Algorithmique 1

Les expressions et les instructions de base

Tarek Boutefara

2025-10-31

Table of Contents

Chapitre 1. Introduction

Après avoir défini les objets que nous allons manipuler (les variables et constantes) ainsi que leur nature (leurs types de données), il est temps de passer à l'action. Le corps de l'algorithme, délimité par les mots-clés Début et Fin, est composé d'instructions qui opèrent sur ces données. Dans cette section, nous allons étudier les briques fondamentales de l'action algorithmique. Nous verrons d'abord comment combiner nos objets pour effectuer des calculs à travers les expressions. Ensuite, nous aborderons les trois instructions de base indispensables à tout algorithme : l'affectation (pour stocker une valeur), la lecture (pour recevoir une donnée) et l'écriture (pour afficher un résultat).

Chapitre 2. Les expressions

Une **expression** est une combinaison syntaxiquement valide d'**opérandes** et d'**opérateurs**, qui, une fois évaluée par l'algorithme, produit une **unique valeur résultat**. Les opérandes peuvent être des variables (ex: `Prix`), des constantes (ex: `TVA`), ou des valeurs littérales (ex: `100` ou `"Bonjour"`). Les opérateurs sont les symboles d'action que nous avons vus précédemment (ex: `+`, `*`, `>`, `ET`). Par exemple, `(Note1 + Note2) / 2` est une expression arithmétique, et `(Age ≥ 18)` est une expression logique.

Une expression possède elle-même un type de données, qui correspond au type de la valeur qu'elle retourne. Ce type résultant dépend directement du **type de ses opérandes** et de la **nature des opérateurs** utilisés. Par exemple, une opération arithmétique (`+`, `*`) entre deux `Entiers` produira un `Entier` ; si l'un des opérandes est un `Réel`, le résultat sera promu en `Réel`. De la même manière, tout opérateur de comparaison (`<`, `≠`, `≥`) produit toujours un résultat de type `Booléen`, et les opérateurs logiques (`ET`, `OU`, `NON`) s'appliquent à des opérandes `Booléens` pour retourner un `Booléen`.

Lorsqu'une expression complexe implique plusieurs opérateurs différents (par exemple `A + B * C`), l'algorithme doit savoir dans quel ordre effectuer les calculs. Cet ordre est défini par des **règles de priorité** (ou préséance) qui sont similaires à celles des mathématiques. Les opérateurs arithmétiques (`*`, `/`, `DIV`, `MOD`) sont prioritaires sur l'addition et la soustraction (`+`, `-`). De même, les opérateurs logiques ont leur propre hiérarchie (le `NON` est prioritaire sur le `ET`, lui-même prioritaire sur le `OU`). Pour forcer un ordre d'évaluation différent ou simplement pour améliorer la lisibilité et éviter toute ambiguïté, il est indispensable d'utiliser des **parenthèses** `()`. Les parties entre parenthèses sont toujours évaluées en premier.

Ainsi, nous pouvons définir l'ordre des opérateurs comme suit :

- `( )`
- `-` (unaire)
- `*`, `/`, `DIV`, `MOD`
- `+`, `-`
- Opérateurs de comparaison
- `NON`
- `ET`
- `OU`

Notes imprtantes :

- Pour les opérateurs de même priorité, l'expression est évaluée de gauche à droite.
- Une expression est évaluée un opérateur à la fois.

Exemple :

- 12 * 3 + 7 * 2 - 10 =
- 36 + 7 * 2 - 10 =

- 36 + 14 - 10 =
- 50 - 10 =
- 40

Ecriture des symboles mathématiques :

Il est essentiel de comprendre qu'un algorithme doit s'écrire de manière **linéaire** (ligne par ligne) et non ambiguë, contrairement à la notation mathématique standard. Les symboles mathématiques que vous utilisez sur papier ne sont pas directement transposables en pseudo-code.

- **La Division :** On n'utilise jamais la barre de fraction (par exemple, $\frac{A+B}{C}$). Toute division doit être "aplatie" en utilisant l'opérateur de division (`/`). L'exemple précédent s'écrirait donc : `(A+B)/C`. L'usage des parenthèses est ici vital pour préserver l'ordre des opérations.
- **Puissance et Racine Carrée :** Il n'existe pas d'opérateur standard pour l'élévation à la puissance (comme x^y) ou pour la racine carrée (comme $\sqrt{x}$) en pseudo-code de base. Ces opérations sont considérées comme des **fonctions** prédéfinies que l'on "appelle".
  - **Puissance :** `resultat ← pow(X, Y)` (pour x^y)
  - **Racine Carrée :** `resultat ← sqrt(X)` (pour $\sqrt{x}$)

Chapitre 3. Les instrcutions de base

Les instructions de base sont les actions les plus élémentaires que l'algorithme peut exécuter. Nous en distinguons trois principales : l'affectation, la lecture et l'écriture.

3.1. L'Instruction d'Affectation

C'est l'opération la plus fondamentale en algorithmique. Son role est d'attribuer une valeur à une variable. C'est l'instruction qui permet de stocker une donnée en mémoire et de la mettre à jour.

L'affectation se note avec une flèche pointant vers la gauche (←).

- Nom_Variable ← Expression

L'instruction d'affectation se déroule toujours en deux temps :

- Évaluation à droite : L'algorithme évalue d'abord tout ce qui se trouve à droite de la flèche. Il calcule l'expression (qui peut être une simple valeur, une autre variable, ou un calcul complexe) pour obtenir une unique valeur résultat.
- Rangement à gauche : Une fois cette valeur obtenue, l'algorithme la stocke dans la "case mémoire" (la variable) dont le nom est indiqué à gauche de la flèche.

L'affectation est une opération destructive. Si la variable à gauche contenait déjà une valeur, cette ancienne valeur est définitivement perdue et remplacée par la nouvelle.

Exemples :

- `Prix_TTC ← 20.5`
- `MonAge ← Age_Utilisateur`
- `Moyenne ← (Note1 + Note2) / 2`
- `Compteur ← Compteur + 1` (Cette opération s'appelle une incrémentation.)

Notes importantes :

- Le sens de la flèche : `A ← B` est radicalement différent de `B ← A`. Le premier copie la valeur de B dans A. Le second copie la valeur de A dans B.
- Compatibilité des types : La valeur de l'expression à droite doit être d'un type compatible avec la variable à gauche. On ne peut pas ranger "Bonjour" (une Chaîne) dans une variable Age (un Entier).
- Affectation n'est pas Égalité : En mathématiques, x = x + 1 est une équation fausse. En algorithmique, `X ← X + 1` est une instruction valide (une incrémentation). La flèche ← est une action, tandis que le = est un opérateur de comparaison (qui retourne VRAI ou FAUX).

3.2. L'Instruction de Lecture (Entrée)

Elle permet à l'algorithme d'obtenir des données de l'extérieur (généralement, ce que l'utilisateur

tape au clavier) pendant son exécution.

- Lecture de d'une variable Var : `Lire(Var)`
- On peut aussi lire plusieurs variables à la suite : `Lire(Var1, Var2, ⋯)`

Lorsque l'algorithme exécute une instruction Lire, il se produit les étapes suivantes :

- L'exécution de l'algorithme se met en pause.
- Le système attend que l'utilisateur saisisse une valeur au clavier et valide (généralement en appuyant sur la touche "Entrée").
- Une fois la valeur saisie, l'algorithme la récupère.
- Il affecte automatiquement cette valeur à la variable (ou aux variables) spécifiée entre les parenthèses.
- L'algorithme reprend son exécution à l'instruction suivante.

Note importante :

L'instruction Lire se contente d'attendre. Elle n'informe pas l'utilisateur de ce qu'elle attend. C'est pourquoi une instruction Lire doit presque toujours être précédée d'une instruction Ecrire pour guider l'utilisateur.

Exemple :

- `Ecrire("Quel est votre âge ? ")`
- `Lire(AgeUtilisateur)`

3.3. L'Instruction d'Écriture (Sortie)

Elle permet de communiquer des informations (des résultats, des messages) de l'algorithme vers l'extérieur (généralement, un affichage à l'écran).

- `Ecrire(Expression)`

On peut afficher plusieurs éléments en les séparant par des virgules.

L'instruction Ecrire évalue ce qui se trouve entre ses parenthèses et l'affiche à l'écran.

- Si c'est un texte littéral (une chaîne de caractères entre guillemets), il l'affiche tel quel.
- Si c'est une variable, il va lire la valeur actuellement stockée dans cette variable et c'est cette valeur qu'il affiche.
- Si c'est une expression (un calcul), il évalue d'abord l'expression et affiche le résultat.

Exemples :

- `Ecrire("Bonjour le monde !")`
  - Affiche le texte : Bonjour le monde !
- `Ecrire(Moyenne)`

    - Si la variable Moyenne contient 14.5, il affiche : 14.5
- `Ecrire("Votre moyenne est : ", Moyenne, " sur 20.")`
    - Combine texte et variable. Si Moyenne vaut 14.5, il affiche : Votre moyenne est : 14.5 sur 20.
- `Ecrire("Le double de 5 est : ", 5 * 2)`
    - Évalue 5 * 2 (qui vaut 10) et affiche : Le double de 5 est : 10

Note importante :

Les guillemets sont utilisées pour définir une chaîne de caractères. Si, par erreur, on les met autours du nom d'une variable, c'est son nom qui sera affiché non pas sa valeur.

- `Ecrire(Age)` : Affiche la valeur de la variable Age (ex: 25).
- `Ecrire("Age")` : Affiche le texte (le mot) "Age".