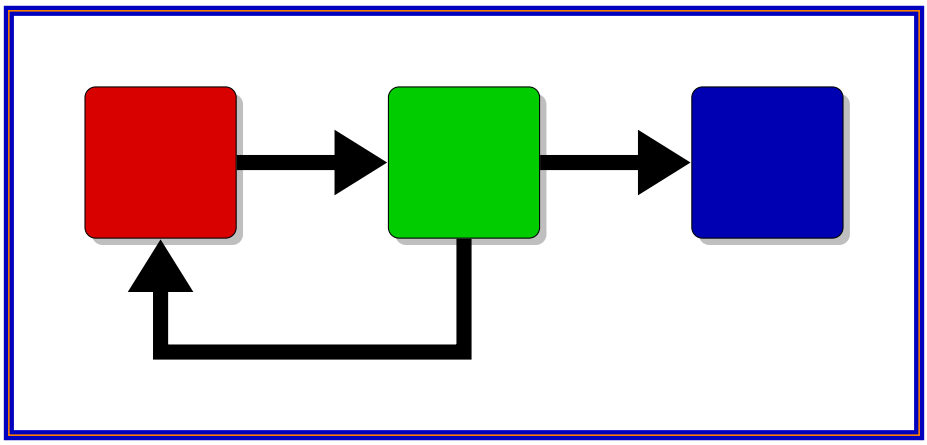


Introduction à l'algorithmique

2 septembre 2025

Remerciments

Diane Tremblay, Pierre Bard, Christian Barrette, département d'info et Steven Pigeon.

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Chapitre 1

Introduction

Le mot *algorithme* tire son origine du nom du mathématicien perse Al-Khwārizm, né vers 783 et mort vers 850 à Bagdad. On peut le créditer pour l'introduction du système de numération indo-arabe et de l'algèbre dans les mathématiques modernes. En effet, au 12ième siècle, la traduction en latin de certains de ses textes eut un impact profond sur le développement des mathématiques en Europe. C'est d'ailleurs de la traduction vers le latin de son nom, Algoritmi, que nous vient le mot *algorithme*.

Knuth, D. (1979). Algorithms in modern mathematics and computer science. Algorithms in Modern Mathematics and Computer Science.

Les algorithmes existent depuis très longtemps, avant même que l'idée des ordinateurs n'existe. Un des premiers algorithmes élaborés est l'anthyphérèse d'Euclide – aussi appelé l'algorithme d'Euclide – qui permet de déterminer le plus grand diviseur commun entre deux nombres naturels non nuls ; nous l'avons tous appris lors de notre passage à l'école. En fait, nous sommes entourés d'algorithmes s'adressant aux humains. Par exemple, une recette de gâteau au chocolat est un algorithme permettant de résoudre le problème « Comment faire un gâteau au chocolat ? » Il en va de même pour le livret contenant les instructions d'assemblage accompagnant un meuble à assembler : il contient l'algorithme permettant de résoudre le problème « Comment assembler les pièces séparées en un meuble ? »

« Les œuvres d'Euclide », traduit littéralement par F. Peyrard, nouveau tirage de mars 1993, Librairie scientifique et technique, Paris, 1993.

L'algorithmique est le fondement logique de la programmation.

En travaillant avec un langage de programmation, les étudiants et étudiantes doivent en apprendre le fonctionnement de base en plus de développer leur maitrise de l'al-

gorithmique. Si le langage choisi pour apprendre la programmation est complexe, il monopolise les efforts de compréhension et le temps de l'étudiant ou l'étudiante, au détriment de l'apprentissage de l'algorithmique, objet essentiel du cours. Il ou elle peut se retrouver dans la situation d'échouer sur les deux fronts : ne pas apprendre véritablement l'algorithmique et ne pas maitriser le langage de programmation pour exprimer l'algorithmique.

L'ordinateur fait du travail pour les humains : il produit des résultats. Ces résultats produits par la machine sont ses sorties et ce sont les fruits d'un travail algorithmique. Les étudiants de l'algorithmique pour ordinateurs font faire du travail à un ou des ordinateurs. L'algorithmique, base fondamentale de la programmation, est une activité abstraite où l'on ordonne, à l'aide d'opérations, la démarche à suivre pour produire les résultats désirés.

L'algorithmique est la science et l'art des algorithmes. Plus formellement, c'est la discipline couvrant « la connaissance des techniques utilisées pour construire des algorithmes et les méthodes d'investigation permettant de créer des algorithmes répondant à un problème donné » (Office québécois de la langue française, s.d.).

Dans sa définition la plus simple, un algorithme est « un ensemble d'étapes qui permettent d'accomplir une tâche » (Cormen, 2013, p. 1). Par exemple, pour le problème « Est-ce que N est un nombre pair ? », pour n'importe quelle valeur numérique entière de N, on peut créer un algorithme permettant à une personne ou à un ordinateur de répondre OUI ou NON.

Les algorithmes existent depuis très longtemps, avant même que l'idée des ordinateurs existe. Un des premiers algorithmes élaborés est l'anthyphérèse d'Euclide qui permet de déterminer le plus grand commun diviseur entre deux nombres naturels non nuls ; nous l'avons tous appris lors de notre passage à l'école secondaire. En fait, nous sommes entourés d'algorithmes s'adressant aux humains. Par exemple, une recette de gâteau au chocolat est un algorithme permettant de résoudre le problème « Comment faire un gâteau au chocolat ? » Il en va de même pour le livret contenant les instructions d'assemblage accompagnant un meuble à assembler : il contient l'algorithme permettant de résoudre le problème « Comment assembler les pièces séparées en un meuble ? »

Dans le cadre de l'informatique, l'algorithmique est appliquée à la programmation des ordinateurs. On veut pouvoir définir les étapes nécessaires à la résolution d'un problème pour que l'ordinateur puisse exécuter ces étapes et nous fournir le résultat. Tous les programmes informatiques sont des constructions d'algorithmes. La maitrise de la pensée algorithmique est un prérequis pour l'apprentissage de la programmation.

Lorsqu'il est destiné à un ordinateur, un algorithme est « un ensemble d'étapes qui permettent d'accomplir une tâche et qui est décrit de manière suffisamment précise pour qu'un ordinateur puisse l'exécuter » (Cormen, 2013, p.2).

Pour la construction d'un programme informatique, les opérations d'un algorithme sont décomposées jusqu'à ce qu'on puisse les arrimer aux instructions élémentaires d'un langage permettant de codifier des algorithmes pour ordinateurs. C'est ce qu'on appelle la programmation. Comme dans une recette de cuisine, c'est l'ensemble des instructions qui forme la procédure à suivre. Dans un ordinateur, il existe une multitude d'instructions possibles. De plus, chaque langage de programmation permet des instructions différentes tout en utilisant une syntaxe qui lui est propre.

En plus des instructions numériques, booléennes et de certaines permettant la manipulation de données textuelles, la plupart des langages fournissent aussi des instructions de sortie pour l'écriture à l'écran, à l'imprimante ou encore dans des fichiers de sortie. On peut habituellement aussi émettre du son et des commandes graphiques permettant de dessiner sur un de ces trois supports. À l'inverse, on peut obtenir des données, des entrées, à partir du clavier, de fichiers, mais aussi de différents périphériques tels que souris, caméra, micro, scanneur ou écran tactile.

Les instructions d'un programme sont les transpositions, dans un langage de programmation, des opérations algorithmiques qui furent décomposées successivement. Ces transpositions ne sont pas toujours directes et l'on doit parfois organiser les instructions entre elles à l'aide de structures de contrôle : ce sont les modes d'organisation des instructions. Il n'existe que trois structures de contrôle dans un ordinateur : la séquence, l'alternative (aussi appelée conditionnelle) et la boucle (aussi appelée répétitive).

Si les rudiments de la discipline d'algorithmique s'enseignent et s'apprennent relativement facilement, l'apprenant, pour devenir efficace, doit acquérir de l'expérience dans différents contextes et développer une certaine intuition. Dans ce sens, l'algorithmique constitue une discipline au seuil bas, mais au plafond élevé, c'est-à-dire que ses rudiments simples conduisent rapidement à la réalisation de tâches complexes sollicitant des opérations cognitives de haut niveau.

Première partie

Les bases de l'algorithmique

Chapitre 2

L'algorithmique dans la vie courante

blah

2.1 Séquence

La séquence est une suite ordonnée d'opérations et elle se lit toujours de haut en bas. La séquence est la plus simple des structures de contrôle de l'algorithmique pour ordinateurs.

Algorithme 1 : Faire une tasse de thé

1 Prendre un sachet de thé
2 Faire bouillir de l'eau
3 Prendre une tasse
4 Mettre le sachet de thé dans la tasse
5 Verser de l'eau bouillante dans la tasse
6 Attendre 5 minutes
7 Enlever le sachet de thé

Exercice : Trouver des exemples de séquences tirées de la vie courante.

2.2 Alternative

La vie quotidienne est remplie de situations où nous devons faire des choix : est-ce que je mets tel vêtement ou tel autre ? Est-ce que je prends mon parapluie ou pas ?

Algorithme 2 : Choisir un accessoire d'été

```
1 si il pleut alors
2 |  Prendre un parapluie
3 sinon
4 |  Prendre mes lunettes de soleil
```

Exercice : Trouver des exemples de d'alternatives tirées de la vie courante.

2.3 Boucle

Plusieurs activités présentent un caractère répétitif. Dans l'algorithmique se destinant aux ordinateurs, la structure de contrôle qui permet de reproduire cette caractéristique est la boucle.

Algorithme 3 : Monter un escalier

```
1 tant que il reste une marche à monter faire
2 |  Monter une marche
```

Exercice : Trouver des exemples de boucles tirées de la vie courante.

Chapitre 3

Variables

Les variables sont des conteneurs et elles contiennent des valeurs. Par exemple, la variable x peut contenir la valeur 1 (un). Si j'additionne 1 à la valeur contenue dans x, la variable x vaut maintenant 2.

Algorithme 4 : Exemple de variable

```
1 x ← 1
2 x ← x + 1
  // Commentaire : ici, x vaut 2
```

En programmation, les variables résident dans la mémoire vive (RAM) de l'ordinateur.

3.1 Déclaration et assignation

Pour déclarer une variable, on doit décider de **son type**, lui donner **un nom** et lui assigner une valeur initiale.

Le **type** de la variable représente la classe des valeurs pouvant y être contenues. Par exemple, une variable peut contenir un nombre (entier ou réel) ou encore une chaîne de caractères (*string*). En pseudocode, les types ne sont pas explicitement déclarés ; ils sont plutôt déduits à partir des noms et des valeurs initiales.

Le choix du nom de la variable est un acte important de l'algorithmique. Les variables doivent être bien nommées pour faciliter la compréhension de l'algorithme et sa maintenance dans le temps.

Critères pour la sélection des noms de variables :

1. Le nom indique correctement ce que la variable représente et les valeurs possibles qu'elle peut contenir.
2. Le nom est significatif et contribue positivement à la compréhension de l'algorithme.
3. Le nom est composé de peu de mots, idéalement un ou deux, peut-être trois ; rarement quatre ou plus.
4. Les espaces sont interdits dans les noms.
5. Les noms sont en minuscules.
6. Si plus d'un mot sont utilisés, ils seront discernables par la mise en majuscule de la première lettre de chaque mot suivant le premier.
7. En programmation, on veut habituellement éviter les accents dans les noms. De plus, beaucoup de langages de programmation prohibent que le premier caractère d'un nom soit un chiffre : il devra donc être habituellement une lettre ; les chiffres étant habituellement permis dans les autres caractères du nom. Pour le pseudocode, ces règles n'ont pas à être respectées.

L'**assignation** est l'opération d'assigner une valeur à une variable. L'opérateur la représentant est la flèche d'assignation $\leftarrow$. Dans un programme informatique, la valeur assignée est copiée dans la zone mémoire de la variable recevant la valeur.

Algorithme 5 : Assignation de la valeur 42 à x

```
1 x ← 42
```

L'**initialisation** d'une variable est l'assignation d'une valeur initiale à une variable. Il s'agit de la première assignation faites à une variable et, en pseudocode, l'initialisation crée la variable.

Lorsqu'une variable est bien nommée, le type peut habituellement être déduit. Logiquement, *nombrePommes* est un nombre entier tandis que la variable *Nom* est assurément une chaîne de caractères. Nous reviendrons plus en détails sur les types.

Des variables bien nommées :

Algorithme 6 : Des variables bien nommées

```
1 nombrePommes ← 4
2 nom ← "Luke"
3 total ← 0
4 estActif ← Vrai
```

Des variables mal nommées :

Algorithme 7 : Des variables mal nommées

```
1 ABC ← "Han"
  // ABC ne représente pas bien le contenu et est en majuscules
2 asdfg ← 1234567890
  // asdfg n'est pas un nom significatif
3 nombrePommes ← 3,14159265
  // un nombre de pommes devrait être un nombre entier
```

3.2 Entrées et sorties

La plupart des langages de programmation fournissent des instructions de sortie pour l'écriture à l'écran, à l'imprimante ou encore dans des fichiers de sortie. On peut habituellement invoquer des commandes graphiques permettant de dessiner sur un de ces trois supports ou encore émettre du son.

À l'inverse, on peut obtenir des données en entrées, à partir du clavier, de fichiers, mais aussi de différents périphériques tels que souris, caméra, micro, scanneur ou écran tactile.

En pseudocode, l'écriture et la lecture de valeurs dans une console texte priment, bien qu'il soit aussi possible d'exécuter des algorithmes sur papier pour produire des dessins si l'exécuteur manipule le crayon en suivant les opérations de l'algorithme.

Il est possible d'obtenir une valeur auprès de l'utilisateur. Pour l'instant, nous prendrons pour acquis que l'utilisateur est parfait, c'est-à-dire qu'il n'entre jamais de valeur erronée. On doit fournir la variable dans laquelle stocker la valeur obtenue.

Algorithme 8 : Obtenir

```
1 obtenir nom
```

Il est aussi possible d'écrire des nombres ou des chaînes de caractères à l'écran. Chaque appel à la commande ***écrire*** produit une ligne à l'écran.

Algorithme 9 : Écrire Bonjour et 42 sur deux lignes distinctes

```
1 écrire "Bonjour"
2 écrire 42
```

Pour la création de tableaux à l'écran, on peut soumettre une liste de valeurs à écrire

pour obtenir une organisation en colonnes. Chaque élément de la liste à écrire est séparée par une virgule.

Algorithme 10 : Écrire un tableau en colonnes

```
1 écrire "Prénom", "Nom"
2 écrire ""
3 écrire "Alan", "Turing"
4 écrire "Donald", "Knuth"
5 écrire "Donald", "Knuth"
6 écrire 2018, 1000000
```

Prénom	Nom
Alan	Turing
Donald	Knuth
Eugene	Djikstra

3.3 Opérations arithmétiques

Les opérations arithmétiques habituelles sont supportées, ainsi on peut additionner (+), soustraire (-), multiplier (*) et diviser (/). La même priorité des opérateurs qu'en mathématiques est applicable, ainsi les multiplications et divisions sont évaluées en premières. Les parenthèses sont aussi supportées et elles sont utilisées, comme en mathématiques, pour ajuster les priorités d'évaluation.

Algorithme 11 : Opérations arithmétiques

```
1 six ← 2 + 2 * 2
2 huit ← (2 + 2) * 2
3 deux ← huit − six
4 pi ← 3, 14159265
5 circonference ← 2 * pi * 10                    // rayon de 10 unités
6 moyenne1a5 ← (1 + 2 + 3 + 4 + 5)/5
```

3.4 Concaténation

La **concaténation** est l'opération permettant de coller ensemble deux chaînes de caractères pour n'en former qu'une seule. La concaténation s'effectue avec l'opérateur +

entre deux chaînes de caractères.

En pseudocode, on peut aussi concaténer une chaîne de caractères et un nombre, et vice-versa : le nombre sera converti automatiquement et temporairement à une chaîne de caractères le représentant. Cette conversion automatisée de type est appelée le transtypage ou, en anglais, un *type cast*.

Algorithme 12 : Concaténation : exemples

```
1 écrire "Bonjour␣" + "tout le monde !"
2 écrire "Il fait␣" + temperature + "␣°C"      // temperature est un nombre
```

Voici un exemple dans lequel on concatène un prénom, une espace et un nom de famille pour former un nom complet.

Algorithme 13 : Concaténation : dire un bonjour personnalisé

```
1 écrire "Entrer votre prénom :␣"
2 obtenir prenom
3 écrire "Entrer votre nom de famille :␣"
4 obtenir nomFamille
5 nomComplet ← prenom + "␣" + nomFamille
6 écrire "Bonjour␣" + nomComplet + " !"
```

3.5 Exercices

1. Déclarer et initialiser une variable contenant votre prénom.
2. De quel type est une variable contenant votre prénom ?
3. Déclarer et initialiser une variable contenant votre année de naissance.
4. De quel type est une variable contenant votre année de naissance ?
5. Déclarer et initialiser une variable contenant l'aire d'un cercle d'un rayon de 10 unités sachant que son aire est donnée par la formule $AireCercle = \pi * Rayon^2$.
6. De quel type est une variable contenant l'aire d'un cercle ?
7. Obtenir le nom et l'année de naissance de l'utilisateur pour faire afficher son nom suivi de son année de naissance entre parenthèses. Par exemple, pour Alan Turing né en 1912, on obtient « Alan Turing (1912) » à l'écran.

Chapitre 4

Expressions, types et opérations

4.1 Expressions

4.1.1 Notions

En algorithmique et dans les langages de programmation, « une expression est un élément de syntaxe qui combine un ensemble de lexèmes retournant une valeur. C'est une combinaison de littéraux, de variables, d'opérateurs, et de fonctions qui est évaluée (ou calculée) en suivant les règles de priorité et d'associativité du langage de programmation pour produire (ou retourner) une nouvelle valeur. »

Source : https ://fr.wikipedia.org/wiki/Expression_(informatique)

Les expressions s'évaluent selon les types et les opérations utilisés pour la décrire. Par exemple, l'opérateur + indique une addition lorsqu'il est entre deux nombres et il représente aussi la concaténation lorsqu'il se retrouve entre deux chaînes de caractères.

Algorithme 14 : L'opérateur + utilisé pour l'addition et la concaténation

```
1 grandTotal ← sousTotal + montantTaxes
2 nomComplet ← prenom + "␣" + nom
```

Pour bien s'approprier une expression algorithmique, il est important de porter particulièrement attention aux types manipulés. L'expression est bien formée et valide si les types utilisés dans l'expression sont compatibles avec les opérations utilisées. On s'intéresse ensuite aux valeurs potentiellement contenues dans ces types. Par exemple, sachant qu'un nombre est entier, on peut se demander si le nombre pourra ou non être négatif.

Les opérations d'un algorithme sont des constructions d'expressions. Chaque ligne d'un algorithme peut exécuter plus d'une opération si elles sont imbriquées correctement dans des expressions. L'imbrication correcte des opérations se fait en respectant les types attendus pour les opérandes.

Le résultat de l'évaluation d'une expression est une valeur d'un certain type. Cela permet d'imbriquer les expressions les unes dans les autres, tant que les types sont compatibles avec les opérations utilisées.

4.1.2 Exemples d'expressions

L'algorithme suivant calcule le rayon d'un cercle à partir de sa circonférence :

Algorithme 15 : Calculer le rayon d'une cercle

```
1 écrire "Entrer la circonférence du cercle :␣"
2 obtenir circonf
3 rayon ← (circonf/π)/2
4 écrire "Le rayon du cercle est␣" + rayon
```

Cet algorithme est composé de multiples expressions ; les voici :

1. La ligne 1 contient une seule expression, de type String : *"Entrer la circonférence du cercle␣ :"*. Il s'agit d'une valeur littérale (aussi appelé *un littéral*), ce qui veut dire que la valeur est contenue directement dans le pseudocode de l'algorithme.
2. La ligne 2 contient une seule expression, de type Nombre : la variable *circonf*.
3. La ligne 3 contient plusieurs expressions :
   (a) *circonf* : une variable de type Nombre ;
   (b) π : un littéral de type Nombre ;
   (c) *2* : un littéral de type Nombre ;
   (d) *(circonf/*π*)* : une expression de type Nombre puisqu'elle représente la division entre deux nombres ;
   (e) *(circonf/*π*)*/2 : aussi de type Nombre.
   (f) *rayon* : une variable de type Nombre.
4. La ligne 4 contient aussi plusieurs expressions :
   (a) *"Le rayon du cercle est␣"* : un littéral de type String.

(b) *rayon* : variable de type Nombre, transtypé automatiquement et temporairement en une String. L'opération de transtypage s'exécute automatique puisque l'expression de concaténation attend une string pour chacun de ses opérandes ;

(c) *"Le rayon du cercle est␣" + rayon* : une expression produisant une String et représentant une concaténation.

4.2 Nombres et opérations numériques

Les nombres supportent les opérations arithmétiques usuelles : addition (+), soustraction (-), multiplication (-) et division (/). La priorité des opérateurs est la même qu'en mathématiques et les parenthèses sont utilisées pour les parenthèses sont utilisées pour ajuster l'ordre d'évaluation de certaines expressions.

En plus des opérations arithmétiques, il est possible de faire générer un nombre au hasard par l'ordinateur. En fait, puisque les ordinateurs sont des machines numériques purement déterministes, les nombres au hasard sur ordinateurs sont appelés des *nombres pseudo-aléatoires* : ce sont des nombres générés à l'aide de calculs, mais ils donnent l'impression d'être au hasard. Il existe plusieurs techniques pour générer des nombres pseudo-aléatoires, mais nous allons nous intéresser à leur utilisation dans un cadre algorithmique plutôt qu'à leur génération par programmation.

Algorithme 16 : Des nombres pseudo-aléatoires

```
1 valeurDé ← piger un nombre entier dans l'intervalle [1, 6]
2 onOff ← piger un nombre entier dans l'intervalle [0, 1]
3 facteurMultiplication ← piger un nombre réel dans l'intervalle [0, 1]
```

4.3 Texte et opérations textuelles

Les éléments textuels d'un algorithme sont contenues dans des chaînes de caractère, communément appelées des *strings*. Les strings sont représentés dans le code par des caractères entre guillements doubles : "cette phrase est une string". La string "" représente la chaîne vide (string vide).

En plus de la concaténation (opérateur + sur deux opérandes de type string), il est possible de demander le caractère se trouvant à une position donnée avec **l'opération d'indexation**, représentée par l'opérateur [ ]. Pour accéder au $n$ième caractère d'une string nommée *str*, on utilise *str[n]*. Le premier caractère d'une string est à l'indice 0.

Ainsi, pour obtenir le premier caractère de la variable string *str*, on utilise *str[0]*.

Algorithme 17 : Générer les initiales d'une personne

```
1 écrire "Prénom :␣"
2 obtenir prenom
3 écrire "Nom :␣"
4 obtenir nom
5 initiales ← prenom[0] + nom[0]
6 initiales ← initiales en majuscules
7 écrire "Les initiales sont :␣" + initiales
```

On peut demander de mettre en majuscules ou en minuscules une string ou un caractère.

Il est aussi possible de demander la **longueur d'une string** à l'aide de la opération « longueur de ». Cette opération prend une string et retourne un nombre contenant la longueur de la string en entrée.

Puisque le premier caractère d'une string se retrouve à l'indice zéro, pour accéder au dernier caractère d'une chaîne, on indexe à l'indice « longueur de la chaîne - 1 ».

Algorithme 18 : La dernière lettre d'un mot

```
1 écrire "Entrer un mot :␣"
2 obtenir mot
3 écrire "La dernière lettre du mot est␣" + mot[longueur de mot-1]
```

4.4 Booléens, opérateurs logiques et opérateurs relationnels

Les valeurs booléennes Vrai et Faux sont de type booléen. Ce sont les valeurs de base de l'algèbre de Bool. Les opérateurs logiques usuels de l'algèbre booléenne s'appliquent : ET, OU, NON, etc. Comme pour les opérations arithmétiques, les parenthèses sont utilisées pour ajuster l'ordre d'évaluation de certaines expressions.

Table de vérité de l'opérateur logique ET

a	b	a ET b
V	V	V
V	F	F
F	V	F
F	F	F

Table de vérité de l'opérateur logique OU

a	b	a OU b
V	V	V
V	F	V
F	V	V
F	F	F

Table de vérité de l'opérateur logique NON

a	NON a
V	F
F	V

Les opérateurs relationnels comparent deux nombres et émettent une valeur booléenne :

Opérateurs relationnels

Opérateur	Nom	Exemple
$=$	Égal	a = b
$\neq$	Non-égal	a $\neq$ b
$<$	Plus petit	a $<$ b
$\leq$	Plus petit ou égal	a $\leq$ b
$>$	Plus grand	a $>$ b
$\geq$	Plus grand ou égal	a $\geq$ b

Les booléens sont transtypés automatiquement et temporairement en string (”vrai” ou ”faux”) lorsqu'ils sont utilisés dans un contexte textuel.

Algorithme 19 : Plus petit ?

```
1 obtenir a
2 obtenir b
3 estPlusPetit ← (a < b)
4 écrire "Il est␣" + estPlusPetit + "␣que␣" + a + "␣<␣" + b
```

4.5 Exercices

1. Donner le **type** des expressions suivantes :
   (a) ”Bonjour”
   (b) 42

(c) ”42”

(d) 4 + 4

(e) (2 + 2) * 2

(f) 2 * π * rayon

(g) ”Bonjour” + ”␣le␣” + ”monde”

(h) ”La circonférence est␣” + (2 * π * r)

(i) $8 \leq 8$

(j) $1 < 10$

(k) Vrai ET Faux

(l) ($1 < 10$) OU ($5 > 2$)

2. Donner la **valeur** des expressions suivantes :

(a) 42

(b) 2 + 3

(c) ”42”

(d) (2 + 2) * 2

(e) ”Bonjour” + ”␣le␣” + ”monde”

(f) ”La circonférence du cercle unitaire est␣” + (2 * π)

(g) $8 \leq 8$

(h) $1 < 10$

(i) Vrai OU Faux

(j) ($1 < 10$) ET ($5 > 2$)

3. Les expressions suivantes sont invalides. Pourquoi ?

(a) ”Bonjour” * 24

(b) 42 - ”23”

(c) 42 < ”Bonjour”

(d) **obtenir** ”Quel est votre nom ?␣”

4. Tracer l'algorithme ci-dessous pour les valeurs suivantes :

(a) a = 3 et b = 5

(b) a = 18 et b = 9

Algorithme 20 : Plus petit ou plus grand ?

```
1 obtenir a
2 obtenir b
3 estPlusPetit ← (a < b)
4 écrire "Il est␣" + estPlusPetit + "␣que␣" + a + "␣<␣" + b
5 estPlusGrand ← (a > b)
6 écrire "Il est␣" + estPlusGrand + "␣que␣" + a + "␣>␣" + b
```

Chapitre 5

Séquence

5.1 Notions

La séquence est une suite ordonnée d'opérations. Une séquence se lit toujours de haut en bas.

Le *pseudocode* est la forme textuelle de l'algorithme tandis que l'*algorigramme* en est la représentation graphique. Dans le cas d'une séquence, à chaque ligne de code correspond un rectangle dans l'algorigramme.

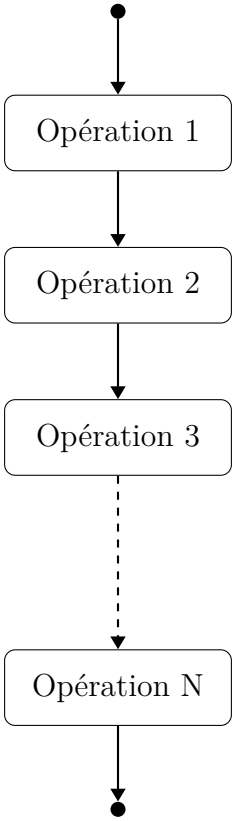


FIG. 5.1 : La forme générale de la séquence

L'algorithme qui suit contient la séquence que nous avons élaborée pour décrire la préparation d'une tasse de thé. Bien entendu, ce n'est pas la seule séquence susceptible de décrire cette activité.

Algorithme 21 : Faire une tasse de thé

```
1 Prendre un sachet de thé
2 Faire bouillir de l'eau
3 Prendre une tasse
4 Mettre le sachet de thé dans la tasse
5 Verser de l'eau bouillante dans la tasse
6 Attendre 5 minutes
7 Enlever le sachet de thé
```

À partir du moment où une séquence est énoncée, **chaque opération qu'elle contient est obligatoire**. C'est donc dire qu'on ne peut sauter aucune opération lors de l'exécution.

Chaque opération d'une séquence est soit élémentaire, soit décomposable.

L'opération élémentaire est celle qu'on ne juge pas utile de décomposer. Une opération est décomposable si on peut la découper en un ensemble d'opérations plus détaillées. Par exemple, pour faire bouillir de l'eau comme il est demandé dans l'algorithme pour faire une tasse de thé, on peut exécuter l'algorithme suivant.

Algorithme 22 : Faire bouillir de l'eau

```
1 Mettre de l'eau dans la bouilloire
2 Brancher la bouilloire
3 Attendre que l'eau bouille
```

La notion de décomposition prendra tout son sens quand nous étudierons le développement graduel.

5.2 Exemples de séquences

Voici quelques exemples de séquences.

L'algorithme suivant est très simple, il s'agit d'une séquence d'une seule opération :

Algorithme 23 : Dire bonjour

```
1 écrire "Bonjour !"
```

Le prochain exemple calcule l'âge de l'utilisateur. L'âge calculée est approximative car l'algorithme ne tient pas compte des mois de l'année.

Algorithme 24 : Calculer l'âge

```
1 écrire "Votre année de naissance : "
2 obtenir anne
3 anneEnCours ← l'année en cours telle que configurée dans l'ordinateur
4 écrire "Vous avez environ " + (anneEnCours − anne) + " ans."
```

L'algorithme ci-dessous calcule le carré d'une valeur obtenue auprès de l'utilisateur. L'opération de multiplication est imbriquée dans une concaténation.

Algorithme 25 : Calculer le carré d'une valeur

```
1 écrire "Entrer une valeur pour obtenir son carré : "
2 obtenir nombre
3 écrire "Le carré de " + nombre + " est " + nombre ∗ nombre
```

L'exemple suivant est une séquence permutant les valeurs de deux variables. Pour y arriver, l'algorithme utilise une troisière variable nommée *temp* pour « temporaire ».

Algorithme 26 : Permutation des valeurs de x et y

```
1 obtenir x
2 obtenir y
3 écrire "Valeur de x :␣" + x
4 écrire "Valeur de y :␣" + y
5 temp ← x
6 x ← y
7 y ← temp
8 écrire "Valeur permutée de x :␣" + x
9 écrire "Valeur permutée de y :␣" + y
```

5.3 Exercices de séquences

1. Élaborer un algorithme permettant de convertir une valeur d'une unité de mesure à une autre unité. La valeur à convertir est obtenue auprès de l'utilisateur. Par exemple : un convertisseur de Celsius à Fahrenheit.

2. Élaborer un algorithme générant une facture à partir du montant avant taxes : le sous-total. À partir du sous-total obtenu auprès de l'utilisateur, calculer et afficher les montants de taxation sur les achats s'appliquant où vous habitez. Sur la facture, indiquer le sous-total et, pour chaque taxe : le nom de la taxe, le pourcentage s'appliquant et le montant de la taxe. Finalement, la facture doit aussi afficher le grand total à payer. Utiliser Internet pour trouver les pourcentages de taxation et la procédure pour les calculer.

Chapitre 6

Tables de traçage

6.1 Notions

La table de traçage de l'exécution d'un algorithme est un outil primordial pour sa compréhension et sa validation. C'est un outil puissant pour détecter les erreurs d'algorithmique et faciliter l'application des correctifs nécessaires.

Bien que la table de traçage ne soit pas une preuve qu'un algorithme fonctionne pour toutes les données en entrées possibles, elle permet de démontrer qu'il fonctionne pour les données utilisées lors du traçage.

Considérer l'algorithme suivant qui demande le nom de l'utilisateur et qui le salut à l'aide de son nom :

Algorithme 27 : Dire bonjour avec le nom

```
1 écrire "Votre nom :␣"
2 obtenir nom
3 écrire "Bonjour␣" + nom
```

La table de traçage suivante est obtenue lorsqu'une utilisatrice nommée « Ada » utilise l'algorithme :

Table de traçage

Ligne	nom	Écran
2	Ada	Votre nom : Ada Bonjour Ada

La table de traçage permet d'étaler l'évolution des valeurs contenues dans les variables d'un algorithme ainsi que le contenu final de l'écran.

L'ensemble des valeurs des variables à n'importe quel moment lors de l'exécution d'un algorithme est connu sous le nom de **l'état de l'algorithme**. Pour l'instant, les opérations d'un algorithme vont soit modifier son état, c'est-à-dire changer la valeur d'une variable, ou soit afficher à l'écran.

Chacune des lignes d'une table de traçage représente un état de l'algorithme à un moment donné dans son exécution. Chaque fois qu'une opération change l'état de l'algorithme, une nouvelle ligne est ajoutée dans la table de traçage.

La première colonne de la table de traçage est titrée « Ligne » et elle contient les numéros des lignes de l'algorithme qui ont changé l'état de l'algorithme. La dernière colonne contient le contenu final de l'écran et elle est remplie graduellement par l'exécution des opérations d'écriture à l'écran. Les autres colonnes, celles du milieu, contiennent les valeurs des variables au fil de l'exécution de d'un algorithme.

6.2 Exemples de tables de traçage

Algorithme 28 : Changement de signe

```
1 obtenir nb
2 negNb ← −1 ∗ nb
3 écrire "Changement de signe :␣" + negNb
```

Table de traçage

Ligne	nb	negNb	Écran
1	6		nb : 6
2	6	-6	Changement de signe : -6

Table de traçage

Ligne	nb	negNb	Écran
1	-12		nb : -12
2	-12	12	Changement de signe : 12

Algorithme 29 : Permutation des valeurs de x et y

```
1 obtenir x
2 obtenir y
3 écrire "Valeur de x :␣" + x
4 écrire "Valeur de y :␣" + y
5 temp ← x
6 x ← y
7 y ← temp
8 écrire "Valeur permutée de x :␣" + x
9 écrire "Valeur permutée de y :␣" + y
```

Table de traçage

Ligne	x	y	temp	Écran
1	3			x : 3
2	3	4		y : 4
5	3	4	3	Valeur de x : 3
6	4	4	3	Valeur de y : 4
7	4	3	3	Valeur permutée de x : 4
				Valeur permutée de y : 3

6.3 Exercices

1. Tracer l'algorithme suivant pour un chien âgé de 3 ans :

   Algorithme 30 : Calculer l'âge d'un chien en années humaines

   ```
   1 écrire "Entrer l'âge du chien en années :␣"
   2 obtenir age
   3 ageHumain ← 7 ∗ age
   4 écrire "Le chien a␣" + ageHumain + "␣années humaines."
   ```

2. Tracer l'algorithme de permutation des valeurs de deux variables pour un utilisateur qui entre 12 et 34 pour les valeurs de x et y respectivement.

3. Tracer votre algorithme de génération de facture du chapitre sur la séquence, pour un sous-total de 100$. Si votre algorithme ne produit pas le résultat attendu, appliquer les correctifs nécessaires et refaire un traçage pour s'assurer du bon fonctionnement.

Chapitre 7

Alternative

7.1 Notions

La vie quotidienne est remplie de situations où nous devons faire des choix : est-ce que je mets tel vêtement ou tel autre ? Est-ce que je prends mon parapluie ou pas ?

L'alternative est une structure dans laquelle, suivant l'évaluation d'une condition, l'une ou l'autre de deux séquences est exécutée.

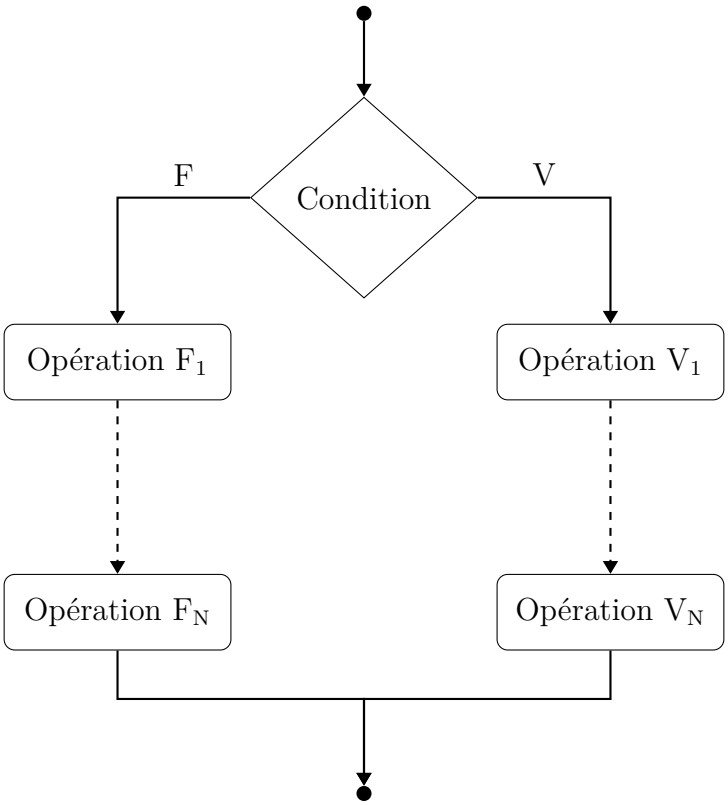


FIG. 7.1 : La forme générale de l'alternative simple

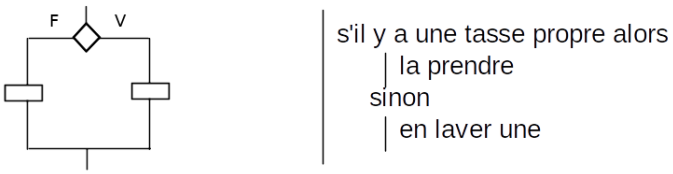


FIG. 7.2 : Prendre une tasse propre

La condition de l'alternative est une expression de type booléenne, c'est à dire qu'on obtient Vrai ou Faux lorsqu'on évalue l'expression.

Le losange représente la condition à évaluer. Nous convenons de situer dans la branche du Vrai à droite. Comme l'illustre la prochaine figure, c'est celle qu'on emprunte lorsque la condition se réalise. Elle correspond à l'opération 1 du pseudocode. Il en découle que c'est la branche du Faux, celle de gauche, que l'on va emprunter quand la condition ne s'évalue pas à Vrai.

Il est toujours possible de déroger à la convention gauche-droite précédente. Il suffit

pour cela d'inverser les symboles V et F indiquant la direction à suivre sur l'une des sorties latérales du losange.

La forme générale de l'alternative ne fait apparaître qu'un seul rectangle dans chacune des branches de l'algorigramme. Il faut comprendre que là où on peut mettre un rectangle, on peut mettre autre chose, y compris une séquence. C'est ce que montre le prochain exemple qui contient en outre une opération à la sortie de l'alternative. Cette opération, qui ne fait pas partie de l'alternative, est située dans le pseudocode au même niveau que le mot si. Les opérations qui constituent chacune des branches de l'alternative sont, quant à elles, indentées.

La branche de droite de l'algorigramme contient deux opérations (correspondant à D-1 et D-2), tandis que la branche de gauche en contient trois (correspondant à G-1, G-2 et G-3). Dans le pseudocode, ces deux blocs d'opérations sont séparés par le mot sinon qui, lui, apparaît avec un décalage moindre. On dit qu'il n'a subi qu'une demi-indentation.

L'opération 1 est exécutée si et seulement si la condition est vraie et l'opération 2 est exécutée si la condition est fausse. La condition évaluée pour sélectionner quel chemin d'exécution emprunter est une expression de type booléen. L'utilisation d'opérateurs relationnels est une des façons de générer une expression booléenne.

Les opérations 1 et 2 peuvent en fait être plusieurs opérations puisqu'on pourra les décomposer pour créer plusieurs sous-opérations.

Il n'est pas nécessaire d'exécuter une ou plusieurs opérations lorsque la condition est fausse. Dans ce cas, on ne fait qu'omettre la partie Sinon de l'alternative :

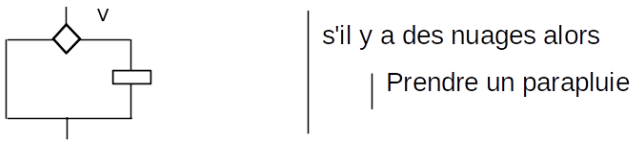


FIG. 7.3 : Prendre ou non son parapluie

Algorithme 31 : Positif ou négatif ?

```
1 obtenir nombre
2 si nombre < 0 alors
3 |  écrire nombre + " est négatif"
4 sinon
5 |  écrire nombre + " est positif"
```

Algorithme 32 : Comparer deux nombres

```
1 obtenir a
2 obtenir b
3 si a < b alors
4 |  écrire a + "␣est plus petit que␣" + b
5 sinon
6 |  écrire a + "␣est plus grand ou égal à␣" + b
```

L'algorithme qui suit permet de déterminer le plus petit de deux nombres :

Algorithme 33 : Trouver le plus petit de deux nombres (version 1)

```
1 obtenir a
2 obtenir b
3 plusPetit ← 0
4 si a < b alors
5 |  plusPetit ← a
6 sinon
7 |  plusPetit ← b
8 écrire plusPetit + "␣est le plus petit"
```

L'algorithme suivant produit le même résultat que l'algorithme précédent, mais il prend pour acquis que le plus petit nombre est b et corrige pour a si $(a < b)$:

Algorithme 34 : Trouver le plus petit de deux nombres (version 2)

```
1 obtenir a
2 obtenir b
3 plusPetit ← b
4 si a < b alors
5 |  plusPetit ← a
6 écrire plusPetit + "␣est le plus petit"
```

7.2 Blocs d'opérations et portée des variables

Les blocs d'opérations sont les opérations indentées sous les lignes « si » et « sinon » de l'alternative. Fondamentalement, un bloc d'opérations est une séquence qui peut être composée d'autres séquences, d'alternatives et/ou de boucles.

Dans la plupart des langages de programmation, les variables ne survivent que pour la

durée du bloc dans lequel elles ont été déclarées, c'est ce qu'on appelle la portée d'une variable. Nous allons utiliser le bloc comme portée pour les variables du pseudocode. Si deux variables sont utilisées dans deux blocs différents, ce sont en fait deux variables différentes puisqu'elles n'existent pas en dehors du bloc d'opération où elles ont été déclarées (en pseudocode : utilisées pour la première fois).

Ainsi, l'algorithme suivant est erroné – on dit qu'il contient une erreur de syntaxe, car il ne respecte pas les règles du pseudocode pour l'algorithmique.

Algorithme 35 : Trouver le plus petit de deux nombres (erreur de syntaxe)

```
1 obtenir a
2 obtenir b
  // Cette ligne de code a été retirée : plusPetit ← 0
3 si a < b alors
4 |  plusPetit ← a
5 sinon
6 |  plusPetit ← b
7 écrire plusPetit + " est le plus petit"      // Erreur: plusPetit n'existe
  pas!
```

7.3 Exemples d'alternatives

L'exemple suivant détermine si un nombre obtenu auprès de l'utilisateur est dans l'intervalle de 1 à 100 inclusivement :

Algorithme 36 : Est-ce que le nombre est dans l'intervalle ?

```
1 écrire "Entrer un nombre dans l'intervalle [1, 100] : "
2 obtenir nombre
3 si (nombre >= 1) ET (nombre <= 100) alors
4 |  écrire "Le nombre est dans l'intervalle."
5 sinon
6 |  écrire "Le nombre n'est pas dans l'intervalle."
```

Finalement, un jeu de pile ou face :

Algorithme 37 : Jeu de pile ou face

```
   // Obtenir l'entrée de l'utilisateur
 1 écrire "Jouons : pile ou face ?␣"
 2 obtenir choix
 3
   // Lancer la pièce
 4 valeurPiece ← piger un nombre entier dans [0,1]     // pile = 0, face = 1
 5 cotePiece ← "pile"                                  // On prédit que 0 est pigé...
 6 si valeurPiece = 1 alors
 7 |  cotePiece ← "face"                               // ...et on corrige si c'est 1
 8
   // Écrire le résultat
 9 écrire "La pièce tombe sur␣" + cotePiece + "."
10 si choix = cotePiece alors
11 |  écrire "Vous avez gagné !"
12 sinon
13 |  écrire "Désolé !"
```

7.4 Exercices d'alternatives

1. Tracer l'algorithme du jeu de pile ou face pour les parties suivantes :
   (a) choix : "pile", valeurPiece : 0
   (b) choix : "pile", valeurPiece : 1

2. Élaborer un algorithme permettant d'écrire à l'écran la valeur absolue d'un nombre obtenu auprès de l'utilisateur. La valeur absolue d'un nombre est la valeur d'un nombre sans son signe. Par exemple, la valeur absolue de -4 est 4 tandis que la valeur absolue de 7 est 7.

3. Élaborer un algorithme permettant à un utilisateur de convertir une valeur d'une unité de mesure à une autre tout en s'assurant que l'utilisateur n'entre pas une valeur négative. Dans le cas malheureux où il entre une valeur négative, l'avertir à l'aide d'un message d'erreur.

4. Élaborer un algorithme déterminant le prix d'entrée au zoo selon l'âge entré par l'utilisateur. Le zoo est gratuit pour les enfants de 17 ans et moins et le prix d'entrée est de 20$ pour un adulte.

5. Élaborer un algorithme capable d'initialiser au hasard un personnage de jeu vidéo. Ce personnage a les caractéristiques suivantes :

(a) Une classe de personnage sélectionnée aléatoirement : soit "magicien" ou "chevalier".

(b) Des points de vie entiers initialisés aléatoirement entre 60 et 100 inclusivement.

(c) Une quantité de mana (nombre réel) aléatoire entre 500 et 1000 s'il s'agit d'un magicien ou de 0 pour un chevalier.

Pour chaque caractéristique du personnage, déclarer une variable et assigner lui une valeur respectant les règles énoncées.

7.5 L'alternative généralisée

L'alternative généralisée permet de traiter les conditions pour lesquelles il existe plus de deux possibilités.

L'alternative généralisée a la forme suivante :

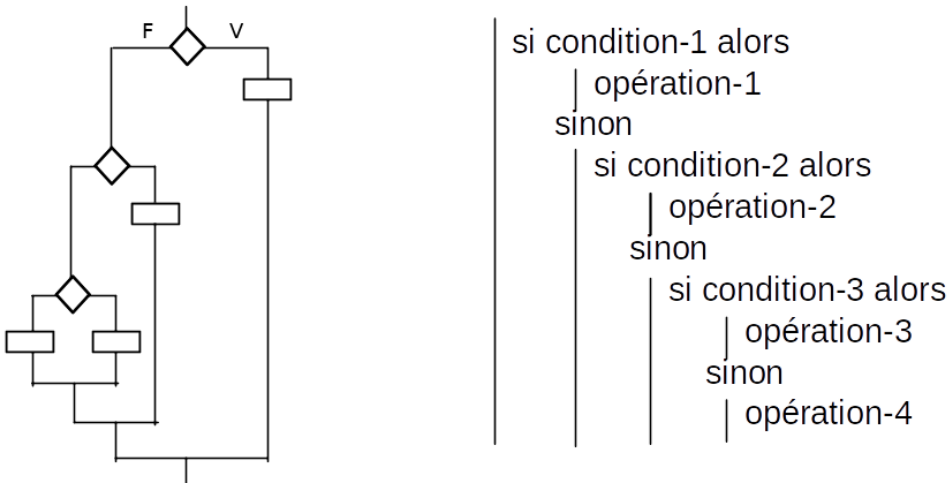


FIG. 7.4 : Forme générale de l'alternative généralisée

Cette disposition des alternatives, bien que formellement juste, n'est pas souhaitable pour deux raisons :

1. Dans une imbrication d'alternatives disposées en cascade, nous avons tendance à considérer la première alternative comme plus importante que celles qui la suivent puisque qu'elle n'est pas indentée, mais toutes les possibilités ont strictement le même poids logique, aucune n'étant plus importante qu'une autre.

2. Si le choix doit se faire parmi un grand nombre de possibilités (10, par exemple), le pseudocode s'étendra de façon exagérée en largeur et vers la droite au point où il ne restera plus assez de place pour les instructions. L'espace est limité : la largeur habituelle d'une page de code imprimée est de 80 colonnes et les indentations sont d'une largeur de 4 espaces. Même si le compilateur permet de dépasser cette limite, il n'est pas souhaitable de le faire, car il est très désavantageux d'avoir une page de code imprimée où les lignes sont tronquées ou reporter sur plusieurs lignes. Quelques normes de programmation permettent de se rendre à 100 colonnes de caractères par ligne.

La disposition suivante, dans laquelle tous les losanges, correspondant aux conditions, et tous les rectangles, correspondant aux opérations, sont respectivement alignés verticalement l'un avec les autres.

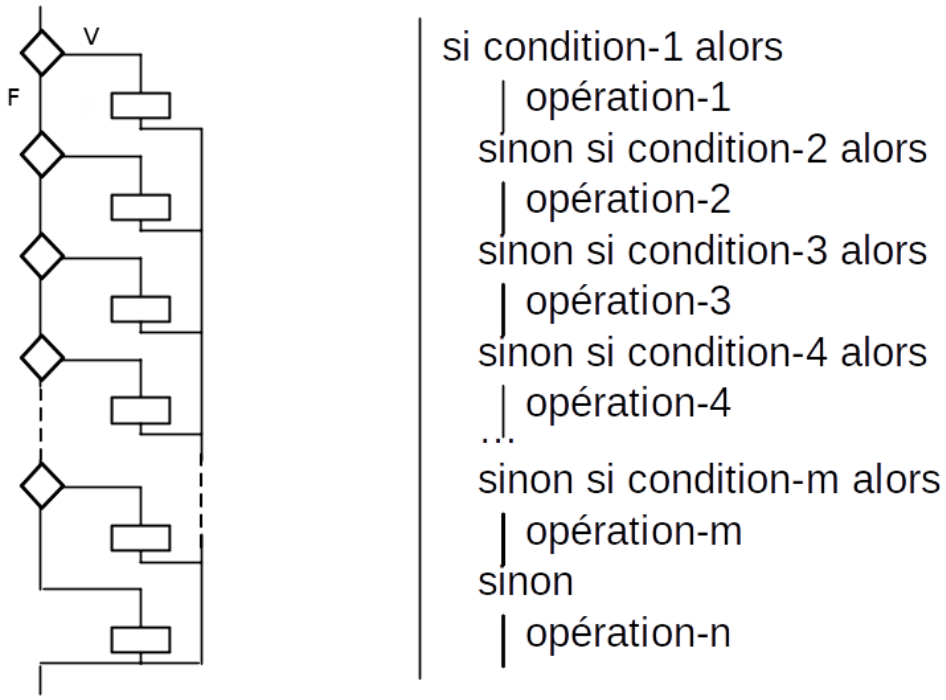


FIG. 7.5 : Forme générale recommandée de l'alternative généralisée

7.6 Exemples d'alternatives généralisées

Par exemple, pour choisir un accessoire selon la saison en cours, on doit évaluer si la saison est l'été et si ce ne l'est pas, évaluer si la saison est l'automne et ainsi de suite.

Algorithme 38 : Choisir un accessoire selon une saison aléatoire

```
1  saison ← piger une string dans ["été", "automne", "hiver", "printemps"]
2  accessoire ← ""                                  // chaîne vide (longueur zéro)
3  si saison = "été" alors
4  |  accessoire ← "parasol"
5  sinon si saison = "automne" alors
6  |  accessoire ← "chapeau"
7  sinon si saison = "hiver" alors
8  |  accessoire ← "foulard"
9  sinon si saison = "printemps" alors
10 |  accessoire ← "parapluie"
11
12 écrire "C'est l'" + saison + ", avez-vous besoin d'un␣" + accessoire + "?"
```

On peut spécifier un Sinon final qui sera exécuté si et seulement si aucune des conditions précédentes est vraie.

Algorithme 39 : Choisir un accessoire selon une saison obtenue auprès de l'utilisateur

```
1  écrire "Entrer la saison en cours :␣"
2  obtenir saison
3
4  accessoire ← ""                                  // chaîne vide (longueur zéro)
5  si saison = "été" alors
6  |  accessoire ← "parasol"
7  sinon si saison = "automne" alors
8  |  accessoire ← "chapeau"
9  sinon si saison = "hiver" alors
10 |  accessoire ← "foulard"
11 sinon si saison = "printemps" alors
12 |  accessoire ← "parapluie"
13 sinon
14 |  accessoire ← "erreur"
15
16 écrire "C'est l'" + saison + ", avez-vous besoin d'un␣" + accessoire + "?"
```

Revenons sur l'algorithme comparant deux nombres obtenus auprès de l'utilisateur. Nous pouvons maintenant traiter les trois états possibles de la condition :

Algorithme 40 : Comparer deux nombres (version 2)

```
1  obtenir a
2  obtenir b
3  si a < b alors
4  |  écrire a + "␣est plus petit que␣" + b
5  sinon si a > b alors
6  |  écrire a + "␣est plus grand que␣" + b
7  sinon
8  |  écrire a + "␣est égal à␣" + b
```

Voici un algorithme déterminant la côte d'un étudiant en utilisant sa note sur 100 :

Algorithme 41 : Déterminer la côte d'un étudiant

```
1  obtenir note
2  cote ← ""                                          // Chaîne vide
3  si note ≥ 90 alors
4  |  cote ← "A"
5  sinon si note ≥ 80 alors
6  |  cote ← "B"
7  sinon si note ≥ 70 alors
8  |  cote ← "C"
9  sinon si note ≥ 60 alors
10 |  cote ← "D"
11 sinon
12 |  cote ← "E"
13 écrire "La côte est␣" + cote
```

7.7 Exercices d'alternatives généralisées

1. Élaborer un algorithme déterminant le prix d'entrée au zoo selon l'âge de l'utilisateur. Le zoo est gratuit pour les enfants de 5 ans et moins, mais le prix d'entrée est de 20$ pour un adulte et de 10$ pour un enfant entre 6 et 17 ans.

2. Élaborer un algorithme déterminant le plus petit de trois nombres obtenus auprès de l'utilisateur. Cet exercice peut être résolu à l'aide d'une alternative généralisée utilisant des conditions composées de propositions booléennes ou encore par une hiérarchie imbriquée d'alternatives simples.

3. Élaborer un algorithme déterminant la couleur résultante de la combinaison de

deux couleurs parmi les trois suivantes : rouge, vert et bleu (RGB : Red Green Blue). Il est possible que l'utilisateur entre deux fois la même couleur et ce cas doit être géré. Par exemple : pour rouge et rouge, le programme produit la string "rouge". Pour bleu et vert, et pour vert et bleu, il produit la string "turquoise". Utiliser un logiciel de dessin sur ordinateur ou un site web pour explorer les mélanges RGB de rouge, vert et bleu.

Chapitre 8

Division entière et modulo

8.1 Notions

8.2 Exemples

est divisible, pair et impair, année bissextile (pré et post réforme)

8.3 Exercices

année bissextile pré et post réforme Faites un algorithme pour établir la somme des nombres impairs de 1 à 99 inclusivement et écrire cette somme.

Chapitre 9

Boucle

9.1 Notions

Plusieurs activités humaines présentent un caractère répétitif. La structure de contrôle qui permet de reproduire cette caractéristique est la boucle, aussi appelée répétitive ou répétition, que nous définissons ainsi :

La boucle est une structure permettant de répéter une ou plusieurs opérations (un bloc d'opérations) un certain nombre de fois, en fonction d'une condition. Tant que la condition est vraie, le bloc d'opérations s'exécutera. Comme pour l'alternative, la boucle utilise une expression booléenne pour la condition.

Les figures suivantes présentent les deux formes générales de la répétitive, l'une contenant une seule opération et l'autre en contenant plusieurs :

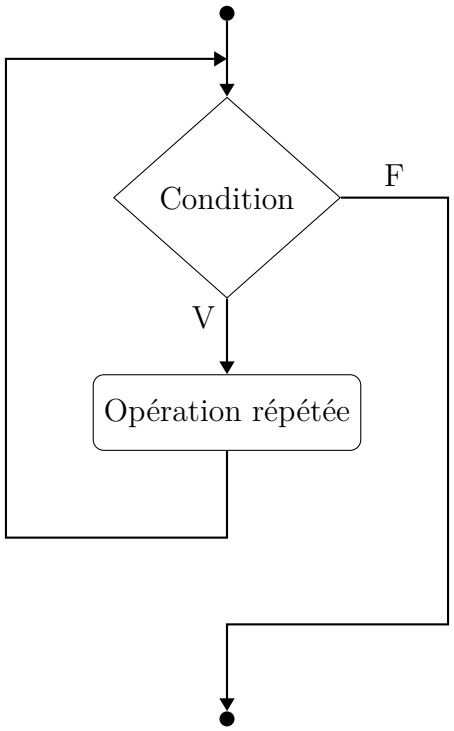


FIG. 9.1 : La forme générale d'une boucle répétant une seule opération

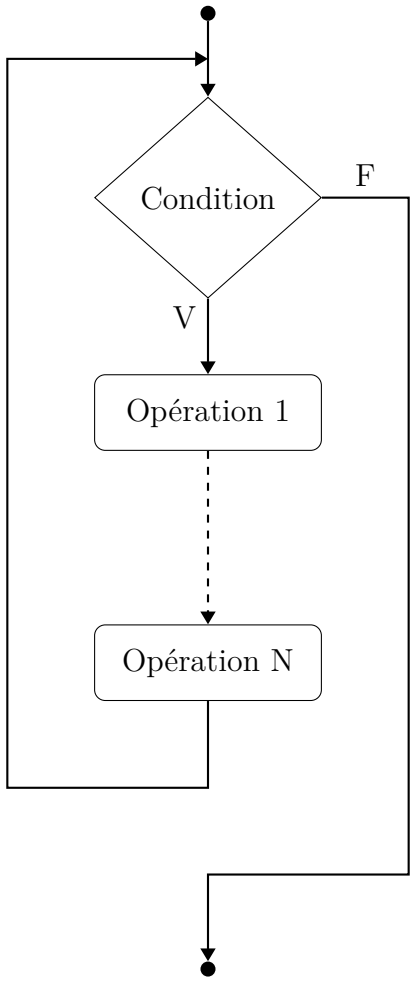


FIG. 9.2 : La forme générale d'une boucle répétant plusieurs opérations

Voici un exemple tiré de la vie courante : un professeur entreprend la correction de copies d'examen. La seule opération qu'il a à répéter est la correction d'une copie particulière. Cette opération doit être répétée tant et aussi longtemps qu'il reste au moins une copie à corriger, ce qui donne l'algorithme suivant :

Algorithme 42 : Corriger des copies d'examen (version 1)

```
1 tant que il reste au moins une copie à corriger faire
2 |  Corriger une copie
```

On peut reprendre ce même exemple et décomposer l'opération « Corriger une copie » en une séquence d'opérations plus élémentaires. Si l'examen comporte quatre questions, on peut décomposer la correction d'une copie de la façon suivante :

Algorithme 43 : Corriger une copie d'examen

```
1 Prendre une copie sur la pile de copies
2 Corriger la question 1
3 Corriger la question 2
4 Corriger la question 3
5 Corriger la question 4
6 Calculer le total
7 Inscrire le total en 1ère page
8 Ranger la copie sous la pile de copies
```

L'algorithme de correction des copies d'examen, lorsque détaillé, devient :

Algorithme 44 : Corriger des copies d'examen (version 2)

```
1 tant que il reste au moins une copie à corriger faire
2     Prendre une copie sur la pile de copies
3     Corriger la question 1
4     Corriger la question 2
5     Corriger la question 3
6     Corriger la question 4
7     Calculer le total
8     Inscrire le total en 1ère page
9     Ranger la copie sous la pile de copies
```

De par sa nature dynamique, il est important de comprendre le flot d'exécution de la boucle pour bien se l'approprier. Tant que la condition à évaluer représentée par le losange reste vraie, il y a exécution de l'opération représentée par le ou les rectangles. La ligne qui remonte à gauche indique qu'il faut retourner évaluer la condition. Ce mouvenement circulaire est la raison pour laquelle cette structure de contrôle est appelée « boucle » :

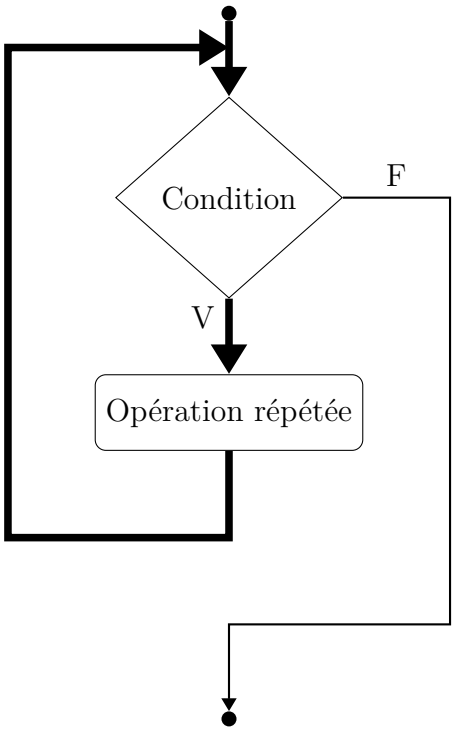


Fig. 9.3 : Exécution de la boucle lorsque la condition est vraie

Quand la condition devient fausse, on quitte la répétitive en empruntant la voie à droite du losange pour atteindre le point de sortie en bas :

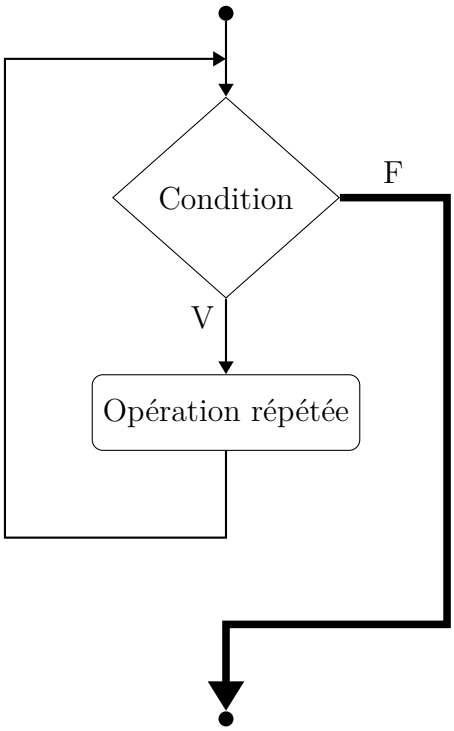


FIG. 9.4 : Exécution de la boucle lorsque la condition est fausse

Le pseudocode d'une boucle comporte une particularité non négligeable : les opérations à répéter apparaissent avec un décalage sur la droite qu'on appelle une indentation. Ceci permet de les distinguer des opérations qui suivent la répétitive. Dans l'exemple suivant, les opérations 1 à n sont décalées par rapport à l'énoncé tant que. Elles font partie de la répétitive, tandis que l'opération x, qui n'en fait pas partie, apparaît plus à gauche. On peut considérer les traits verticaux ajoutés au pseudocode comme les marques des niveaux d'indentation.

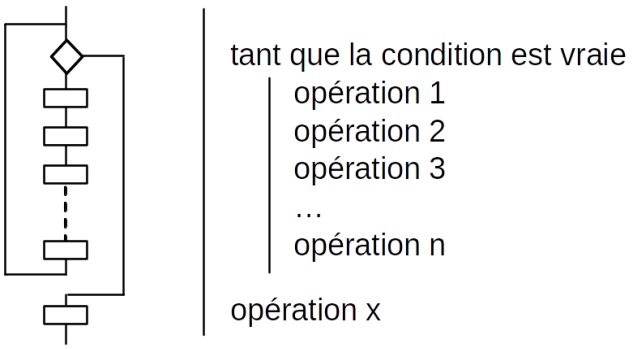


FIG. 9.5 : La boucle et son indentation

Dans l'exemple de la correction des copies d'examen, on imagine mal que les opérations ne soient pas exécutées au moins une fois, puisque cela impliquerait une classe sans élèves. Il ne faut pourtant pas croire que, dans une répétitive, les opérations sont nécessairement exécutées. On peut imaginer une situation où la condition est fausse au départ. Prenons l'exemple de l'ajustement de la pression d'air dans un pneu. La procédure normale consiste à mesurer la pression avec une jauge et, si la pression est insuffisante, à ajouter de l'air. Après quoi on vérifie à nouveau la pression et, éventuellement, on ajoute encore de l'air. Cette procédure peut se répéter un certain nombre de fois. En voici la formulation :

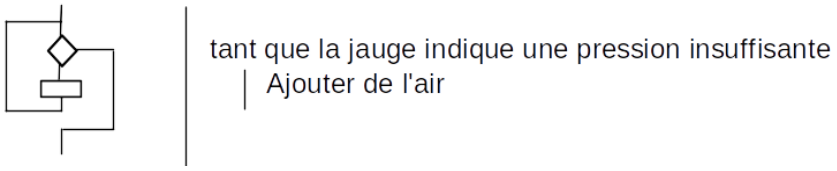


FIG. 9.6 : Ajustement de la pression d'un pneu

Si on applique cet algorithme à l'ajustement de la pression des pneus d'une voiture, il est fort probable que l'un d'eux aura dès le départ une pression suffisante. Ainsi l'opération prévue à l'intérieur de la répétitive n'aura pas à être exécutée dans ce cas-là.

9.2 Exemples de boucles

Algorithme 45 : Écrire les nombres de 1 à 10

```
1 i ← 1
2 tant que i <= 10 faire
3 |   écrire i
4 |   i ← i + 1
```

Algorithme 46 : La somme des nombres de 1 à 10

```
1 somme ← 0
2 i ← 1
3 tant que i <= 10 faire
4 |   somme ← somme + i
5 |   i ← i + 1
6 écrire somme
```

Algorithme 47 : Piger 10 nombres aléatoirement et indiquer s'ils sont pairs ou impairs.

```
1 i ← 1
2 tant que i <= 10 faire
3     nb ← piger un nombre entier dans [1, 100]
4     si nb modulo 2 = 0 alors
5         écrire nb + " est pair."
6     sinon
7         écrire nb + " est impair."
8     i ← i + 1
```

Algorithme d'Euclide permettant de calculer le plus grand commun diviseur entre deux nombres.

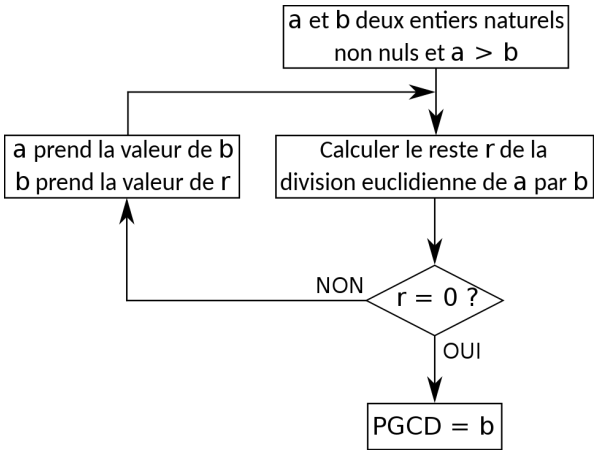


FIG. 9.7 : Algorigramme de l'algorithme d'Euclide

Algorithme 48 : Algorithme d'Euclide

```
1 obtenir a
2 obtenir b
3 écrire "Le plus grand commun diviseur de " + a + " et " + b + " :"
4 r ← a modulo b
5 tant que r != 0 faire
6     a ← b
7     b ← r
8     r ← a modulo b
9 écrire b
```

Compter les voyelles d'une phrase :

Algorithme 49 : Compter les voyelles dans une phrase

```
1 obtenir phrase
2 nbVoyelles ← 0
3 i ←0
4 tant que i < longueur de phrase faire
5 |   si le caractère phrase[i] est une voyelle alors
6 |   |   nbVoyelles ← nbVoyelles + 1
7 |   i ← i + 1
8 écrire "Cette phrase contient␣" + nbVoyelles + "␣voyelles"
```

9.3 Exercices de boucles

1. Élaborer un algorithme permettant de faire le décompte des nombres de 5 à -5 inclusivement.

2. Élaborer un algorithme écrivant successivement les nombres pairs de 2 à 50 inclusivement.

3. Élaborer un algorithme déterminant le prix d'entrée au zoo pour une famille. Le zoo est gratuit pour les enfants de 5 ans et mois, mais le prix d'entrée est de 20$ pour chaque adulte et de 10$ pour chaque enfant entre 6 et 17 ans. L'algorithme demandera le nombre de personnes dans la famille et, pour chacune d'elles, son âge. S'assurer du bon fonctionnement de l'algorithme à l'aide d'une table de traçage pour une famille de 4 personnes : deux adultes et deux enfants, un de 4 ans et un de 7 ans.

4. En vous basant sur l'algorithme de la somme de nombres, élaborer un algorithme calculant la factorielle d'un nombre obtenu auprès de l'utilisateur. En mathématiques, la factorielle d'un nombre entier « n » est le produit des nombres entiers strictement positifs inférieurs ou égaux à n. Cette opération est notée avec un point d'exclamation : « $n!$ ». Par exemple : $4! = 4 * 3 * 2 * 1 = 24$.

5. Élaborer un algorithme calculant la somme, le produit et la moyenne de N nombres obtenus auprès de l'utilisateur. Le nombre de nombres, N, doit lui aussi être obtenu auprès de l'utilisateur.

6. Élaborer un algorithme générant 12 tables de multiplication et, pour chaque table, lister les résultats pour les facteurs de multiplication de 1 à 10.

   Table de multiplication de 1

   1 x 1 = 1

   1 x 2 = 2

1 x 3 = 3

...

1 x 10 = 10

Table de multiplication de 2

2 x 1 = 2

2 x 2 = 4

2 x 3 = 6

...

2 x 10 = 20

...

Table de multiplication de 12

12 x 1 = 12

12 x 2 = 24

12 x 3 = 36

...

12 x 10 = 120

7. Élaborer un algorithme générant 10 séries de 5 nombres pseudo-aléatoires et, pour chaque série, afficher les nombres pigés ainsi que la somme et la moyenne de ces nombres.

8. Élaborer un algorithme générant 10 séries de 5 nombres pseudo-aléatoires et, pour chaque série, afficher les nombres pigés et indiquer le nombre de nombres pairs, impairs et de multiples de 10 dans la série.

Chapitre 10

Dormir

10.1 Notions

Il est possible de demander l'arrêt de l'exécution d'un algorithme pendant un certain temps. Cette pause dans l'exécution d'un algorithme est utile pour la création de programmes gérant le temps, par exemple un chronomètre, une minuterie ou une horloge. C'est avec l'instruction « **dormir** » que l'on suspend l'exécution. On doit indiquer combien de temps dormir ; cette valeur est mesurée en millisecondes (1/1000 de secondes).

Algorithme 50 : Dormir

```
1 écrire "Écrire cette ligne de texte."
2 dormir 1000                          // 1000 millisecondes est 1 seconde
3 écrire "Écrire cette autre ligne de texte une seconde plus tard."
```

Lors de l'exécution de l'algorithme sur un ordinateur, la première ligne de texte apparaîtra à l'écran et il s'écoulera *approximativement* une seconde avant que la deuxième ligne de texte se manifeste à l'écran.

10.2 Exemples

Une minuterie débutant à un nombre de minutes et de secondes spécifiées par l'utilisateur :

Algorithme 51 : Minuterie

```
1 obtenir m                                                    // Minutes
2 obtenir s                                                    // Secondes
3 tant que m > 0 OU s > 0 faire
4     écrire m + " :" + s
5     dormir 1000
6     s ← s − 1
7     si s = −1 alors
8         s ← 59
9         m ← m − 1
```

10.3 Exercices

1. En se basant sur l'algorithme de la minuterie, élaborer un algorithme simulant une horloge, en mode 24h, débutant à l'heure $(h : m : s)$ spécifiée par l'utilisateur et s'arrêtant à minuit pile.

Chapitre 11

Itération

11.1 Notions d'itération

11.2 Patron d'accumulation

11.2.1 Notions

11.2.2 Exemples

addition des nombres entre [A,B], Addition des impairs entre [A,B]

11.2.3 Exercices

Faites un algorithme pour calculer la factorielle d'un nombre. Par exemple, le calcul de la factorielle de 5 est :

5 ! = 5 * 4 * 3 * 2 * 1 = 120

Voici un exemple du résultat attendu pour le calcul de la factorielle de 5 : 120

11.3 Patron de vérification

11.3.1 Notions

11.3.2 Exemples

si un multiple d'un nombre N existe entre [A, B]

11.3.3 Exercices

Deuxième partie

Modularité

Une des difficultés de l'algorithmique est que, pour un problème complexe, le nombre d'étapes à expliciter peut être très grand. Ce nombre considérable d'étapes empêche l'écriture spontanée d'un algorithme. Puisque toutes les étapes d'un algorithme ne peuvent pas être écrites d'un seul coup, on doit plutôt décomposer le problème complexe initial en sous-problèmes plus simples. Chaque sous-problème est ensuite analysé et une évaluation de sa complexité est faite. Si ce sous-problème est « trop grand » (chacun juge de cela par lui-même d'après ses capacités ou contraintes), on pourra décomposer ce sous-problème en des sous-sous-problèmes et ainsi de suite jusqu'à ce qu'on ait plusieurs petits problèmes de taille « acceptable » à régler au lieu d'un seul problème d'une grande complexité. Cette approche de détermination d'une séquence logique des opérations est ce qu'on peut appeler la décomposition successive des opérations.

On peut décomposer aussi finement que nécessaire chaque opération en sous- opérations. Cette tâche est plutôt abstraite et demande un travail cognitif important.

Chapitre 12

Développement graduel par décomposition

Élaborer un algorithme n'est pas toujours une tâche simple. Le succès repose en grande partie sur le type de démarche suivie, surtout quand le problème est complexe. La méthode la plus valable consiste à procéder étape par étape, en allant du général au particulier : c'est ce qu'on appelle le développement graduel (aussi appelé le raffinement graduel).

12.1 Notions

En principe, tout le monde est d'accord avec cette démarche. On trouve normal, par exemple, que l'architecte s'intéresse d'abord, dans ses plans, à l'allure générale d'une maison et à la division des pièces, avant de régler des questions comme la plomberie ou l'électricité.

Le développement graduel est une notion de base à laquelle nous accordons une grande importance puisqu'il s'agit d'une façon de mettre en place une démarche d'analyse de l'activité de programmation. D'une manière générale, cette démarche comporte autant d'étapes que nécessaire pour que l'algorithme décrive les opérations de façon suffisamment claire.

Pour illustrer la démarche par développement graduel, nous allons utiliser l'exemple de l'ajustement de la pression d'air dans un pneu. Supposons que la pression peut, au départ, être trop haute ou trop basse.

Étape 1 : On décrit d'abord l'activité dans ses grandes lignes.

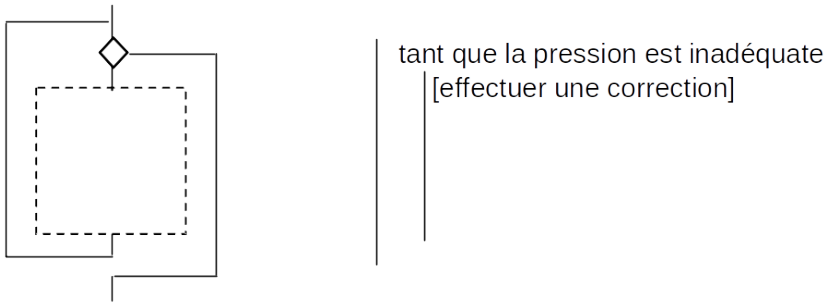


FIG. 12.1 : Étape 1 d'un exemple de développement graduel tiré de la vie courante

Notons que la répétitive s'impose d'emblée, car il est probable qu'il faudra effectuer plusieurs corrections avant d'atteindre la pression souhaitée.

L'une des difficultés du développement graduel est précisément de trouver une formulation générale satisfaisante pour décrire les opérations qui devront être décomposées dans une étape ultérieure. Dans cet exemple, la formulation effectuer une correction convient tout aussi bien à l'addition qu'à la soustraction d'une certaine quantité d'air.

Le petit rectangle habituellement utilisé pour représenter une opération dans les algorigrammes est ici remplacé par un grand rectangle pointillé pour indiquer que cette opération est décomposable.

Étape 2 : On décompose ensuite l'opération [Effectuer une correction], qui était alors formulée de façon générale.

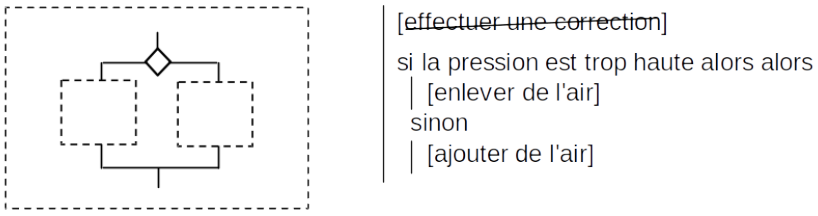
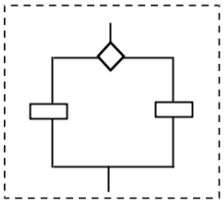


FIG. 12.2 : Étape 2 d'un exemple de développement graduel tiré de la vie courante

Étape 3 : Il est possible de pousser plus loin l'analyse et de développer davantage cet algorithme en décomposant les deux opérations [Enlever de l'air] et [Ajouter de l'air]. En effet, selon l'importance de l'écart indiqué par la jauge par rapport à la pression désirée, on pourra enlever ou ajouter une quantité d'air plus ou moins grande.

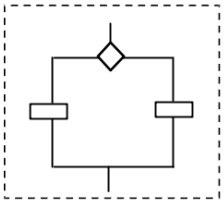
Ainsi l'opération [Enlever de l'air] devient :



[enlever de l'air]
si l'écart est grand alors
| Enlever beaucoup d'air
sinon
| Enlever un peu d'air

FIG. 12.3 : Étape 3a d'un exemple de développement graduel tiré de la vie courante

Et l'opération [Ajouter de l'air] prend la forme suivante :

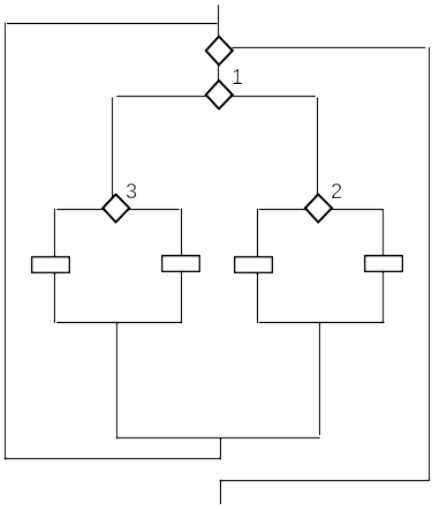


[ajouter de l'air]
si l'écart est grand alors
| Ajouter beaucoup d'air
sinon
| Ajouter un peu d'air

FIG. 12.4 : Étape 3b d'un exemple de développement graduel tiré de la vie courante

Quand on développe des algorithmes dans un contexte de la vie quotidienne, il n'est pas facile de décider où le développement doit s'arrêter. Dans le cas présent, en s'arrêtant maintenant, on obtient un algorithme tout à fait satisfaisant qu'on peut voir au complet dans la prochaine figure. Dans le cas de problèmes de programmation par contre, la limite est plus facile à poser puisqu'on ne peut aller au-delà des instructions de base que l'ordinateur exécute.

Comme nous venons de le voir, le développement graduel permet de construire un algorithme en procédant par étapes et en n'utilisant que les trois outils de base de la programmation structurée : la séquence, la répétitive et l'alternative. L'une des caractéristiques fondamentales de cette démarche, c'est qu'à chaque nouvelle étape, seules les opérations peuvent être décomposées et transformées. L'algorithme se développe donc toujours de l'intérieur, sans qu'il y ait jamais de remise en cause de la structure déjà élaborée. Le manque de souplesse apparent qui en résulte est toutefois compensé par la fiabilité du produit fini et on peut, de toute façon, atténuer ce manque par l'importance qu'on aura accordée dès le départ à l'analyse du problème.




```
tant que la pression est inadéquate
1 | si la pression est trop haute alors
  2 | si l'écart est grand alors
    |   | Enlever beaucoup d'air
    | sinon
    |   | Enlever un peu d'air
  sinon
  3 | si l'écart est grand alors
    |   | Ajouter beaucoup d'air
    | sinon
    |   | Ajouter un peu d'air
```

FIG. 12.5 : Algorithme monolithique d'un exemple de développement graduel tiré de la vie courante

12.2 Exemples

Prenons un programme que nous avons déjà rencontré : la génération des tables de multiplication de 1 à 12 pour les 10 premiers facteurs de chaque table. Ainsi, on veut que le programme produise les tables suivantes :

Table de multiplication de 1

1 x 1 = 1

1 x 2 = 2

1 x 3 = 3

...

1 x 10 = 10

Table de multiplication de 2

2 x 1 = 2

2 x 2 = 4

2 x 3 = 6

...

2 x 10 = 20

...

Table de multiplication de 12

12 x 1 = 12

12 x 2 = 24

12 x 3 = 36

...

12 x 10 = 120

Voici la décomposition par développement graduel de la solution au problème posé :

Algorithme 52 : Développement graduel du programme générant les tables de multiplication

```
1  table ← 1
2  tant que table <= 12 faire
3  |   [ Générer une table ]
4  |   table ← table + 1
5
6  [ Générer une table ] :
7  facteur ← 1
8  tant que facteur <= 10 faire
9  |   [ Générer une ligne de la table ]
10 |   facteur ← facteur + 1
11
12 [ Générer une ligne de la table ] :
13 resultat ← table * facteur
14 écrire table, "×", facteur, "=", resultat
```

Cet algorithme est décomposé en 3 niveaux :

1. Niveau 0 : la mise en place d'une boucle (initialisation à la ligne 1, condition à la ligne 2 et incrémentation à la ligne 4) pour chaque table.
2. Niveau 1 : Le code à exécuter pour chaque table : la mise en place d'une boucle pour les facteurs de multiplication (initialisation à la ligne 7, condition à la ligne 8 et incrémentation à la ligne 10).
3. Niveau 2 : Pour chacun des facteurs de chacune des tables, on voudra générer la ligne en calculant le résultat de la multiplication et en écrivant à l'écran la ligne.

L'avantage de l'approche par développement graduel est que chaque bloc de code (chaque niveau) peut être conçu et lu sans devoir se référer explicitement aux autres

blocs. Il y a donc une certaine isolation entre les différents niveaux. Cette isolation n'est pas complète puisque, par exemple, pour comprendre pleinement la bloc [Générer une ligne de la table], il faut comprendre quel rôle joue les variables table et facteur. Éventuellement, l'utilisation des routines permettra une indépendance complète des blocs d'opérations.

12.3 Exercices

1. À l'aide du développement graduel, élaborer un algorithme permettant à un nouvel employé de calculer quel sera son salaire annuel dans un nombre d'années spécifié à partir de son salaire annuel de départ et de sa classe d'employé sachant que :

   - Il existe deux classes d'employés : syndiqué et cadre.
   - Tous les employés reçoivent au moins 2% d'augmentation salariale par année.
   - Les syndiqués reçoivent une augmentation supplémentaire de 0,5% par année tandis que les cadres reçoivent 0,7% d'augmentation supplémentaire par année.
   - À chaque 10 années de service, un cadre reçoit une augmentation supplémentaire ponctuelle de 5%.
   - Pour la 5ième et la 10ième année de service, un syndiqué reçoit une augmentation ponctuelle de 7%.

Chapitre 13

Routines

Page 18 des notes de cours Word : Modularité

Chapitre 14

Procédures

Chapitre 15

Paramètres par valeur

Chapitre 16

Fonctions

16.1 Exercices

1. Élaborer un programme convertissant une vitesse obtenue auprès de l'utilisateur en mètres/seconde vers pieds/seconde. Au minimum, votre programme contiendra une fonction appelée *ConvertirMètresVersPieds*.

2. Produire les tables de traçage pour démontrer que le programme précédemment créé fonctionne correctement quand l'utilisateur entre 2,5 mètres par seconde.

3. En réutilisant la fonction *ConvertirMètresVersPieds*, élaborer un programme produisant à l'écran une table de conversion de mètres par seconde vers pieds par seconde pour les entiers de 0 à 10.

4. Produire les tables de traçage démontrant que votre algorithme fonctionne pour les trois premières lignes de la table de conversion du numéro précédent.

5. Élaborer un programme permettant de calculer l'aire d'une forme géométrique. L'utilisateur doit fournir le nom de la forme et les valeurs nécessaires au calcul. Faites des fonctions pour calculer l'aire d'un cercle, l'aire d'un triangle et l'aire d'un rectangle. Au minimum, votre programme contiendra 4 routines : la procédure principale et 3 fonctions.

6. Procéder au traçage du code élaboré précédemment pour un triangle de largeur 10 et de hauteur 8.

Chapitre 17

Diagrammes hiérarchiques

Troisième partie

La programmation des algorithmes

Chapitre 18

La validation

Chapitre 19

Les fichiers