonction lireListe

Algorithme

fonction lireListe(**out** *tab* : tableau de chaînes, **in** *nbElem* : entier) : **vide**

Lit *nbElem* chaînes et les place dans le tableau *tab*.

Lexique local des variables

i (entier) compteur de la boucle de lecture

Algorithme de lireListe

pour *i* **de** 0 **à** $nbElem - 1$ **faire**
  $tab[i] \leftarrow$ lire
fpour

- Cette fonction est utilisable pour la lecture des noms des étudiants, et pour la lecture des noms des matières.

Fonction lireNotes

fonction lireNotes(**out** *tNotes* : tableau de tableaux de réels,
in *tEtu* : tableau de chaînes, **in** *tMat* : tableau de chaînes,
in *nbE* : entier, **in** *nbM* : entier) : **vide**

Lit $nbE \times nbM$ notes et les place dans le tableau *tNotes*.

Lexique local des variables

iEtu	(entier)	indice de l'étudiant courant
iMat	(entier)	indice de la matière courante

Algorithme de lireNotes

pour *iEtu* **de** 0 **à** $nbE - 1$ **faire**
  pour *iMat* **de** 0 **à** $nbM - 1$ **faire**
    répéter
      écrire "`Entrez la note de `", $tEtu[iEtu]$, "` en `", $tMat[iMat]$, "`: `"
      $tNotes[iEtu, iMat] \leftarrow$ lire
    tant que $\neg(tNotes[iEtu, iMat] \geqslant 0 \wedge tNotes[iEtu, iMat] \leqslant 20)$
  fpour
fpour

Calcul des moyennes

Algorithme

Résultat

Lecture d'une liste de matières, d'une liste d'étudiants et de leurs notes, puis affichage des moyennes par étudiant et par matière.

Lexique des constantes

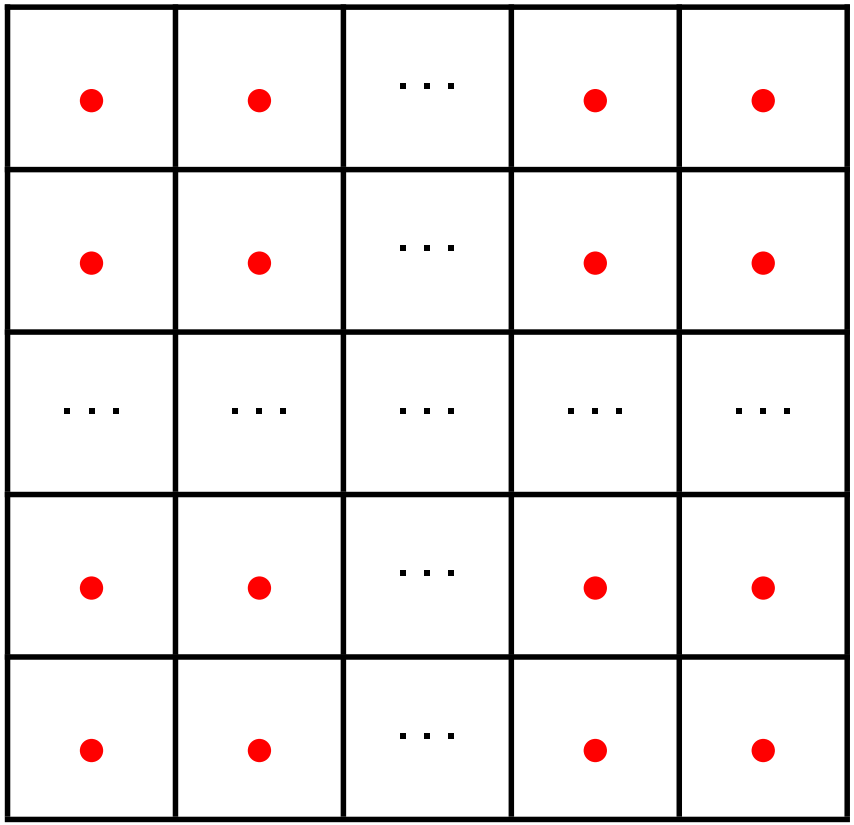
NBE	(entier)	$= 110$	nombre d'étudiants
NBM	(entier)	$= 15$	nombre de matières

Lexique des variables

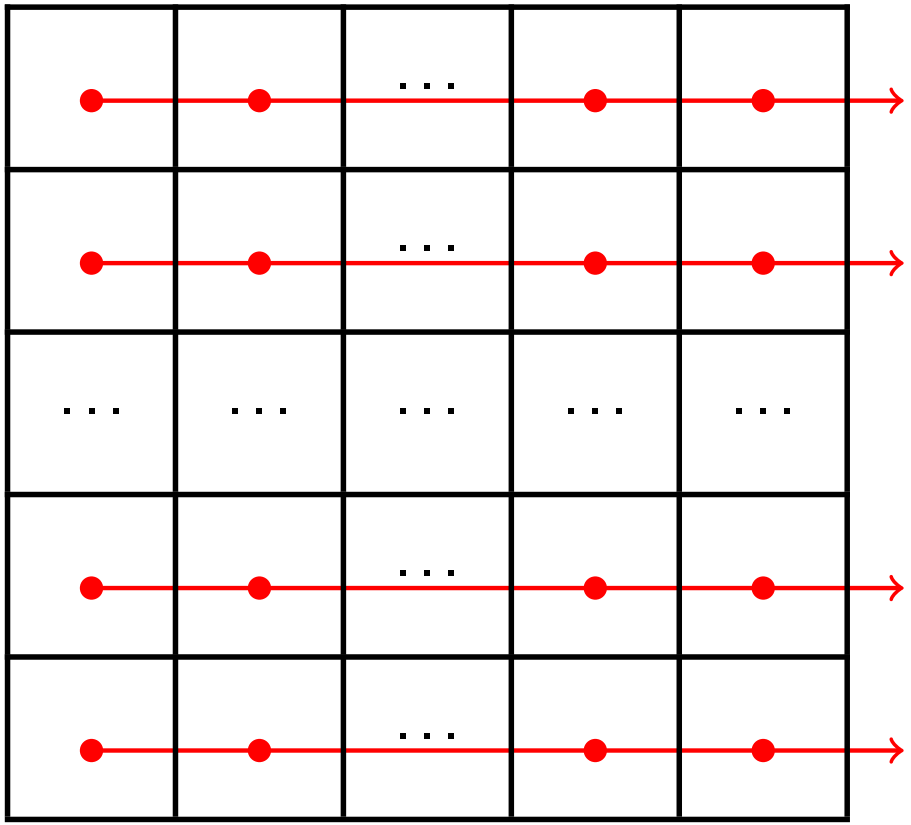
iEtu	(entier)	indice de l'étudiant courant
iMat	(entier)	indice de la matière courante
totEtu	(réel)	somme des notes pour un étudiant
totMat	(réel)	somme des notes pour une matière
tEtu	(tableau $[NBE]$ de chaînes)	liste des noms des étudiants
tMat	(tableau $[NBM]$ de chaînes)	liste des noms des matières
tNotes	(tableau $[NBE, NBM]$ de réels)	notes par étudiant et par matière

NB : toutes les variables ont le rôle « INTERMÉDIAIRE ».

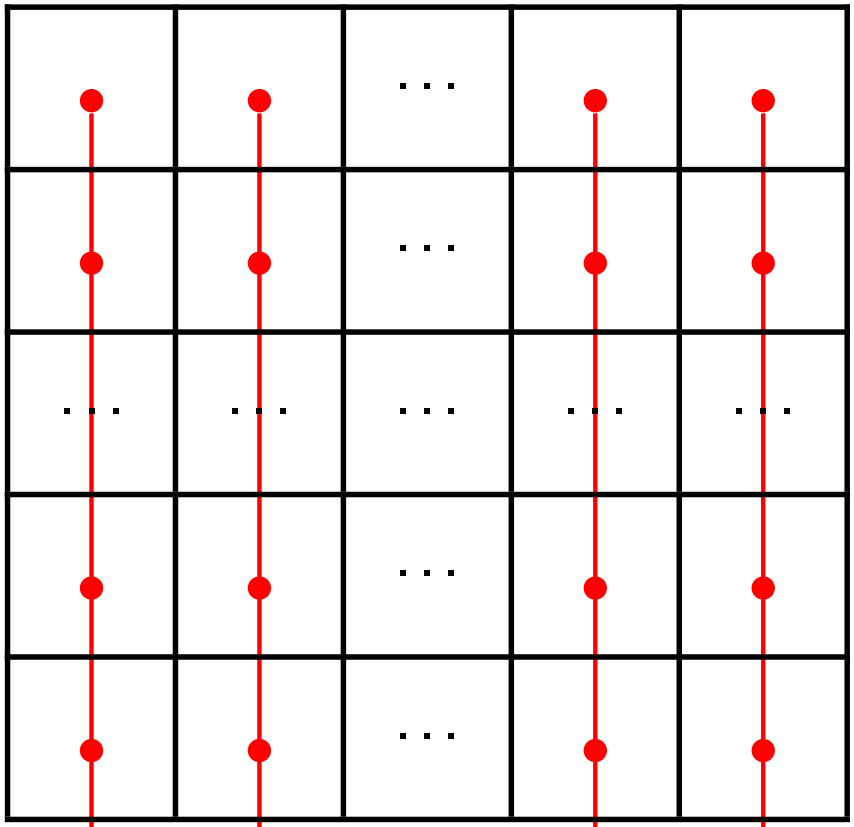
Calcul des moyennes (illustration)



Calcul des moyennes (illustration)



Calcul des moyennes (illustration)



moyenne pour la matière *NBM*

moyenne pour la matière $NBM - 1$

...

moyenne pour la matière 2

moyenne pour la matière 1

Calcul des moyennes (algorithme)

Algorithme

lireListe(*tMat*, *NBM*)
lireListe(*tEtu*, *NBE*)
lireNotes(*tNotes*, *tEtu*, *tMat*, *NBE*, *NBM*)

pour *iEtu* **de** 0 **à** $NBE-1$ **faire**
  totEtu ← 0
  pour *iMat* **de** 0 **à** $NBM-1$ **faire**
    $totEtu \leftarrow totEtu + tNotes[iEtu, iMat]$
  fpour
  écrire "La moyenne de l'étudiant ", $tEtu[iEtu]$, " est ", $totEtu/NBM$
fpour

pour *iMat* **de** 0 **à** $NBM-1$ **faire**
  totMat ← 0
  pour *iEtu* **de** 0 **à** $NBE-1$ **faire**
    $totMat \leftarrow totMat + tNotes[iEtu, iMat]$
  fpour
  écrire "La moyenne pour la matière ", $tMat[iMat]$, " est ", $totMat/NBE$
fpour

Calcul des moyennes (algorithme)

Algorithme

lireListe(*tMat*, *NBM*)
lireListe(*tEtu*, *NBE*)
lireNotes(*tNotes*, *tEtu*, *tMat*, *NBE*, *NBM*)

pour *iEtu* **de** 0 **à** *NBE* − 1 **faire**
  totEtu ← 0
  pour *iMat* **de** 0 **à** *NBM* − 1 **faire**
    totEtu ← *totEtu* + *tNotes*[*iEtu*, *iMat*]
  fpour
  écrire "La moyenne de l'étudiant ", *tEtu*[*iEtu*], " est ", *totEtu*/*NBM*
fpour

pour *iMat* **de** 0 **à** *NBM* − 1 **faire**
  totMat ← 0
  pour *iEtu* **de** 0 **à** *NBE* − 1 **faire**
    totMat ← *totMat* + *tNotes*[*iEtu*, *iMat*]
  fpour
  écrire "La moyenne pour la matière ", *tMat*[*iMat*], " est ", *totMat*/*NBE*
fpour

L'ordre des boucles change !

Calcul des moyennes (algorithme)

Algorithme

lireListe(*tMat*, *NBM*)
lireListe(*tEtu*, *NBE*)
lireNotes(*tNotes*, *tEtu*, *tMat*, *NBE*, *NBM*)

pour *iEtu* **de** 0 **à** *NBE* − 1 **faire**
  totEtu ← 0
  pour *iMat* **de** 0 **à** *NBM* − 1 **faire**
    totEtu ← *totEtu* + *tNotes*[*iEtu*, *iMat*]
  fpour
  écrire "La moyenne de l'étudiant ", *tEtu*[*iEtu*], " est ", *totEtu*/*NBM*
fpour

pour *iMat* **de** 0 **à** *NBM* − 1 **faire**
  totMat ← 0
  pour *iEtu* **de** 0 **à** *NBE* − 1 **faire**
    totMat ← *totMat* + *tNotes*[*iEtu*, *iMat*]
  fpour
  écrire "La moyenne pour la matière ", *tMat*[*iMat*], " est ", *totMat*/*NBE*
fpour

L'ordre des boucles change !

L'ordre des indices du tableau ne change pas !

Implémentation en Java

```
import java.util.*;

class Notes {

    static final Scanner input = new Scanner(System.in);

    static final int NBE = 110;
    static final int NBM = 15;

    static void lireListe(String[] tab, int nbElem) {
        for (int i = 0; i < nbElem; ++i)
            tab[i] = input.nextLine();
    }

    static void lireNotes(double[][] tNotes, String[] tEtu, String[] tMat,
                          int nbE, int nbM) {
        for (int iEtu = 0; iEtu < nbE; ++iEtu)
            for (int iMat = 0; iMat < nbM; ++iMat)
                do {
                    System.out.print("Entrez la note de " + tEtu[iEtu] +
                                     " en " + tMat[iMat] + ": ");
                    tNotes[iEtu][iMat] = input.nextDouble();
                } while (!(tNotes[iEtu][iMat] >= 0 && tNotes[iEtu][iMat] <= 20));
    }
```

Implémentation en Java (suite)

```
    public static void main(String[] args) {
        String [] tEtu = new String[NBE];
        String [] tMat = new String[NBM];
        double[][] tNotes = new double[NBE][NBM];

        System.out.println("Entrez " + NBM + " matières:");
         lireListe (tMat, NBM);
        System.out.println("Entrez " + NBE + " étudiants:");
         lireListe (tEtu, NBE);
        System.out.println("Entrez les notes par matière:");
        lireNotes(tNotes, tEtu, tMat, NBE, NBM);

        for (int iEtu = 0 ; iEtu < NBE ; ++iEtu) {
            double totEtu = 0;
            for (int iMat = 0 ; iMat < NBM ; ++iMat)
                totEtu += tNotes[iEtu][iMat];
            System.out.println("La moyenne de l'étudiant " + tEtu[iEtu] +
                               " est " + (totEtu / NBM));
        }

        for (int iMat = 0 ; iMat < NBM ; ++iMat) {
            double totMat = 0;
            for (int iEtu = 0 ; iEtu < NBE ; ++iEtu)
                totMat += tNotes[iEtu][iMat];
            System.out.println("La moyenne pour la matière " + tMat[iMat] +
                               " est " + (totMat / NBE));
        }
    }
}
```

Exemple : sudoku

- Compléter une grille de 9×9 cases avec les chiffres de 1 à 9.
- Chacun des chiffres de 1 à 9 doit apparaître exactement une fois par ligne, par colonne et par carré de 3×3.
- Exemple :

8	5	1	2	9	7	4	3	6
9	3	2	6	4	5	8	1	7
6	4	7	3	8	1	9	2	5
1	6	5	9	3	4	2	7	8
7	9	4	8	1	2	6	5	3
3	2	8	5	7	6	1	9	4
4	1	9	7	5	8	3	6	2
2	7	3	4	6	9	5	8	1
5	8	6	1	2	3	7	4	9

- On souhaite une fonction permettant de vérifier si une grille de 9×9 nombres forme une grille de sudoku valide.

Idée de l'algorithme

Idée générale

- On commence par définir une fonction verif9 qui :
  - prend en paramètre un tableau de 9 nombres entiers ;
  - retourne vrai si le tableau contient les chiffres de 1 à 9, tous distincts ;
  - retourne faux sinon.
- On utilise ensuite la fonction verif9 pour vérifier que chaque ligne, colonne et carré de 3×3 contient tous les nombres de 1 à 9.

Idée de l'algorithme de verif9

- La liste de nombre est parcourue.
- La fonction retourne faux si :
  - un des nombres n'est pas compris entre 1 et 9 ;
  - un nombre apparaît plusieurs fois.
- Pour cette dernière vérification, on utilise un tableau de 9 booléens permettant de retenir si un nombre a déjà été vu ou non.

Algorithme de verif9

fonction verif9(**in** *liste* : tableau $[9]$ d'entiers) : **ret** booléen

Lexique local des variables

present	(tableau $[9]$ de booléens)	*present*$[k-1]$ est vrai si k est dans *liste*
valide	(booléen)	le résultat de la fonction
i	(entier)	indice de parcours de la liste

Algorithme de verif9

present $\leftarrow \{$faux, faux, faux, faux, faux, faux, faux, faux, faux$\}$
valide $\leftarrow$ vrai
$i \leftarrow 0$
tant que *valide* $\wedge$ $i < 9$ **faire**
  si *liste*$[i] < 1 \vee$ *liste*$[i] > 9 \vee$ *present*$[$*liste*$[i]-1]$ **alors**
    valide $\leftarrow$ faux
  sinon
    present$[$*liste*$[i]-1] \leftarrow$ vrai
  fsi
  $i \leftarrow i+1$
ftant
retourner *valide*

Algorithme de verif_sudoku

Partie 1/4 : en-tête et lexiques

fonction verif_sudoku(**in** *grille* : tableau $[9,9]$ d'entiers) : **ret** booléen

Retourne vrai si le tableau *grille* correspond à une grille de sudoku valide. Retourne faux sinon.

Lexique local des fonctions

fonction verif9(**in** *liste* : tableau $[9]$ d'entiers) : **ret** booléen

Retourne vrai si le tableau *liste* contient exactement les nombres de 1 à 9. Retourne faux sinon.

Lexique local des variables

valide	(booléen)	le résultat de la fonction
tmp	(tableau $[9]$ d'entiers)	liste temporaire de 9 entiers
lig	(entier)	indice de parcours des lignes de nombres
col	(entier)	indice de parcours des colonnes de nombres
carlig	(entier)	indice de parcours des lignes de carrés de 3×3 nombres
carcol	(entier)	indice de parcours des colonnes de carrés de 3×3 nombres
ind	(entier)	indice de parcours de *tmp*

Algorithme de verif_sudoku

Partie 2/4 : initialisation et vérification des lignes

Algorithme

$valide \leftarrow$ vrai

$lig \leftarrow 0$
tant que $valide \wedge lig < 9$ **faire**
  `// copie des nombres de la ligne lig`
  pour col **de** 0 **à** $9-1$ **faire**
    $tmp[col] \leftarrow grille[lig, col]$
  fpour
  $valide \leftarrow$ verif9(tmp)
  $lig \leftarrow lig + 1$
ftant

...

Algorithme de verif_sudoku

Partie 3/4 : vérification des colonnes

...

$col \leftarrow 0$
tant que $valide \wedge col < 9$ **faire**
    `// copie des nombres de la colonne col`
    pour lig **de** 0 **à** $9-1$ **faire**
        $tmp[lig] \leftarrow grille[lig, col]$
    fpour
    $valide \leftarrow \text{verif9}(tmp)$
    $col \leftarrow col + 1$
ftant

...

Algorithme de verif_sudoku

Partie 4/4 : vérification des carrés de 3×3, et retour

```
...
carlig ← 0
tant que valide ∧ carlig < 9 faire
    carcol ← 0
    tant que valide ∧ carcol < 9 faire
        // copie des nombres du carré qui commence en (carlig, carcol)
        ind ← 0
        pour lig de carlig à carlig + 2 faire
            pour col de carcol à carcol + 2 faire
                tmp[ind] ← grille[lig, col]
                ind ← ind + 1
            fpour
        fpour
        valide ← verif9(tmp)
        carcol ← carcol + 3
    ftant
    carlig ← carlig + 3
ftant
retourner valide
```

Plan

Tri sur les vecteurs

- La notion d'ordre n'est pas implicite dans un tableau.
- Il est souvent nécessaire d'avoir des valeurs triées.
  - Exemple : recherche dichotomique.
- Nous allons voir les tris en ordre croissant suivants :
  - tri par insertion ;
  - tri par sélection-permutation ;
  - tri à bulles ;
  - double tri à bulles.
- On peut facilement déduire les versions décroissantes.
- Ces tris simples sont peu efficaces sur de grands vecteurs.
- On utilise d'autre algorithmes comme, par exemple :
  - tri fusion ;
  - tri rapide (quick sort).

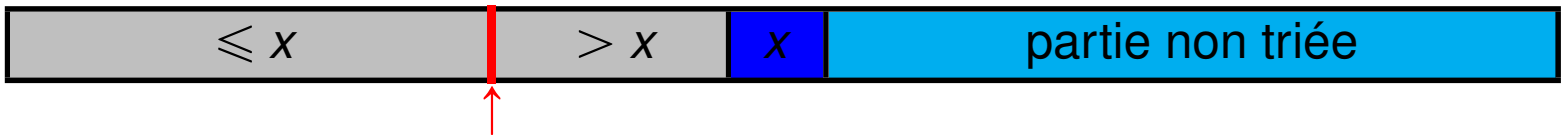
Tri par insertion

- Utilisé pour trier les cartes que l'on a en main.
- Principe :
  - on prend un élément et on recherche sa place dans la partie déjà triée puis on l'insère à cet endroit, et ainsi de suite...
- Application aux tableaux :
  - Deux parties dans le tableau :

  - On procède itérativement en prenant à chaque fois un élément de la partie non triée et en l'insérant à la bonne place dans la partie triée.
    $\Rightarrow$ La proportion de partie triée/partie non triée va donc évoluer.
- À chaque fois, on prend le 1er élément de la partie non triée comme élément à insérer dans la partie déjà triée.

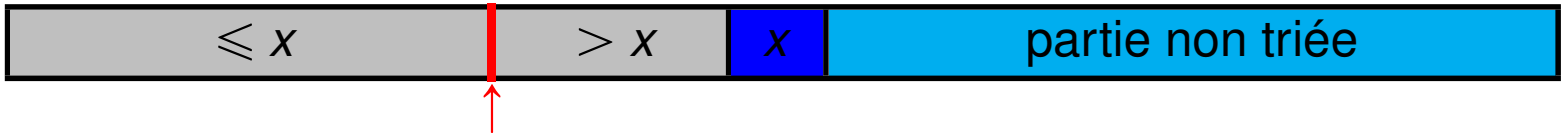
Étapes d'insertion d'un élément

- L'insertion de l'élément courant se fait en 4 étapes :
  - Trouver le point d'insertion dans la partie triée.

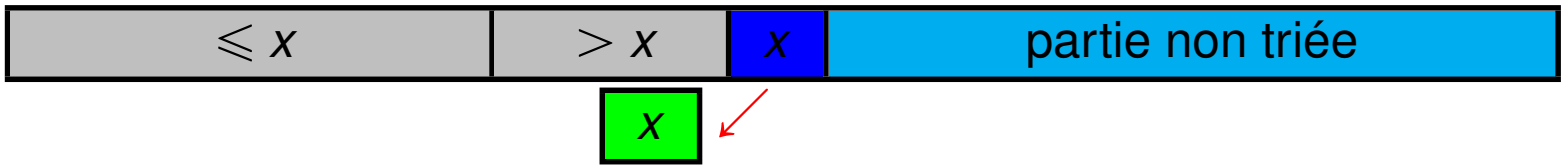


Étapes d'insertion d'un élément

- L'insertion de l'élément courant se fait en 4 étapes :
  - Trouver le point d'insertion dans la partie triée.

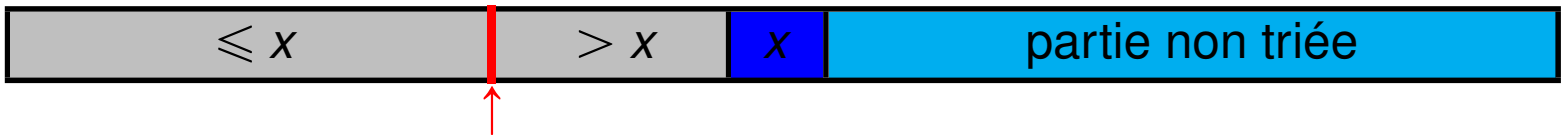


  - Recopier l'élément courant.

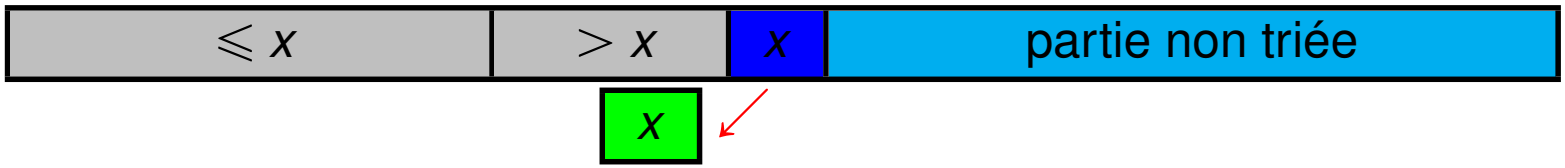


Étapes d'insertion d'un élément

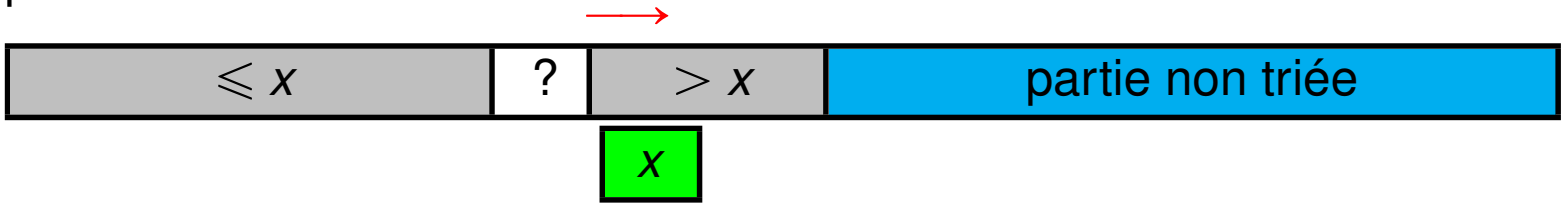
- L'insertion de l'élément courant se fait en 4 étapes :
  - Trouver le point d'insertion dans la partie triée.



  - Recopier l'élément courant.

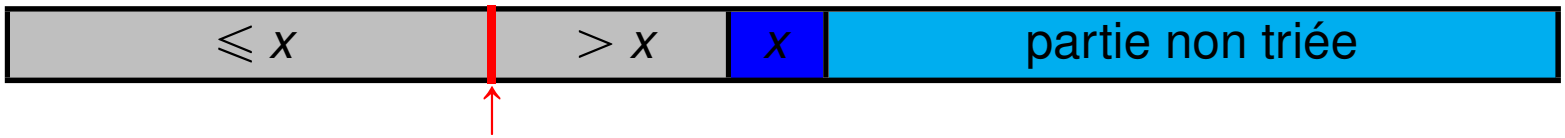


  - Décaler d'une case vers la droite les éléments déjà triés qui sont après le point d'insertion.

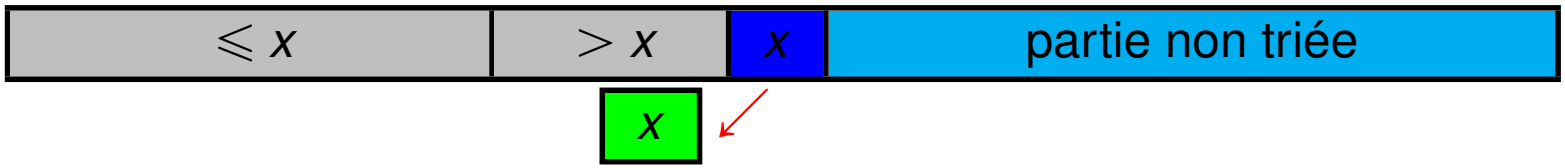


Étapes d'insertion d'un élément

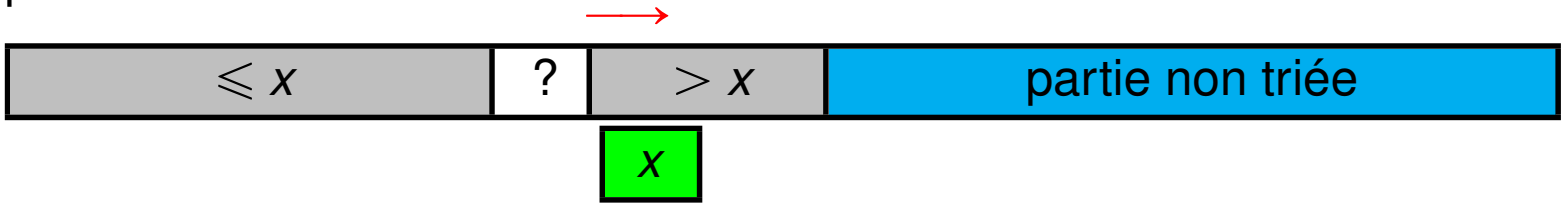
- L'insertion de l'élément courant se fait en 4 étapes :
  - Trouver le point d'insertion dans la partie triée.



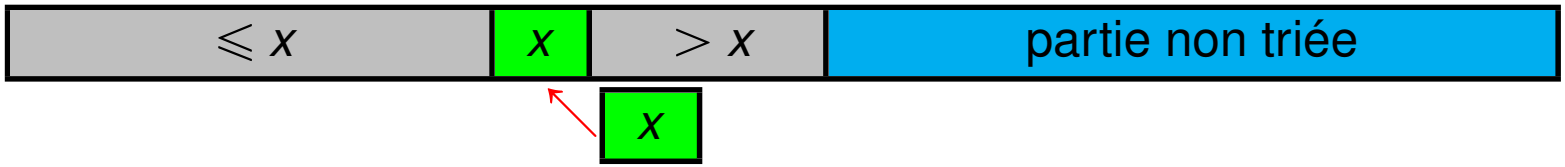
  - Recopier l'élément courant.



  - Décaler d'une case vers la droite les éléments déjà triés qui sont après le point d'insertion.



  - Placer l'élément courant à l'emplacement ainsi libéré.



Remarques

- On voit que la nouvelle partie grisée est toujours triée :

- Au départ, on a logiquement :

  partie triée vide
  partie non triée tout le tableau

- Mais on peut gagner une étape en prenant :

  partie triée 1^{er} élément du tableau
  partie non triée le reste du tableau

- La recherche du point d'insertion et le décalage des éléments peuvent se faire en même temps.
- On termine quand la partie non triée devient vide :
  quand le dernier élément du tableau a été inséré dans la partie triée.

Algorithme

fonction triInsert(**in-out** *tab* : tableau d'entiers, **in** *taille* : entier) : **vide**

Trie le tableau *tab* de *taille* éléments en utilisant le tri par insertion.

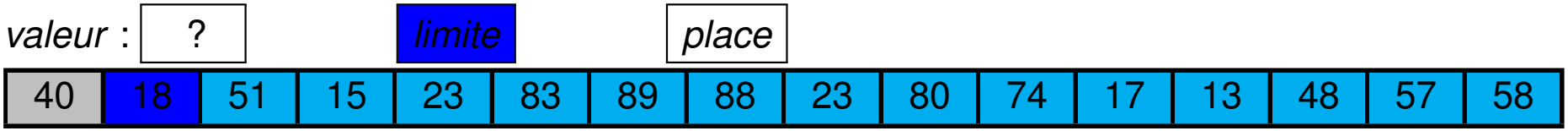
Lexique local des variables

valeur	(entier)	valeur à insérer dans la partie triée
limite	(entier)	indice du 1er élément de la partie non triée
place	(entier)	indice d'insertion de la valeur dans la partie triée

Algorithme de triInsert

pour *limite* **de** 1 **à** $taille - 1$ **faire**
  $valeur \leftarrow tab[limite]$
  $place \leftarrow limite$
  tant que $place > 0 \wedge tab[place - 1] > valeur$ **faire**
    $tab[place] \leftarrow tab[place - 1]$
    $place \leftarrow place - 1$
  ftant
  $tab[place] \leftarrow valeur$
fpour

Déroulement de l'algorithme



Déroulement de l'algorithme

valeur : 18 *limite* *place*

40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58

Déroulement de l'algorithme

valeur : 18 *limite* *place*

40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58

Déroulement de l'algorithme

valeur : 18 *limite* *place*

40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58

Déroulement de l'algorithme

valeur : 51 *limite* *place*

40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58

Déroulement de l'algorithme

valeur : 51 *limite* *place*

40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58

Déroulement de l'algorithme

valeur : 15 *limite* *place*

40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58

Déroulement de l'algorithme

valeur : 15 *limite* *place*

40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	51	23	83	89	88	23	80	74	17	13	48	57	58

Déroulement de l'algorithme

valeur : 15 | *limite* | *place*

40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	51	23	83	89	88	23	80	74	17	13	48	57	58
18	40	40	51	23	83	89	88	23	80	74	17	13	48	57	58

Déroulement de l'algorithme

valeur : 15 *limite* *place*

40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	51	23	83	89	88	23	80	74	17	13	48	57	58
18	40	40	51	23	83	89	88	23	80	74	17	13	48	57	58
18	18	40	51	23	83	89	88	23	80	74	17	13	48	57	58

Déroulement de l'algorithme

valeur : 15 | *limite* | *place*

40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	51	23	83	89	88	23	80	74	17	13	48	57	58
18	40	40	51	23	83	89	88	23	80	74	17	13	48	57	58
18	18	40	51	23	83	89	88	23	80	74	17	13	48	57	58
15	18	40	51	23	83	89	88	23	80	74	17	13	48	57	58

Déroulement de l'algorithme

valeur : 23 *limite* *place*

40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	51	23	83	89	88	23	80	74	17	13	48	57	58
18	40	40	51	23	83	89	88	23	80	74	17	13	48	57	58
18	18	40	51	23	83	89	88	23	80	74	17	13	48	57	58
15	18	40	51	23	83	89	88	23	80	74	17	13	48	57	58
15	18	40	51	23	83	89	88	23	80	74	17	13	48	57	58

Déroulement de l'algorithme

valeur : 23 *limite* *place*

40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	51	23	83	89	88	23	80	74	17	13	48	57	58
18	40	40	51	23	83	89	88	23	80	74	17	13	48	57	58
18	18	40	51	23	83	89	88	23	80	74	17	13	48	57	58
15	18	40	51	23	83	89	88	23	80	74	17	13	48	57	58
15	18	40	51	23	83	89	88	23	80	74	17	13	48	57	58
15	18	40	51	51	83	89	88	23	80	74	17	13	48	57	58

Déroulement de l'algorithme

valeur : 23 | *limite* | *place*

40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	51	23	83	89	88	23	80	74	17	13	48	57	58
18	40	40	51	23	83	89	88	23	80	74	17	13	48	57	58
18	18	40	51	23	83	89	88	23	80	74	17	13	48	57	58
15	18	40	51	23	83	89	88	23	80	74	17	13	48	57	58
15	18	40	51	23	83	89	88	23	80	74	17	13	48	57	58
15	18	40	51	51	83	89	88	23	80	74	17	13	48	57	58
15	18	40	40	51	83	89	88	23	80	74	17	13	48	57	58

Déroulement de l'algorithme

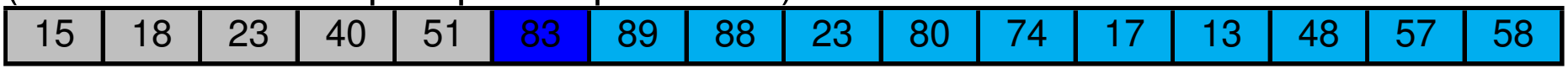
valeur : 23 | *limite* | *place*

40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	18	51	15	23	83	89	88	23	80	74	17	13	48	57	58
40	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	15	23	83	89	88	23	80	74	17	13	48	57	58
18	40	51	51	23	83	89	88	23	80	74	17	13	48	57	58
18	40	40	51	23	83	89	88	23	80	74	17	13	48	57	58
18	18	40	51	23	83	89	88	23	80	74	17	13	48	57	58
15	18	40	51	23	83	89	88	23	80	74	17	13	48	57	58
15	18	40	51	23	83	89	88	23	80	74	17	13	48	57	58
15	18	40	51	51	83	89	88	23	80	74	17	13	48	57	58
15	18	40	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58

...

Déroulement de l'algorithme (suite)

(en continuant un peu plus rapidement)



Déroulement de l'algorithme (suite)

(en continuant un peu plus rapidement)

15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58

Déroulement de l'algorithme (suite)

(en continuant un peu plus rapidement)

15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58

Déroulement de l'algorithme (suite)

(en continuant un peu plus rapidement)

15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	88	89	23	80	74	17	13	48	57	58

Déroulement de l'algorithme (suite)

(en continuant un peu plus rapidement)

15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	88	89	23	80	74	17	13	48	57	58
15	18	23	23	40	51	83	88	89	80	74	17	13	48	57	58

Déroulement de l'algorithme (suite)

(en continuant un peu plus rapidement)

15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	88	89	23	80	74	17	13	48	57	58
15	18	23	23	40	51	83	88	89	80	74	17	13	48	57	58
15	18	23	23	40	51	80	83	88	89	74	17	13	48	57	58

Déroulement de l'algorithme (suite)

(en continuant un peu plus rapidement)

15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	88	89	23	80	74	17	13	48	57	58
15	18	23	23	40	51	83	88	89	80	74	17	13	48	57	58
15	18	23	23	40	51	80	83	88	89	74	17	13	48	57	58
15	18	23	23	40	51	74	80	83	88	89	17	13	48	57	58

Déroulement de l'algorithme (suite)

(en continuant un peu plus rapidement)

15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	88	89	23	80	74	17	13	48	57	58
15	18	23	23	40	51	83	88	89	80	74	17	13	48	57	58
15	18	23	23	40	51	80	83	88	89	74	17	13	48	57	58
15	18	23	23	40	51	74	80	83	88	89	17	13	48	57	58
15	17	18	23	23	40	51	74	80	83	88	89	13	48	57	58

Déroulement de l'algorithme (suite)

(en continuant un peu plus rapidement)

15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	88	89	23	80	74	17	13	48	57	58
15	18	23	23	40	51	83	88	89	80	74	17	13	48	57	58
15	18	23	23	40	51	80	83	88	89	74	17	13	48	57	58
15	18	23	23	40	51	74	80	83	88	89	17	13	48	57	58
15	17	18	23	23	40	51	74	80	83	88	89	13	48	57	58
13	15	17	18	23	23	40	51	74	80	83	88	89	48	57	58

Déroulement de l'algorithme (suite)

(en continuant un peu plus rapidement)

15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	88	89	23	80	74	17	13	48	57	58
15	18	23	23	40	51	83	88	89	80	74	17	13	48	57	58
15	18	23	23	40	51	80	83	88	89	74	17	13	48	57	58
15	18	23	23	40	51	74	80	83	88	89	17	13	48	57	58
15	17	18	23	23	40	51	74	80	83	88	89	13	48	57	58
13	15	17	18	23	23	40	51	74	80	83	88	89	48	57	58
13	15	17	18	23	23	40	48	51	74	80	83	88	89	57	58

Déroulement de l'algorithme (suite)

(en continuant un peu plus rapidement)

15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	88	89	23	80	74	17	13	48	57	58
15	18	23	23	40	51	83	88	89	80	74	17	13	48	57	58
15	18	23	23	40	51	80	83	88	89	74	17	13	48	57	58
15	18	23	23	40	51	74	80	83	88	89	17	13	48	57	58
15	17	18	23	23	40	51	74	80	83	88	89	13	48	57	58
13	15	17	18	23	23	40	51	74	80	83	88	89	48	57	58
13	15	17	18	23	23	40	48	51	74	80	83	88	89	57	58
13	15	17	18	23	23	40	48	51	57	74	80	83	88	89	58

Déroulement de l'algorithme (suite)

(en continuant un peu plus rapidement)

15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	89	88	23	80	74	17	13	48	57	58
15	18	23	40	51	83	88	89	23	80	74	17	13	48	57	58
15	18	23	23	40	51	83	88	89	80	74	17	13	48	57	58
15	18	23	23	40	51	80	83	88	89	74	17	13	48	57	58
15	18	23	23	40	51	74	80	83	88	89	17	13	48	57	58
15	17	18	23	23	40	51	74	80	83	88	89	13	48	57	58
13	15	17	18	23	23	40	51	74	80	83	88	89	48	57	58
13	15	17	18	23	23	40	48	51	74	80	83	88	89	57	58
13	15	17	18	23	23	40	48	51	57	74	80	83	88	89	58
13	15	17	18	23	23	40	48	51	57	58	74	80	83	88	89

Tri par sélection-permutation

- On aimerait ne plus avoir à modifier la partie déjà triée (déplacement des éléments trop coûteux).
  - On distingue toujours les deux parties des éléments mais cette fois :

partie triée (élts $\leqslant$ élts non triés)	partie non triée (élts $\geqslant$ élts triés)

  - Ajout d'un élément à la fin de la partie triée
    $\Rightarrow$ il doit être $\leqslant$ aux éléments de la partie non triée
- Principe :
  - on cherche le plus petit élément dans la partie non triée (sélection) et on le place au début de cette partie (permutation).
- Nouvelle zone triée $=$ ancienne zone triée, plus cet élément.
- Nouvelle zone non triée $=$ ancienne zone non triée, moins cet élément.

Étapes de sélection-permutation

- La sélection-permutation d'un élément se fait en 2 étapes :
  - Trouver le minimum dans la partie non triée : x

(partie triée $\leqslant$) (partie non triée $\geqslant$)

Étapes de sélection-permutation

- La sélection-permutation d'un élément se fait en 2 étapes :
  - Trouver le minimum dans la partie non triée : x

(partie triée $\leqslant$) (partie non triée $\geqslant$)

  - Permuter avec le premier élément de la partie non triée

(partie triée $\leqslant$) (partie non triée $\geqslant$)

Étapes de sélection-permutation

- La sélection-permutation d'un élément se fait en 2 étapes :
  - Trouver le minimum dans la partie non triée : x

(partie triée $\leqslant$) (partie non triée $\geqslant$)

  - Permuter avec le premier élément de la partie non triée

(partie triée $\leqslant$) (partie non triée $\geqslant$)

(partie triée $\leqslant$) (partie non triée $\geqslant$)

Remarques

- On voit que la nouvelle partie grisée est toujours triée :

- Au départ, on a logiquement :

  partie triée vide

  partie non triée tout le tableau

- Cette fois, on peut gagner une étape à la fin (dernier élément).
- On termine quand la partie non triée ne contient plus qu'un seul élément : il s'incorpore directement dans la partie triée.

Algorithme

fonction triSélect(**in-out** *tab* : tableau d'entiers, **in** *taille* : entier) : **vide**

Trie le tableau *tab* de *taille* éléments en utilisant le tri par sélection-permutation.

Lexique local des variables

iMin	(entier)	indice du minimum de la partie non triée
min	(entier)	valeur du minimum de la partie non triée
limite	(entier)	indice du 1er élément de la partie non triée
i	(entier)	indice de parcours du tableau pour trouver le min

Algorithme de triSélect

```
pour limite de 0 à taille − 2 faire
    iMin ← limite
    pour i de limite + 1 à taille − 1 faire
        si tab[i] < tab[iMin] alors
            iMin ← i
        fsi
    fpour
    min ← tab[iMin]
    tab[iMin] ← tab[limite]
    tab[limite] ← min
fpour
```

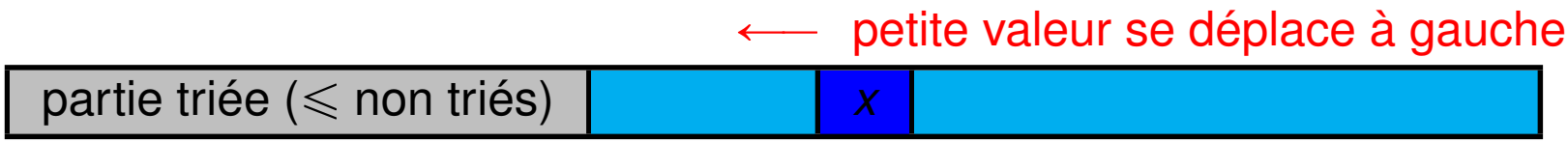
Déroulement de l'algorithme

(en exercice. . .)

Tri à bulles

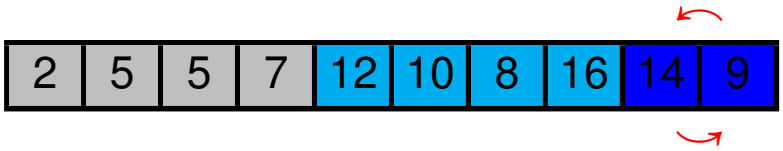
- On aimerait éviter le calcul des minimums (recherche coûteuses).
  - On distingue toujours la partie triée ($\leqslant$) et la partie non triée :

  - On fait descendre les éléments les plus petits par parcours successifs :



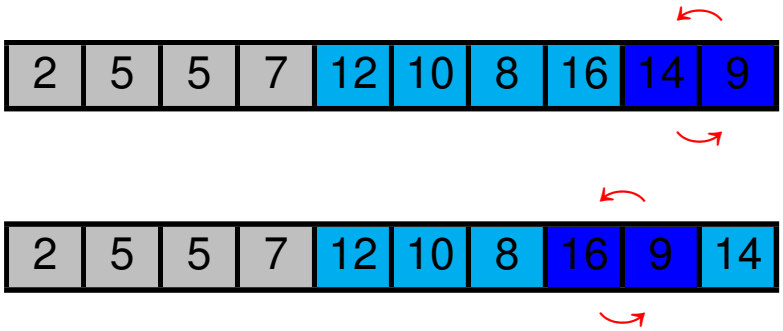
- Principe :
  - Parcours droite–gauche de la partie non triée avec comparaison deux à deux des éléments consécutifs et remise en ordre éventuelle.
- Nouvelle zone triée $=$ ancienne zone triée plus élément minimum de la zone non triée.
- Nouvelle zone non triée $=$ ancienne zone non triée, moins cet élément.

Déroulement d'un parcours



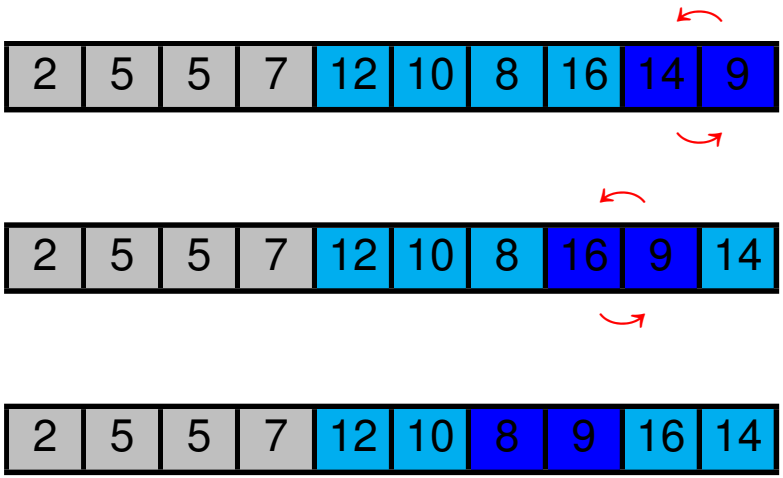
- On commence à la fin du tableau.
- On compare les éléments deux à deux.
- On les échange s'ils ne sont pas dans l'ordre.

Déroulement d'un parcours



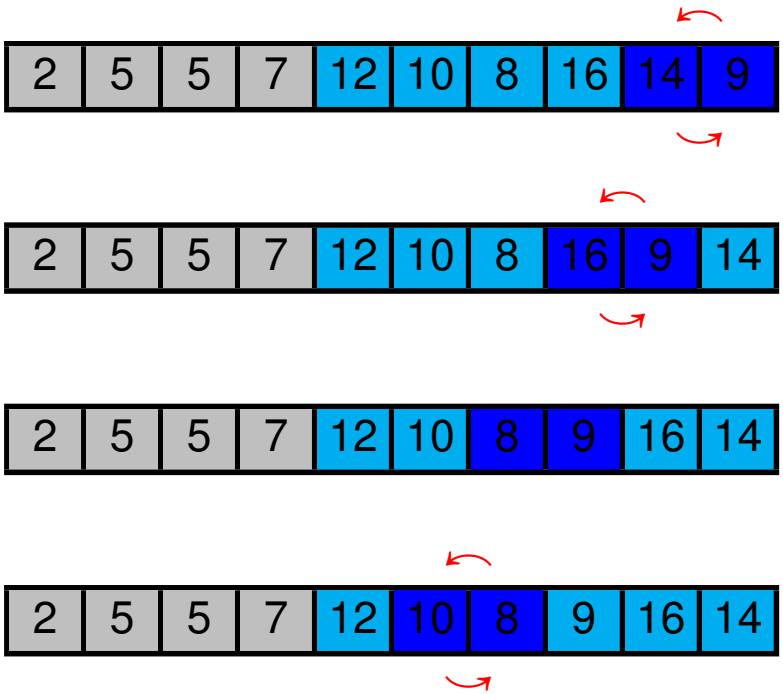
- On commence à la fin du tableau.
- On compare les éléments deux à deux.
- On les échange s'ils ne sont pas dans l'ordre.
- On recommence en se décalant d'une case à gauche, jusqu'au début de la partie non triée.

Déroulement d'un parcours



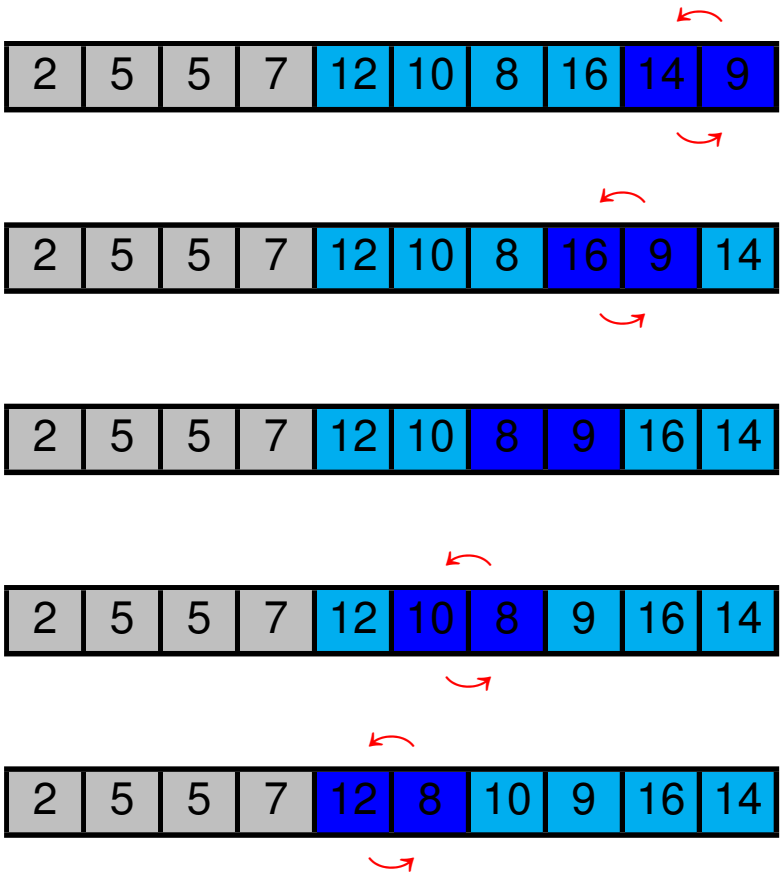
- On commence à la fin du tableau.
- On compare les éléments deux à deux.
- On les échange s'ils ne sont pas dans l'ordre.
- On recommence en se décalant d'une case à gauche, jusqu'au début de la partie non triée.

Déroulement d'un parcours



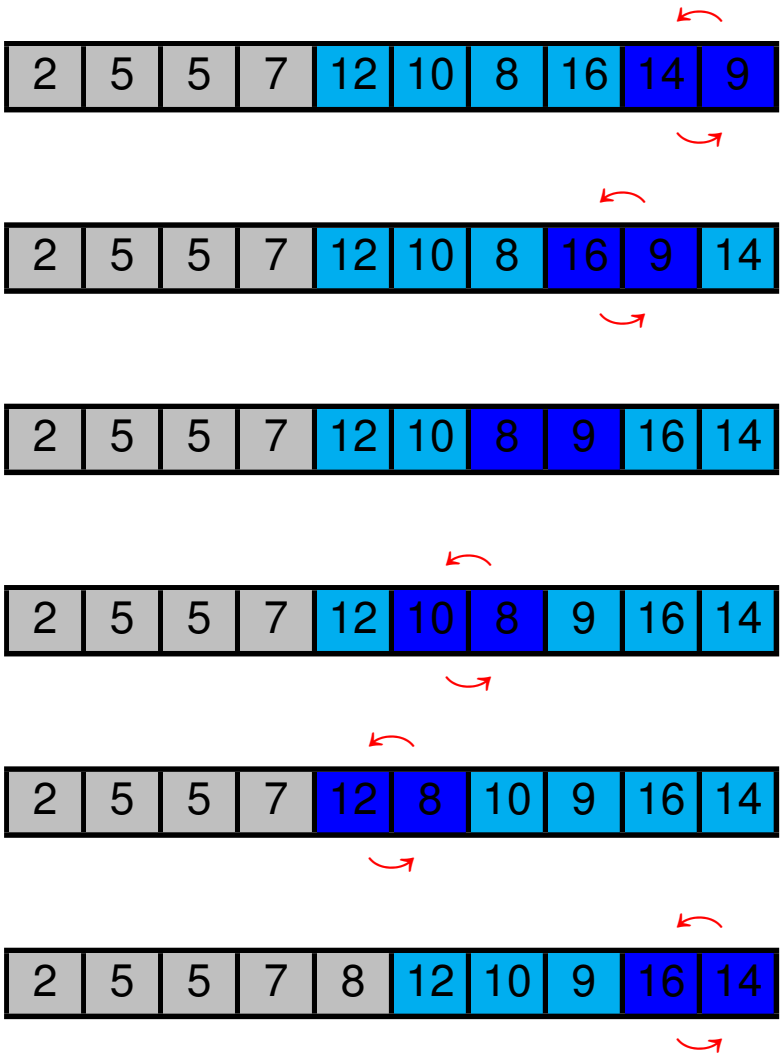
- On commence à la fin du tableau.
- On compare les éléments deux à deux.
- On les échange s'ils ne sont pas dans l'ordre.
- On recommence en se décalant d'une case à gauche, jusqu'au début de la partie non triée.

Déroulement d'un parcours



- On commence à la fin du tableau.
- On compare les éléments deux à deux.
- On les échange s'ils ne sont pas dans l'ordre.
- On recommence en se décalant d'une case à gauche, jusqu'au début de la partie non triée.

Déroulement d'un parcours



- On commence à la fin du tableau.
- On compare les éléments deux à deux.
- On les échange s'ils ne sont pas dans l'ordre.
- On recommence en se décalant d'une case à gauche, jusqu'au début de la partie non triée.
- À la fin du parcours, le min. de la partie non triée est au début de celle-ci.
- On recommence avec la nouvelle zone non triée.
- Arrêt quand la zone non triée contient un seul élément, ou est vide.
- NB : le min. n'est pas la seule valeur déplacée, 9 a aussi été déplacé vers la gauche.

Algorithme

fonction triBulles(**in-out** *tab* : tableau d'entiers, **in** *taille* : entier) : **vide**

Trie le tableau *tab* de *taille* éléments en utilisant le tri à bulles.

Lexique local des variables

tmp	(entier)	variable pour l'échange des valeurs
limite	(entier)	indice du 1er élément de la partie non triée
i	(entier)	indice de parcours du tableau pour trouver le min

Algorithme de triBulles

pour *limite* **de** 0 **à** $taille - 2$ **faire**
  pour *i* **de** $taille - 1$ **à** $limite + 1$ **en descendant faire**
    si $tab[i] < tab[i-1]$ **alors**
      $tmp \leftarrow tab[i]$
      $tab[i] \leftarrow tab[i-1]$
      $tab[i-1] \leftarrow tmp$
    fsi
  fpour
fpour

Remarques

- C'est une variante du tri par sélection-permutation où la recherche du min. est remplacée par les déplacements des petites valeurs.
- Il peut avoir un coût supérieur à cause des échanges (ordre inverse).
- De plus, si au k^{e} parcours, la fin tu tableau est déjà triée, on effectue quand même les $\mathit{taille} - 1 - k$ parcours restant.
  $\Rightarrow$ Pas efficace !
- On souhaite s'arrêter dès qu'il n'y a plus d'échange de valeurs.
- On commence un nouveau parcours seulement s'il y a eu au moins un échange pendant le parcours précédent :
  - utilisation d'un booléen pour la détection de l'échange ;
  - la boucle principale devient conditionnelle.

Algorithme (v2)

Algorithme :

fonction triBulles(**in-out** *tab* : tableau d'entiers, **in** *taille* : entier) : **vide**

Trie le tableau *tab* de *taille* éléments en utilisant le tri à bulles.

Lexique local des variables

tmp	(entier)	variable pour l'échange des valeurs
limite	(entier)	indice du 1er élément de la partie non triée
i	(entier)	indice de parcours du tableau pour trouver le min
échange	(booléen)	vrai si au moins un échange lors d'un parcours

Algorithme (v2), suite

Algorithme :

Algorithme

limite $\leftarrow 0$
échange $\leftarrow$ vrai
tant que *échange* $\wedge$ *limite* $\leqslant$ *taille* -2 **faire**
  échange $\leftarrow$ faux
  pour *i* **de** *taille* -1 **à** *limite* $+1$ **en descendant faire**
    si $tab[i] < tab[i-1]$ **alors**
      $tmp \leftarrow tab[i]$
      $tab[i] \leftarrow tab[i-1]$
      $tab[i-1] \leftarrow tmp$
      échange $\leftarrow$ vrai
    fsi
  fpour
  limite $\leftarrow$ *limite* $+1$
ftant

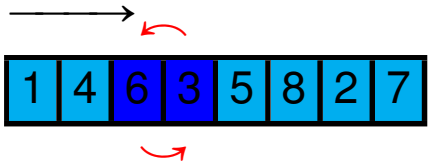
Double tri à bulles

- On aimerait ne plus traiter un seul élément à la fois (recherches des min. trop coûteuses).
  - On distingue cette fois :

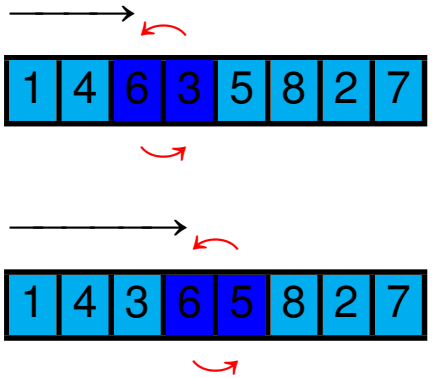
  - On fait descendre/monter les éléments les plus petits/grands par parcours successifs de la partie non triée :

- Principe :
  - Parcours de la partie non triée avec comparaison deux à deux des éléments consécutifs et remise en ordre éventuelle.
- Nouvelles zones triées = anciennes zones triées plus éléments min./max. de la zone non triée.
- Nouvelle zone non triée = ancienne zone non triée, moins ces éléments.

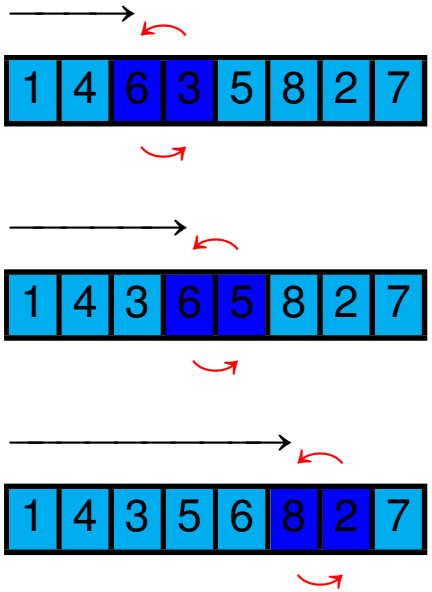
Exemple d'exécution



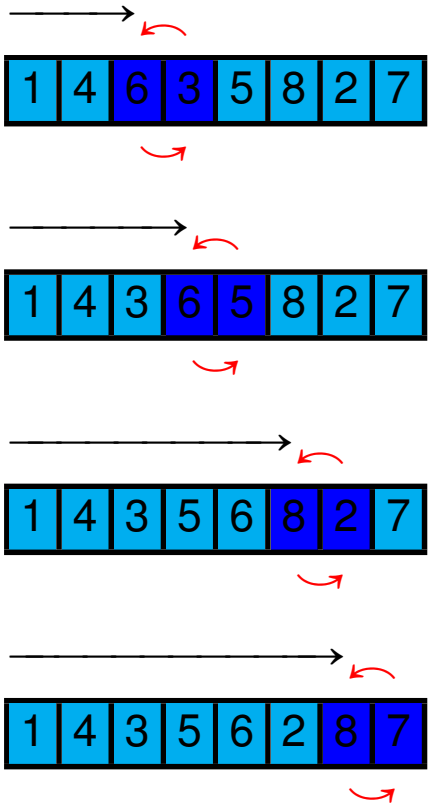
Exemple d'exécution



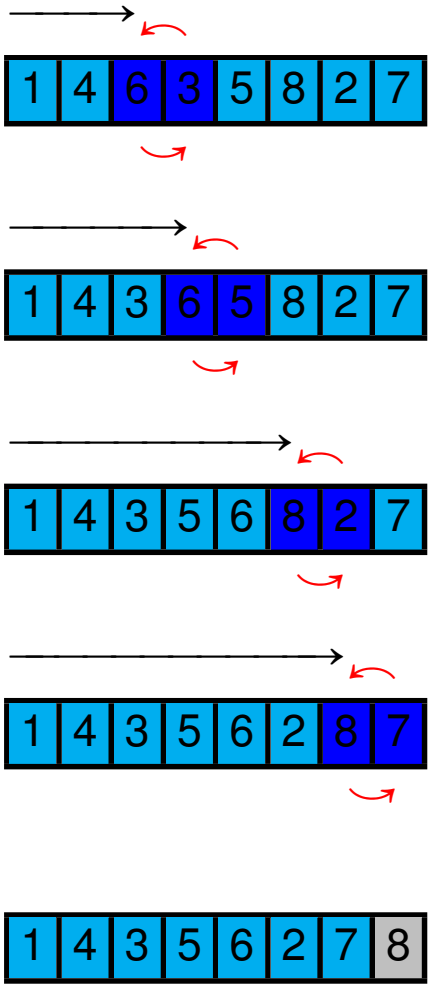
Exemple d'exécution



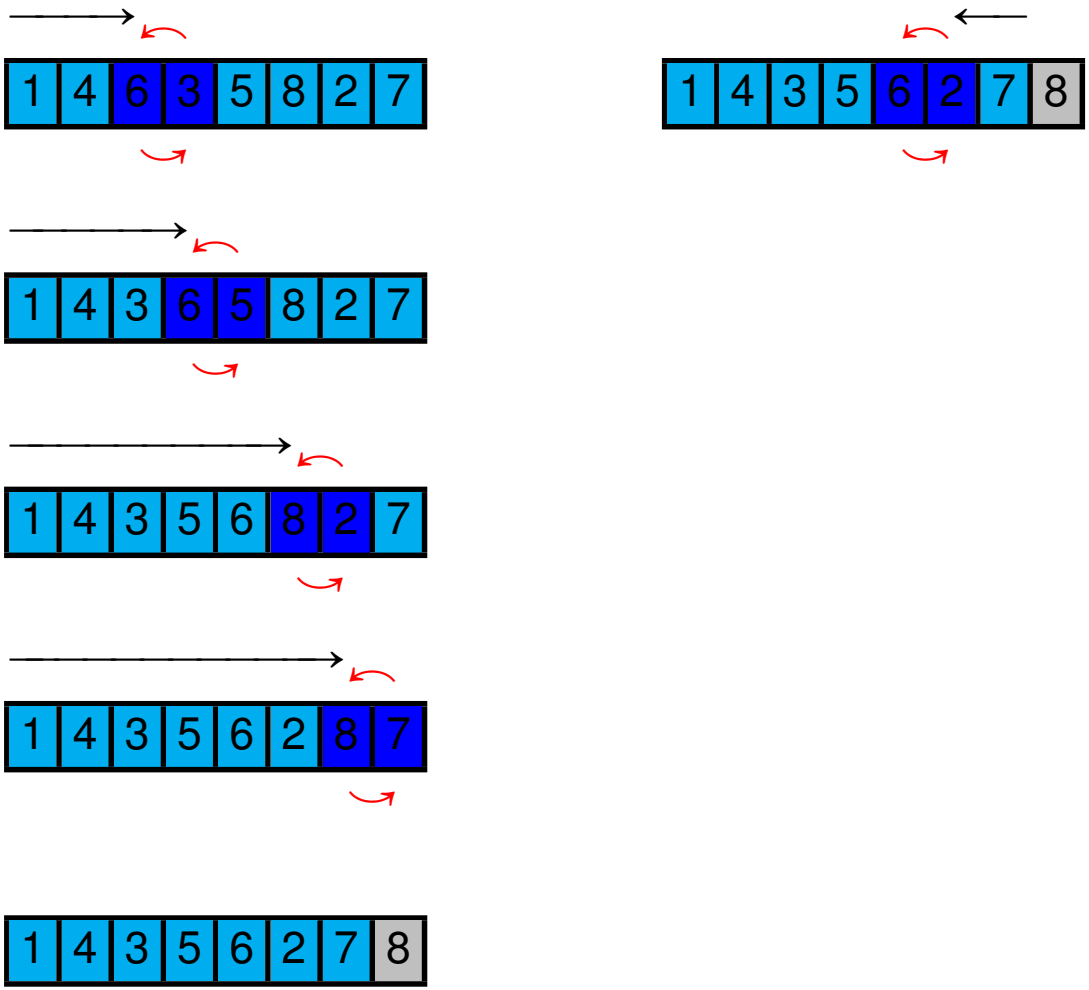
Exemple d'exécution



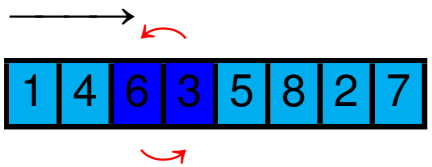
Exemple d'exécution

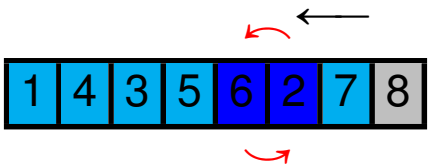


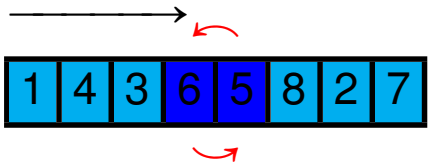
Exemple d'exécution

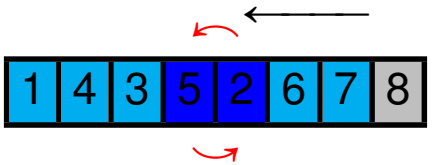


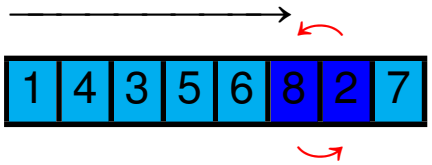
Exemple d'exécution

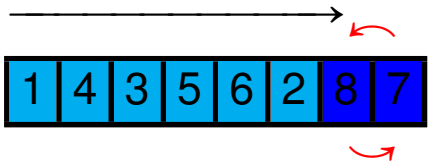




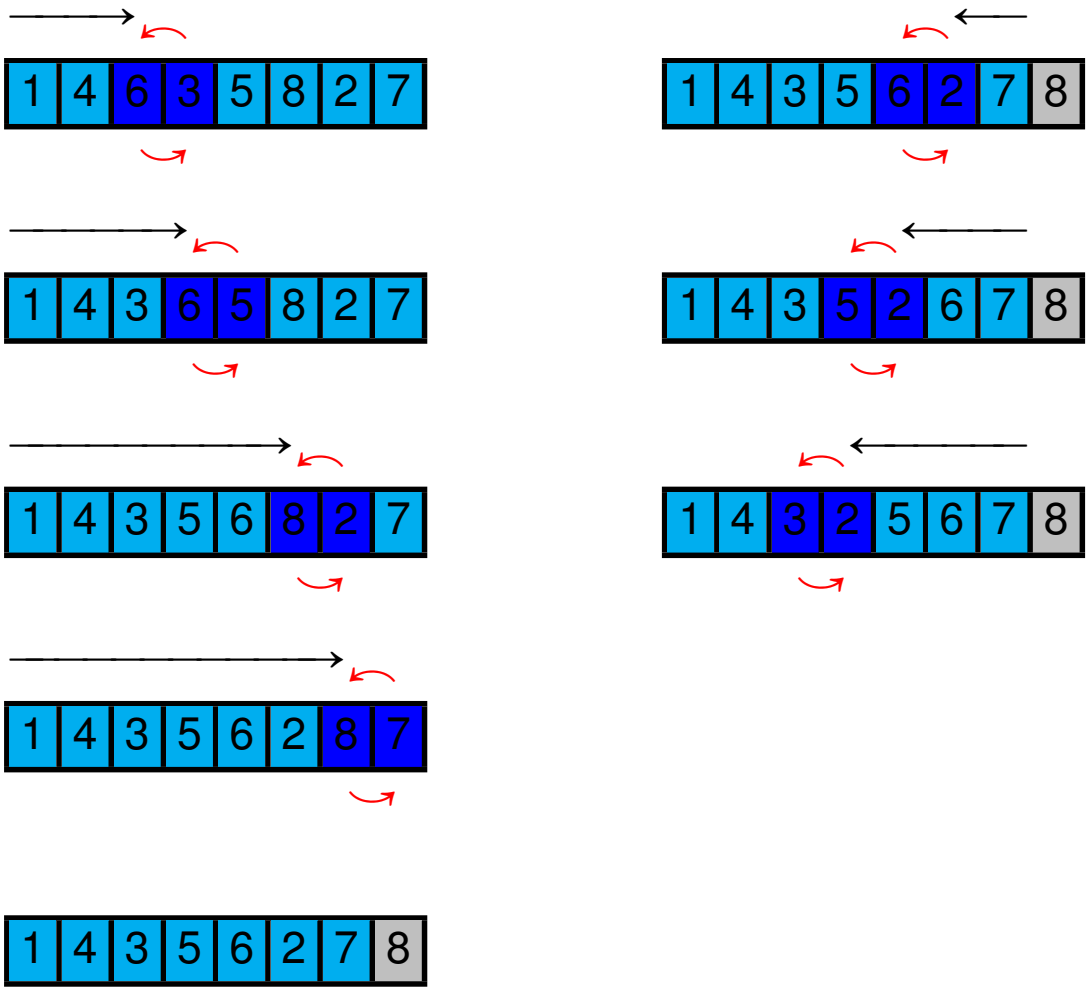




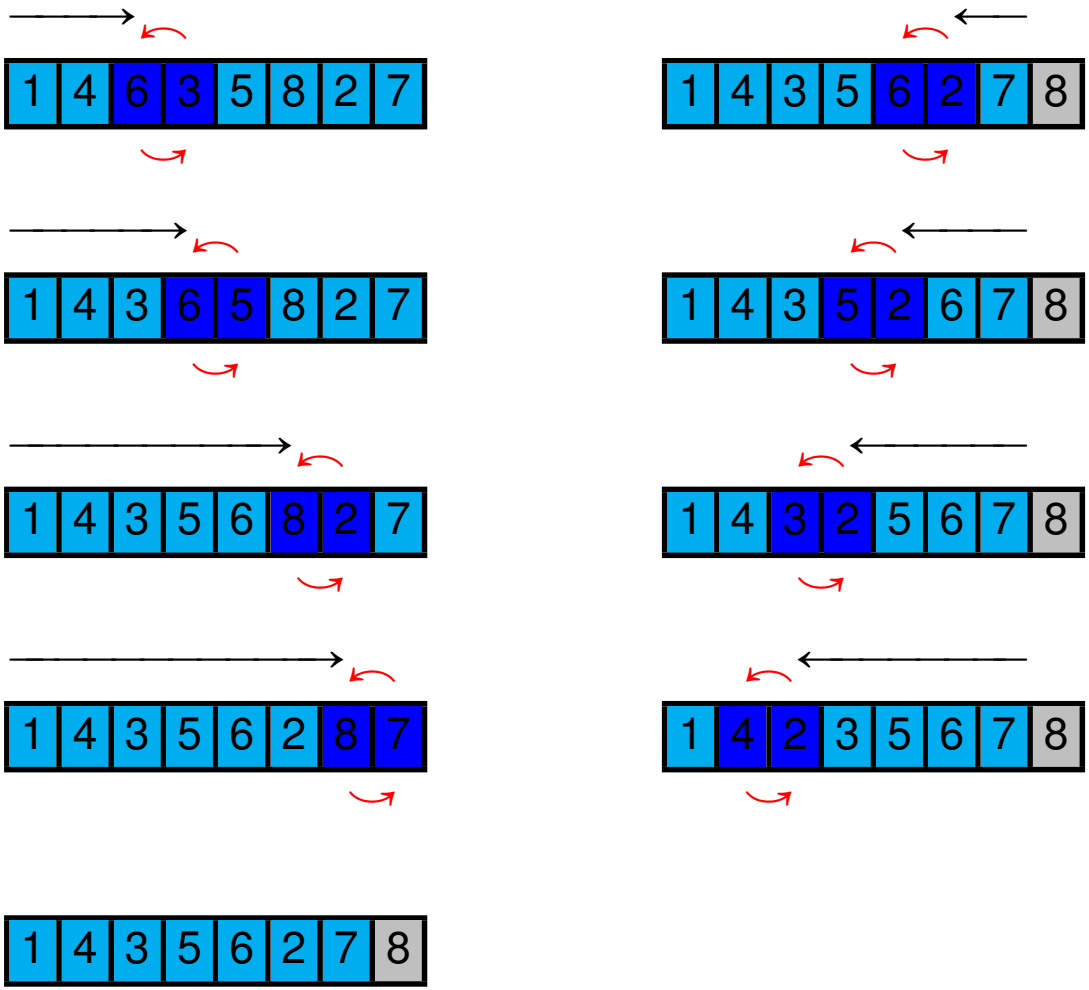




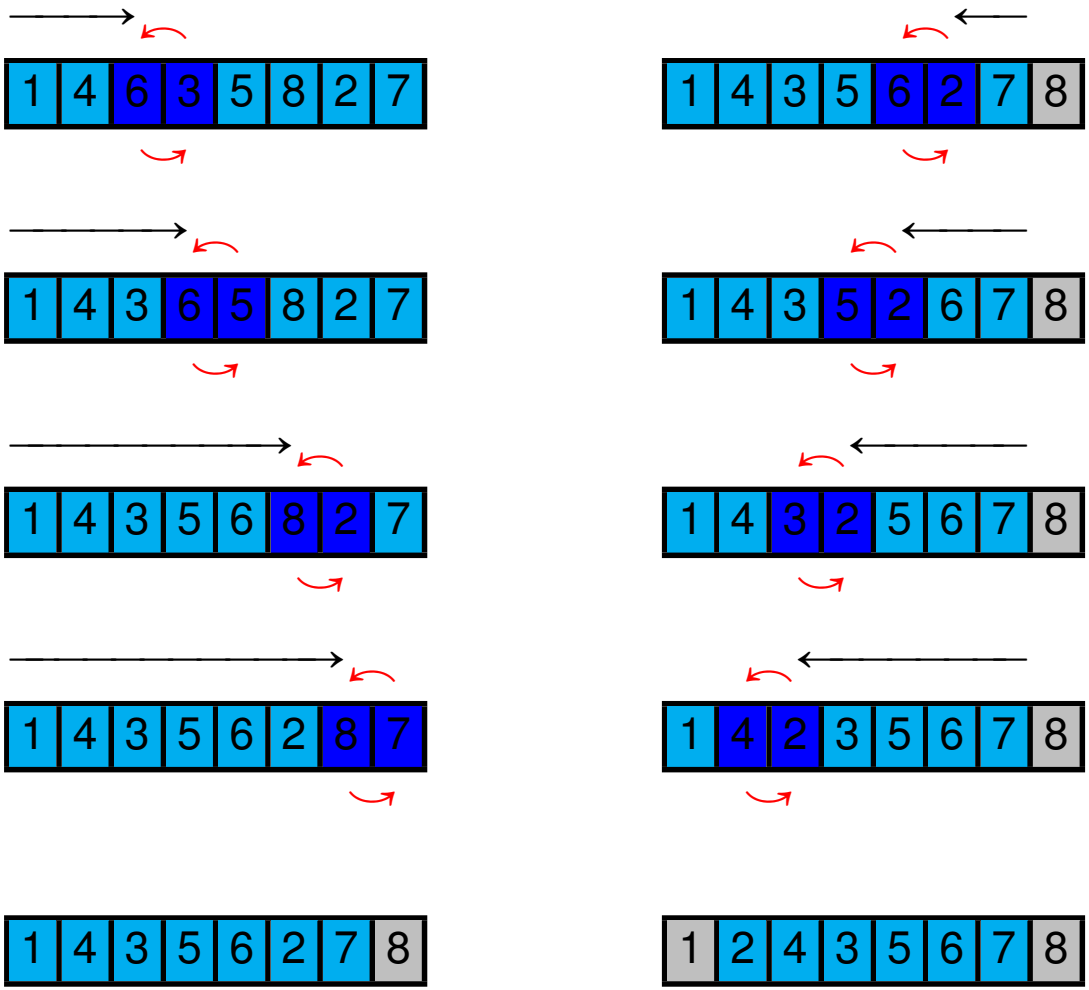
Exemple d'exécution



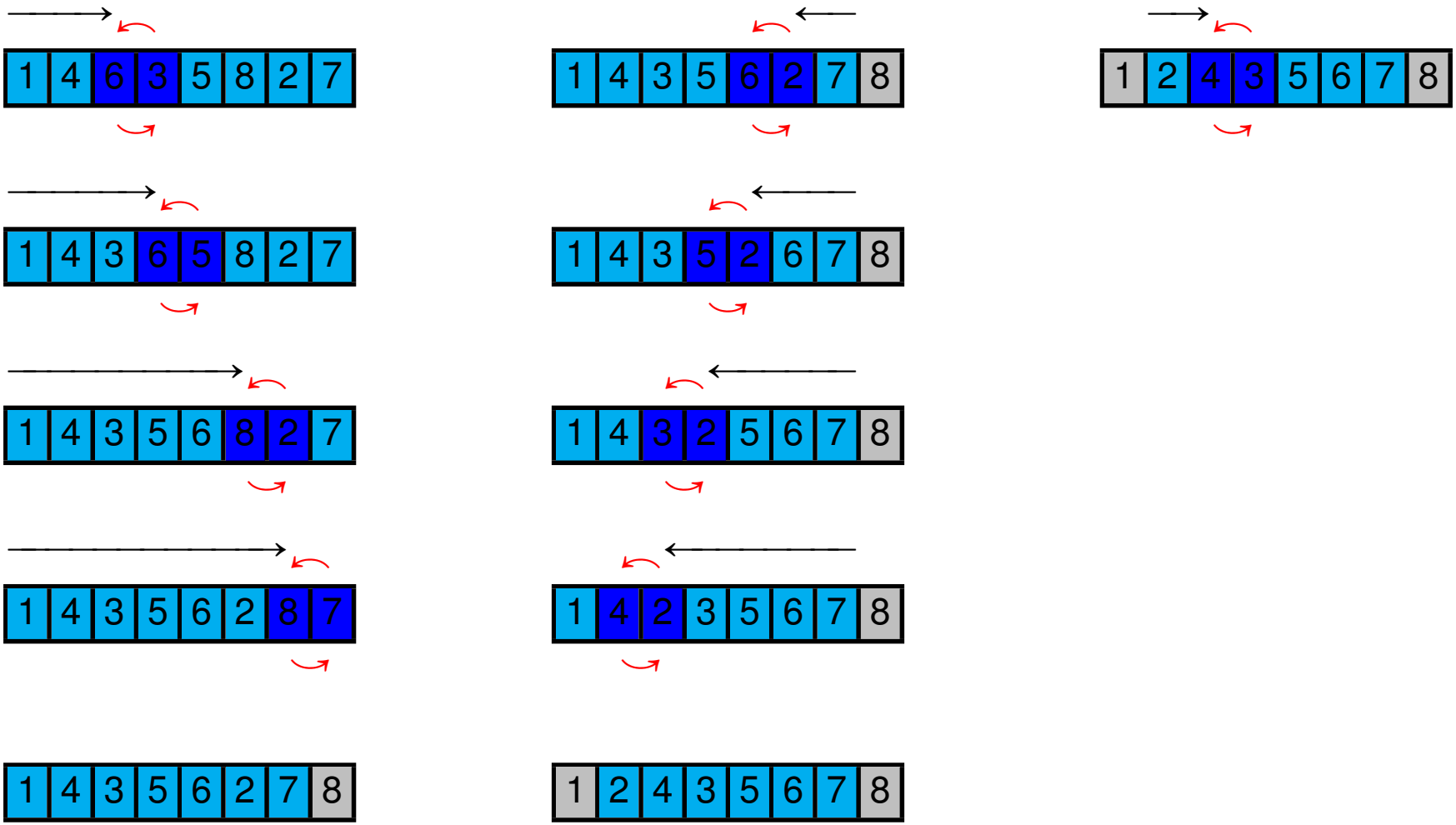
Exemple d'exécution



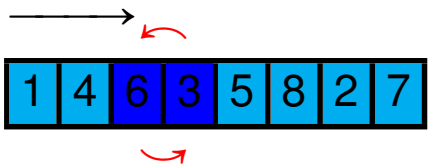
Exemple d'exécution

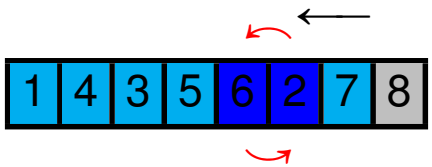


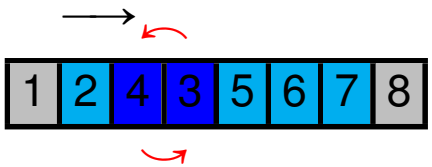
Exemple d'exécution

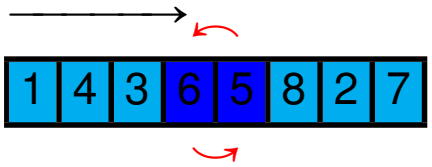


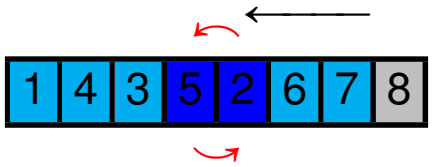
Exemple d'exécution

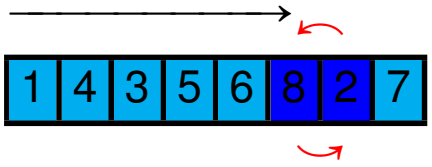


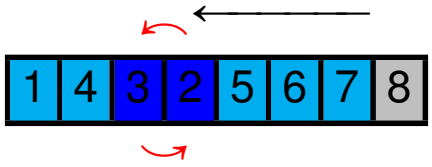


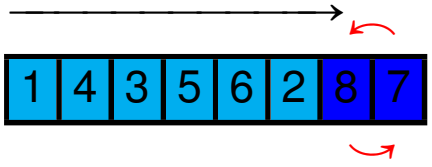


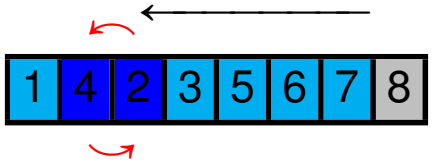




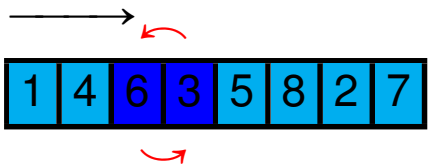


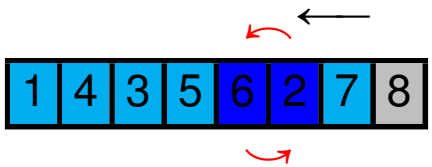


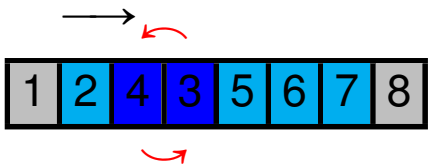


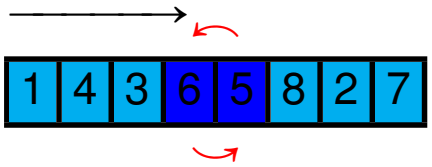


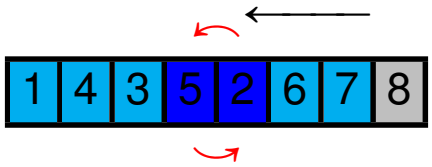
Exemple d'exécution

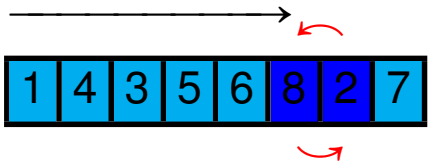


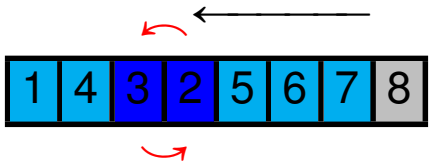


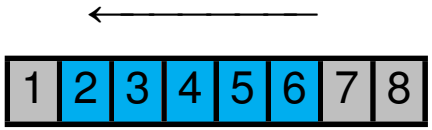


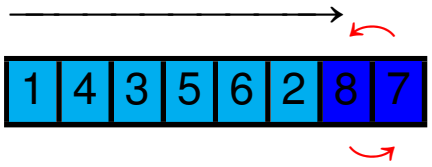


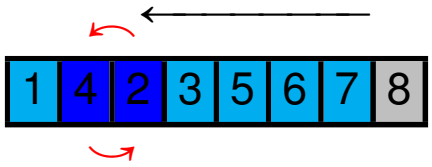




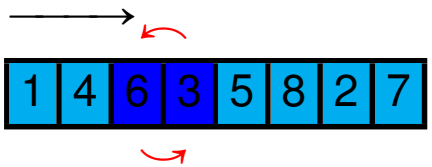


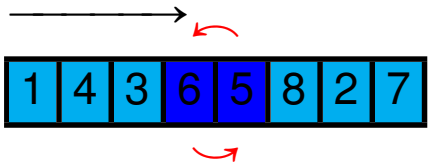


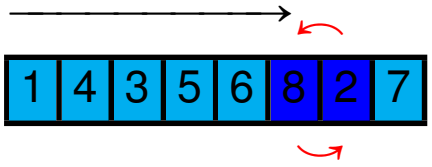


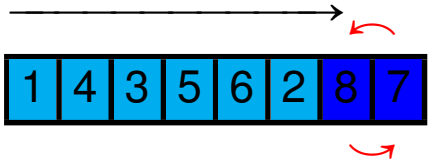


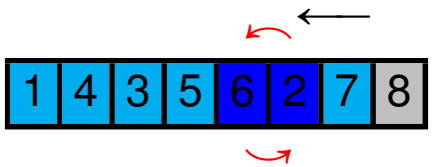
Exemple d'exécution

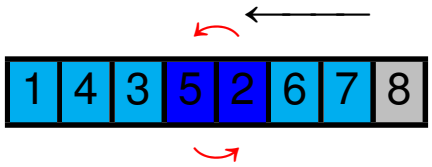


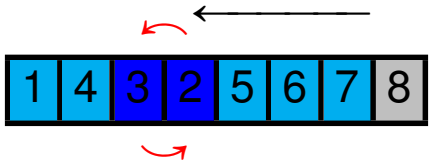


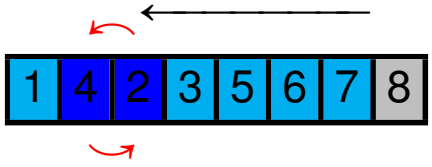


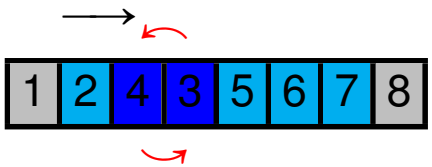


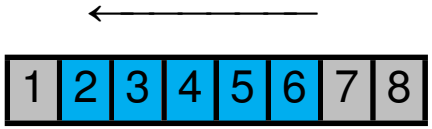








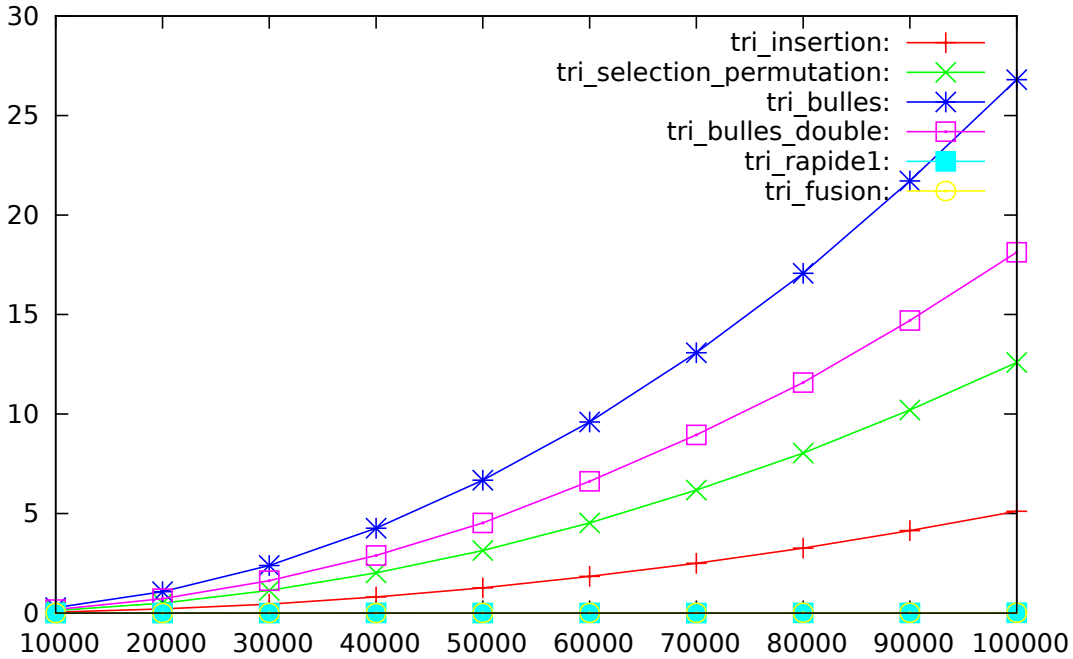




Arrêt sur parcours
sans aucune permutation

Conclusion

- La formalisation du double tri à bulles est laissée en exercice.
- Démonstration (illustration) des algorithmes de tri. . .
- Il existe de nombreux autres algorithmes de tri.



Plan

Les énumérations

- Une énumération permet de modéliser des ensembles finis de valeurs abstraites.
- Les valeurs de l'ensemble ont généralement un lien sémantique entre elles.

Par exemple, une variable *couleur* peut être définie par :

Lexique des variables

couleur (**énum** {rouge, vert, bleu, jaune}) une couleur ...

- Cette définition spécifie une liste particulière de valeurs possibles pour la variable *couleur*.
- La valeur de *couleur* ne pourra donc être que rouge, vert, bleu ou jaune, et rien d'autre.

Propriétés

- Les noms des éléments d'une énumération ne sont pas des chaînes de caractères, mais des noms internes à l'algorithme.
  $\Rightarrow$ la lecture et l'affichage des énumérations ne sont pas possibles directement.
- Il faut donc faire la conversion manuellement.

Exemple :

Lexique des variables

jour	(**énum** {lun, mar, mer, jeu, ven, sam, dim})	un jour de la semaine	...

- On ne peut lire directement les valeurs lun, mar, ... et il faut donc passer par une variable intermédiaire ayant un type supportant les lectures/affichages.

Exemple de lecture

Lexique des variables

val	(entier)	valeur lue	INTERMÉDIAIRE
jour	(**énum** {lun, mar, mer, jeu, ven, sam, dim})	jour de la semaine	INTERMÉDIAIRE

Algorithme

écrire "`Entrez le numéro d'un jour de la semaine entre 1 et 7 :`"
val ← lire
selon que *val* **est**
  cas *1* **:** *jour* ← lun
  cas *2* **:** *jour* ← mar
  cas *3* **:** *jour* ← mer
  cas *4* **:** *jour* ← jeu
  cas *5* **:** *jour* ← ven
  cas *6* **:** *jour* ← sam
  cas *7* **:** *jour* ← dim
  défaut : écrire "`valeur invalide:`", *val*
fselon

Exemple d'affichage

Lexique des variables

jour	(**énum** {lun, mar, mer, jeu, ven, sam, dim})	jour de la semaine	INTERMÉDIAIRE

Algorithme

```
...
selon que jour est
    cas lun : écrire "lundi"
    cas mar : écrire "mardi"
    cas mer : écrire "mercredi"
    cas jeu : écrire "jeudi"
    cas ven : écrire "vendredi"
    cas sam : écrire "samedi"
    cas dim : écrire "dimanche"
    // le cas défaut est inutile ici :
    // toutes les valeurs possibles sont traitées.
fselon
```

En Java

- En Java les énumération sont déclarées grâce au mot clef **enum** :

```
enum <Nom> { CONSTANTE1, CONSTANTE2, ... }
```

  - le nom permet de faire référence ultérieurement à l'énumération ;
  - en général, les valeurs de l'énumération sont notées `Nom.CONSTANTE1` ;
  - sauf dans un `switch` où le nom de l'énumeration est omis.

- Exemple :

```
enum Jour { LUN, MAR, MER, JEU, VEN, SAM, DIM }; // l'énumération
Jour jour1, jour2;          // déclaration de variables
jour1 = Jour.LUN;           // affectation d'une valeur
if (jour1 == Jour.MAR)
    jour2 = Jour.VEN;       // affectation d'une autre valeur
switch (jour2) {            // exemple avec un switch
case LUN:
    ...
    break;
case MAR:
    ...
}
```

- Cf. correction du TP Labyrinthe pour un exemple concret.
- Cf. manuel de Java pour tous les détails.

Plan

Le type structure

- Les tableaux permettent de réunir plusieurs données de même type dans une seule entité.
- La structure permet de réunir des données de types quelconques dans une même entité logique.
- Les éléments de la structure (appelés champs) ont normalement un lien logique entre eux :
  - ils représentent les différentes caractéristiques qui définissent l'entité.
- Exemple : représentation d'un étudiant

Numéro d'étudiant	(entier)
Nom	(chaîne de caractères)
Prénom	(chaîne de caractères)
Âge	(entier)

Déclaration d'une variable structure

Exemple : écriture de la structure étudiant

structure

num	*: (entier)*
nom	*: (chaîne)*
prénom	*: (chaîne)*
âge	*: (entier)*

- Ainsi, la déclaration d'une variable *étudiant* peut se faire par :

Lexique des variables

étudiant (**structure**

num	: (entier)
nom	: (chaîne)
prénom	: (chaîne)
âge	: (entier)

) les données d'un étudiant . . .

Définition d'un type structure

- Pour alléger l'écriture des définitions de variables, on peut aussi définir un nouveau type dans le lexique des types qui se place avant les autres lexiques.
- La déclaration d'un nouveau type se fait simplement par :

  NomDuType *définition du type*
- Par convention, les noms de types commencent par une majuscule.
- Ainsi, la déclaration du type *Étudiant* se fait par :

Lexique des types

Étudiant	**structure**		représentation d'un étudiant
	num	: (entier)	
	nom	: (chaîne)	
	prénom	: (chaîne)	
	âge	: (entier)	

Type structure et accès

- La déclaration de la variable *étudiant* peut alors se faire par :

Lexique des variables

étudiant (Étudiant) les données d'un étudiant ...

Type structure et accès

- La déclaration de la variable *étudiant* peut alors se faire par :

Lexique des variables

étudiant (Étudiant) les données d'un étudiant ...

- Accès à un champ :

  nomDeVariable.nomDuChamp

  Exemple :

  étudiant.nom ← lire

- Affectation de structure : si *étudiant1* et *étudiant2* sont deux variables de type Étudiant, l'instruction :

  étudiant1 ← *étudiant2*

  recopie les valeurs de tous les champs de la variable *étudiant2* dans *étudiant1* : même effet que pour les types simples (entier, réel, ...).

Structures et fonctions

- Une structure peut être passée en paramètre d'une fonction et aussi être renvoyée en résultat.

Exemples

Lexique des types

Étudiant	...	cf. exemple précédent
Date	**structure**	représentation d'une date
	jour : (entier)	
	mois : (entier)	
	année : (entier)	

Lexique des fonctions

fonction lireÉtu(**out** *étu* : Étudiant) : **ret** booléen

Lit les données d'un étudiant dans la structure *étu* et retourne vrai si la lecture a pu se faire correctement.

fonction demain(**in** *aujourdhui* : Date) : **ret** Date

Retourne une structure Date contenant la date du lendemain de *aujourdhui*.

Exemple complet

Types, constantes et fonctions...

Lexique des types

Date	**structure**		représentation d'une date
	jour	: (entier)	
	mois	: (entier)	
	année	: (entier)	
Étudiant	**structure**		représentation d'un étudiant
	num	: (entier)	
	nom	: (chaîne)	
	prénom	: (chaîne)	
	naissance	: (Date)	

Lexique des constantes

NB_ÉTU	(entier)	= 120	nombre d'étudiants dans la promo

Lexique des fonctions

fonction lireÉtu(**out** *étu* : Étudiant) : **ret** booléen

Lit les données d'un étudiant et retourne vrai si la lecture a réussi.

Exemple (suite)

Variables et algorithme principal...

Lexique des variables

tabÉtu	(tableau $[NB\_ÉTU]$ d'Étudiant)	tableau des étudiants	INTERMÉDIAIRE
i	(entier)	indice de parcours du tableau	INTERMÉDIAIRE
unÉtu	(Étudiant)	étudiant lu	INTERMÉDIAIRE

Algorithme

$i \leftarrow 0$
tant que $i < NB\_ÉTU \wedge$ lireÉtu$(unÉtu)$ **faire**
  $tabÉtu[i] \leftarrow unÉtu$
  si *tabÉtu[i].naissance.mois = 6* **alors**
    écrire "L'étudiant numéro ", *unÉtu.num*, " est né en juin."
  fsi
  $i \leftarrow i + 1$
ftant

Exemple (fin)

Définition de la fonction lireÉtu

fonction lireÉtu(**out** *étu* : Étudiant) : **ret** booléen

Lit les données d'un étudiant et retourne vrai si la lecture a réussi.

Algorithme de lireÉtu

étu.num ← lire
si *étu.num* = EOF **alors**
  retourner faux
fsi
étu.nom ← lire
si *étu.nom* = EOF **alors**
  retourner faux
fsi
...
`// Lecture et vérification des autres champs...`
...
retourner vrai

Structures en Java

Déclaration

- En Java, on parle de *classes*.
- Forme générale :

```
class <Nom> {
    <type> <champ>;
    <type> <champ>;
    ...
}
```

- Comme pour les tableaux, les objets d'une classe donnée doivent être construits explicitement avec le mot clef `new`.
- Exemple :

```
class Etudiant {
    int num;
    String nom;
    String prenom;
    int age;
}
Etudiant etudiant = new Etudiant();      // déclaration et construction d'une variable
```

Structures en Java (suite)

Utilisation

- Les champs peuvent être accédés par : `<variable>.<champ>`
- Les structures peuvent être utilisées comme type pour les paramètres ou le retour d'une fonction.
- Exemple :

```
Date demain(Date aujourdhui)
{
    Date dem = new Date();
    dem.jour = aujourdhui.jour;
    dem.mois = aujourdhui.mois;
    dem.annee = aujourdhui.annee;
    if (++dem.jour == 32) {
        dem.jour = 1;
        if (++dem.mois == 13) {
            dem.mois = 1;
            ++dem.annee;
        }
    }
    return dem;
}
```

Structures en Java (conclusion)

En réalité. . .

- Dans le langage Java, il est plus naturel de définir et manipuler des *objets*.
- En programmation orientée objet, les entités manipulées (*objets*) contiennent des données (*attributs*, similaires aux champs d'une structures), mais ont également des comportements associés (*méthodes*, des sortes de fonctions).
- Le concept de programmation orientée objets sera développé en S2.

Annexe

Références


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


David FLANAGAN
Java in a Nutshell
O'Reilly, 5e édition, 2005


Vincent GRANET
Algorithmique et programmation en Java
Dunod, 3e édition, 2010


ORACLE
Java Platform Standard Edition Documentation
http://docs.oracle.com/javase/

Consignes pour les examens

Documents

- Aucun document autorisé.
- Ordinateurs, calculatrices, téléphones, etc. interdits.

Mise en page

- Laisser une marge suffisante sur le bord gauche de chaque page.
- Numéroter les copies (et non les pages !) s'il y en a plusieurs.

Conseils généraux

- Lire intégralement le sujet avant de commencer.
- Répondre aux questions de manière précise et concise.

Consignes pour les examens (suite)

Rédaction des algorithmes

- Sauf instruction contraire, les algorithmes devront être décrits en utilisant le langage algorithmique présenté en cours.
- Le langage Java ne devra être utilisé que si c'est explicitement demandé.
- Un algorithme sans description de ses données et résultats n'a aucun sens.
- De même, la section « Idée de l'algorithme » est importante.
- La description et le rôle (donnée/résultat/intermédiaire) des variables est également important.