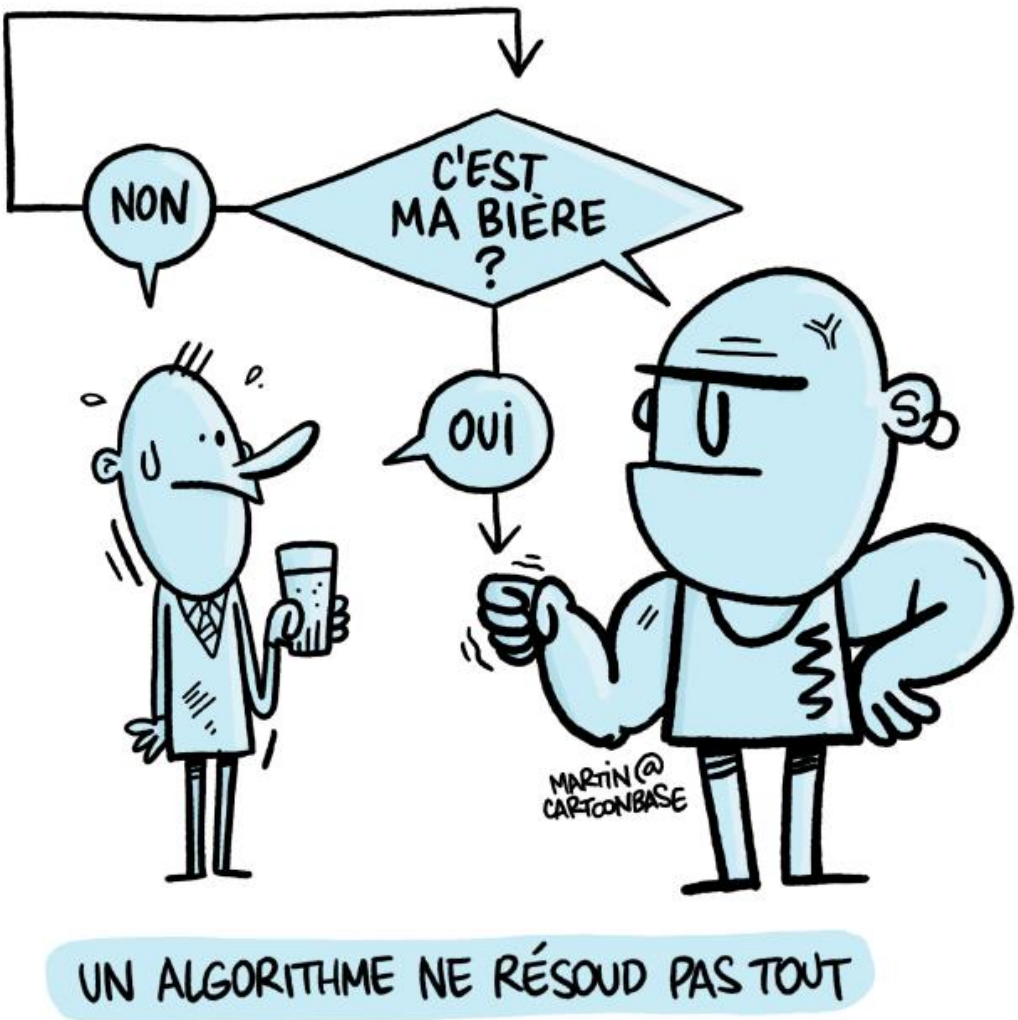
16/10/2020

Algorithme



Lionel CICALA

VERSION 1.0

Table des matières

1 But du document

Ce document a pour objectif de vous donner les bonnes pratiques en terme d'analyse et de compréhension d'un besoin et de former correctement votre réflexion hors langage de programmation.

2 Introduction à l'Algorithmique.

2.1 L'Algorithmique c'est quoi ?

L'algorithmique est un terme d'origine arabe, mot dérivé du nom du mathématicien « Al-Khawarizmi » qui a vécu au IX ème siècle, membre de l'académie des sciences à Bagdad.

Avez-vous déjà ouvert un livre de recettes de cuisine ?
Avez-vous déjà déchiffré un mode d'emploi traduit directement du coréen pour faire fonctionner un magnétoscope ou un répondeur téléphonique réticent ?
Si oui, sans le savoir, vous avez déjà exécuté des algorithmes.

Plus fort : avez-vous déjà indiqué un chemin à un touriste égaré ?
Avez-vous fait chercher un objet à quelqu'un par téléphone ?
Ecrit une lettre anonyme stipulant comment procéder à une remise de rançon ?
Si oui, vous avez déjà fabriqué – et fait exécuter – des algorithmes.

Comme quoi, l'algorithmique n'est pas un savoir ésotérique réservé à quelques rares initiés touchés par la grâce divine, mais une aptitude partagée par la totalité de l'humanité. Donc, pas d'excuses…

Un algorithme, c'est une suite d'instructions, qui une fois exécutée correctement, conduit à un résultat donné.

Si l'algorithme est juste, le résultat est le résultat voulu, et le touriste se retrouve là où il voulait aller.
Si l'algorithme est faux, le résultat est, disons, aléatoire, et décidément, ce satané répondeur ne veut rien savoir.

Complétons toutefois cette définition. Après tout, en effet, si l'algorithme, comme on vient de le dire, n'est qu'une suite d'instructions menant celui qui l'exécute à résoudre un problème, pourquoi ne pas donner comme instruction unique : « résous le problème », et laisser l'interlocuteur se débrouiller avec ça ?
A ce tarif, n'importe qui serait champion d'algorithmique sans faire aucun effort. Pas de ça, ce serait trop facile.

Le malheur (ou le bonheur, tout dépend du point de vue) est que justement, si le touriste vous demande son chemin, c'est qu'il ne le connaît pas. Donc, si on n'est

pas un goujat intégral, il ne sert à rien de lui dire de le trouver tout seul. De même les modes d'emploi contiennent généralement (mais pas toujours) un peu plus d'informations que « débrouillez-vous pour que ça marche ».

Pour fonctionner, un algorithme doit donc contenir uniquement des instructions compréhensibles par celui qui devra l'exécuter.

C'est d'ailleurs l'un des points délicats pour les rédacteurs de modes d'emploi : les références culturelles, ou lexicales, des utilisateurs, étant variables, un même mode d'emploi peut être très clair pour certains et parfaitement abscons pour d'autres.

En informatique, heureusement, il n'y a pas ce problème : les choses auxquelles nous devons donner des instructions sont des ordinateurs, et ceux-ci ont le bon goût de faire strictement ce que nous leur demandons.

2.2 Que faut-il pour être bon en Algorithmique ?

La maîtrise de l'algorithmique requiert deux qualités, très complémentaires :

- il faut avoir une certaine intuition, car aucune recette ne permet de savoir a priori quelles instructions permettront d'obtenir le résultat voulu. C'est là, si l'on y tient, qu'intervient la forme « d'intelligence » requise pour l'algorithmique. Alors, c'est certain, il y a des gens qui possèdent au départ davantage cette intuition que les autres. Cependant, et j'insiste sur ce point, les réflexes, cela s'acquiert. Et ce qu'on appelle l'intuition n'est finalement que de l'expérience tellement répétée que le raisonnement, au départ laborieux, finit par devenir « spontané ».
- il faut être méthodique et rigoureux. En effet, chaque fois qu'on écrit une série d'instructions qu'on croit justes, il faut systématiquement se mettre mentalement à la place de la machine qui va les exécuter, armé d'un papier et d'un crayon, afin de vérifier si le résultat obtenu est bien celui que l'on voulait. Cette opération ne requiert pas la moindre once d'intelligence. Mais elle reste néanmoins indispensable, si l'on ne veut pas écrire à l'aveuglette.

Et petit à petit, à force de pratique, vous verrez que vous pourrez faire de plus en plus souvent l'économie de cette dernière étape : l'expérience fera que vous « verrez » le résultat produit par vos instructions, au fur et à mesure que vous les écrirez. Naturellement, cet apprentissage est long, et demande des heures de travail patient. Aussi, dans un premier temps, évitez de sauter les étapes :

La vérification méthodique, pas à pas, de chacun de vos algorithmes représente plus de la moitié du travail à accomplir... et le gage de vos progrès.

2.3 Quels sont les éléments qui font l'Algorithmique ?

Les ordinateurs, quels qu'ils soient, ne sont fondamentalement capables de comprendre que quatre catégories d'ordres (en programmation, on n'emploiera pas le terme d'ordre, mais plutôt celui d'instructions). Ces quatre familles d'instructions sont :

- l'affectation de variables
- la lecture / écriture
- les tests
- les boucles

Un algorithme informatique se ramène donc toujours au bout du compte à la combinaison de ces quatre petites briques de base. Il peut y en avoir quelques-unes, quelques dizaines, et jusqu'à plusieurs centaines de milliers dans certains programmes de gestion. Rassurez-vous, dans le cadre de ce cours, nous n'irons pas jusque-là (cependant, la taille d'un algorithme ne conditionne pas en soi sa complexité : de longs algorithmes peuvent être finalement assez simples, et de petits très compliqués).

2.4 Algorithmique et programmation.

Pourquoi apprendre l'algorithmique pour apprendre à programmer ? En quoi a-t-on besoin d'un langage spécial, distinct des langages de programmation compréhensibles par les ordinateurs ?

Parce que l'algorithmique exprime les instructions résolvant un problème donné indépendamment des particularités de tel ou tel langage.

Pour prendre une image, si un programme était une dissertation, l'algorithmique serait le plan, une fois mis de côté la rédaction et l'orthographe.

Or, vous savez qu'il vaut mieux faire d'abord le plan et rédiger ensuite que l'inverse…

Apprendre l'algorithmique, c'est apprendre à manier la structure logique d'un programme informatique. Cette dimension est présente quelle que soit le langage de programmation ; mais lorsqu'on programme dans un langage (en C, en Visual Basic, etc.) on doit en plus se colleter les problèmes de syntaxe, ou de types d'instructions, propres à ce langage.

Apprendre l'algorithmique de manière séparée, c'est donc cibler les difficultés pour mieux les vaincre.

A cela, il faut ajouter que des générations de programmeurs, souvent autodidactes (mais pas toujours, hélas !), ayant directement appris à programmer dans tel ou tel langage, ne font pas mentalement clairement la différence entre ce qui relève de la structure logique générale de toute programmation (les règles fondamentales de l'algorithmique) et ce qui relève du langage particulier qu'ils ont appris. Ces programmeurs, non seulement ont beaucoup plus de mal à passer ensuite à un langage différent, mais encore écrivent bien souvent des programmes qui même s'ils

sont justes, restent laborieux. Car on n'ignore pas impunément les règles fondamentales de l'algorithmique… Alors, autant l'apprendre en tant que telle !

2.5 Avec quelles conventions écrit-on un algorithme ?

Historiquement, plusieurs types de notations ont représenté des algorithmes.

Il y a eu notamment une représentation graphique, avec des carrés, des losanges, etc. qu'on appelait des organigrammes.

Aujourd'hui, cette représentation est quasiment abandonnée, pour deux raisons.

D'abord, parce que dès que l'algorithme commence à grossir un peu, ce n'est plus pratique du tout du tout.

Ensuite parce que cette représentation favorise le glissement vers un certain type de programmation, dite non structurée (nous définirons ce terme plus tard), que l'on tente au contraire d'éviter.

C'est pourquoi on utilise généralement une série de conventions appelée « pseudo-code », qui ressemble à un langage de programmation authentique dont on aurait évacué la plupart des problèmes de syntaxe.

Ce pseudo-code est susceptible de varier légèrement d'un livre (ou d'un enseignant) à un autre.

C'est bien normal : le pseudo-code, encore une fois, est purement conventionnel ; aucune machine n'est censée le reconnaître.

Donc, chaque cuisinier peut faire sa sauce à sa guise, avec ses petites épices bien à lui, sans que cela prête à conséquence.

3 Les variables

3.1 A quoi servent les variables ?

Dans un programme informatique, on va avoir en permanence besoin de stocker provisoirement des valeurs.

Il peut s'agir de données issues du disque dur, fournies par l'utilisateur (frappées au clavier), ou que sais-je encore.

Il peut aussi s'agir de résultats obtenus par le programme, intermédiaires ou définitifs.

Ces données peuvent être de plusieurs types : elles peuvent être des nombres, du texte, etc.

Toujours est-il que dès que l'on a besoin de stocker une information au cours d'un programme, on utilise une variable.

Pour employer une image, une variable est une boîte, que le programme (l'ordinateur) va repérer par une étiquette. Pour avoir accès au contenu de la boîte, il suffit de la désigner par son étiquette.

Dans l'ordinateur, physiquement, il y a un emplacement de mémoire, repéré par une adresse binaire.

Si on programmait dans un langage directement compréhensible par la machine, on devrait se fader de désigner nos données par de superbes 10011001 et autres 01001001.

Mauvaise nouvelle : de tels langages existent ! Ils portent le doux nom d'assembleur. Bonne nouvelle : ce ne sont pas les seuls langages disponibles.

Les langages informatiques plus évolués (ce sont ceux que presque tout le monde emploie) se chargent précisément, entre autres rôles, d'épargner au programmeur la gestion fastidieuse des emplacements mémoire et de leurs adresses.

Et, comme vous commencez à le comprendre, il est beaucoup plus facile d'employer les étiquettes de son choix, que de devoir manier des adresses binaires.

3.2 Déclaration des variables

La première chose à faire avant de pouvoir utiliser une variable est de créer la boîte et de lui coller une étiquette.

Ceci se fait tout au début de l'algorithme, avant même les instructions proprement dites.

C'est ce qu'on appelle la déclaration des variables.

Le nom de la variable (l'étiquette de la boîte) obéit à des impératifs changeant selon les langages.

Toutefois, une règle absolue est qu'un nom de variable peut comporter des lettres et des chiffres, mais qu'il exclut la plupart des signes de ponctuation, en particulier les espaces.

Un nom de variable correct commence également impérativement par une lettre. Quant au nombre maximal de signes pour un nom de variable, il dépend du langage utilisé.

En pseudo-code algorithmique, on est bien sûr libre du nombre de signes pour un nom de variable, même si pour des raisons purement pratiques, on évite généralement les noms à rallonge.

Lorsqu'on déclare une variable, il ne suffit pas de créer une boîte (réserver un emplacement mémoire) ; encore doit-on préciser ce que l'on voudra mettre dedans, car de cela dépendent la taille de la boîte (de l'emplacement mémoire) et le type de codage utilisé.

3.2.1 Types numériques classiques

Commençons par le cas très fréquent, celui d'une variable destinée à recevoir des nombres.

Si l'on réserve un octet pour coder un nombre, on ne pourra coder que $2^8 = 256$ valeurs différentes.

Cela peut signifier par exemple les nombres entiers de 1 à 256, ou de 0 à 255, ou de –127 à +128…

Si l'on réserve deux octets, on a droit à 65 536 valeurs ; avec trois octets, 16 777 216, etc.

Et là se pose un autre problème : ce codage doit-il représenter des nombres décimaux ? Des nombres négatifs ?

Bref, le type de codage (autrement dit, le type de variable) choisi pour un nombre va déterminer :

- les valeurs maximales et minimales des nombres pouvant être stockés dans la variable
- la précision de ces nombres (dans le cas de nombres décimaux).

Tous les langages, quels qu'ils soient offrent un « bouquet » de types numériques, dont le détail est susceptible de varier légèrement d'un langage à l'autre.

On retrouve cependant les types suivants :

Type Numérique	Plage
Byte (octet)	0 à 255
Entier simple	-32 768 à 32 767
Entier long	-2 147 483 648 à 2 147 483 647
Réel simple	$-3{,}40\text{x}10^{38}$ à $-1{,}40\text{x}10^{45}$ pour les valeurs négatives $1{,}40\text{x}10^{-45}$ à $3{,}40\text{x}10^{38}$ pour les valeurs positives
Réel double	$1{,}79\text{x}10^{308}$ à $-4{,}94\text{x}10^{-324}$ pour les valeurs négatives $4{,}94\text{x}10^{-324}$ à $1{,}79\text{x}10^{308}$ pour les valeurs positives

Pourquoi ne pas déclarer toutes les variables numériques en réel double, histoire de bétonner et d'être certain qu'il n'y aura pas de problème ?

En vertu du principe de l'économie de moyens. Un bon algorithme ne se contente pas de « marcher » ; il marche en évitant de gaspiller les ressources de la machine.

Sur certains programmes de grande taille, l'abus de variables surdimensionnées peut entraîner des ralentissements notables à l'exécution, voire un plantage pur et simple de l'ordinateur.

Alors, autant prendre dès le début de bonnes habitudes d'hygiène.

En algorithmique, on ne se tracassera pas trop avec les sous-types de variables numériques (sachant qu'on aura toujours assez de soucis comme ça).

On se contentera donc de préciser qu'il s'agit d'un nombre, en gardant en tête que dans un vrai langage, il faudra être plus précis.

En pseudo-code, une déclaration de variables aura ainsi cette tête :

```
Variable g en Numérique
ou encore
Variables PrixHT, TauxTVA, PrixTTC en Numérique
```

3.2.2 Autres types numériques

Certains langages autorisent d'autres types numériques, notamment :

- le type monétaire (avec strictement deux chiffres après la virgule)
- le type date (jour/mois/année).

Nous n'emploierons pas ces types dans ce cours ; mais je les signale, car vous ne manquerez pas de les rencontrer en programmation proprement dite.

3.2.3 Type alphanumérique

Fort heureusement, les boîtes que sont les variables peuvent contenir bien d'autres informations que des nombres.

Sans cela, on serait un peu embêté dès que l'on devrait stocker un nom de famille, par exemple.

On dispose donc également du type alphanumérique (également appelé type caractère, type chaîne ou en anglais, le type string.

Dans une variable de ce type, on stocke des caractères, qu'il s'agisse de lettres, de signes de ponctuation, d'espaces, ou même de chiffres.

Le nombre maximal de caractères pouvant être stockés dans une seule variable string dépend du langage utilisé.

Un groupe de caractères (y compris un groupe de un, ou de zéro caractères), qu'il soit ou non stocké dans une variable, d'ailleurs, est donc souvent appelé chaîne de caractères.

En pseudo-code, une chaîne de caractères est toujours notée entre guillemets « »

Pourquoi diable ?

Pour éviter deux sources principales de possibles confusions :

- la confusion entre des nombres et des suites de chiffres. Par exemple, 423 peut représenter le nombre 423 (quatre cent vingt-trois), ou la suite de caractères 4, 2, et 3. Et ce n'est pas du tout la même chose !

  Avec le premier, on peut faire des calculs, avec le second, point du tout.

Dès lors, les guillemets permettent d'éviter toute ambiguïté : s'il n'y en a pas, 423 est quatre cent vingt-trois. S'il y en a, "423" représente la suite des chiffres 4, 2, 3.

- L'autre confusion, bien plus grave - et bien plus fréquente – consiste à se mélanger les pinceaux entre le nom d'une variable et son contenu.

Pour parler simplement, cela consiste à confondre l'étiquette d'une boîte et ce qu'il y a à l'intérieur…

3.2.4 Type booléen

Le dernier type de variables est le type booléen : on y stocke uniquement les valeurs logiques VRAI et FAUX.

On peut représenter ces notions abstraites de VRAI et de FAUX par tout ce qu'on veut : de l'anglais (TRUE et FALSE) ou des nombres (0 et 1).

Ce qui compte, c'est de comprendre que le type booléen est très économique en termes de place mémoire occupée, puisque pour stocker une telle information binaire, un seul bit suffit.

Le type booléen est très souvent négligé par les programmeurs, à tort.
Il est vrai qu'il n'est pas à proprement parler indispensable, et qu'on pourrait écrire à peu près n'importe quel programme en l'ignorant complètement.

Pourtant, si le type booléen est mis à disposition des programmeurs dans tous les langages, ce n'est pas pour rien.

Le recours aux variables booléennes s'avère très souvent un puissant instrument de lisibilité des algorithmes : il peut faciliter la vie de celui qui écrit l'algorithme, comme de celui qui le relit pour le corriger.

Alors, maintenant, c'est certain, en algorithmique, il y a une question de style : c'est exactement comme dans le langage courant, il y a plusieurs manières de s'exprimer pour dire sur le fond la même chose.

Nous verrons plus loin différents exemples de variations stylistiques autour d'une même solution.

En attendant, vous êtes prévenus : l'auteur de ce cours est un adepte fervent (mais pas irraisonné) de l'utilisation des variables booléennes.

3.3 L'instruction d'affectation

3.3.1 Syntaxe et signification

La seule chose qu'on puisse faire avec une variable, c'est l'affecter, c'est-à-dire lui attribuer une valeur.

En pseudo-code, l'instruction d'affectation se note avec le signe ←

Ainsi :

```
Toto ← 24
```

Attribue la valeur 24 à la variable Toto.

Ceci, soit dit en passant, sous-entend impérativement que Toto soit une variable de type numérique.
Si Toto a été défini dans un autre type, il faut bien comprendre que cette instruction provoquera une erreur.
C'est un peu comme si, en donnant un ordre à quelqu'un, on accolait un verbe et un complément incompatibles, du genre « Epluchez la casserole ».
Même dotée de la meilleure volonté du monde, la ménagère lisant cette phrase ne pourrait qu'interrompre dubitativement sa tâche.
Alors, un ordinateur, vous pensez bien…

On peut en revanche sans aucun problème attribuer à une variable la valeur d'une autre variable, telle quelle ou modifiée.

Par exemple :

```
Tutu ← Toto
```

Signifie que la valeur de Tutu est maintenant celle de Toto.
Notez bien que cette instruction n'a en rien modifié la valeur de Toto :

Une instruction d'affectation ne modifie que ce qui est situé à gauche de la flèche.

```
Tutu ← Toto + 4
```

Si Toto contenait 12, Tutu vaut maintenant 16.
De même que précédemment, Toto vaut toujours 12.

```
Tutu ← Tutu + 1
```

Si Tutu valait 6, il vaut maintenant 7. La valeur de Tutu est modifiée, puisque Tutu est la variable située à gauche de la flèche.

Pour revenir à présent sur le rôle des guillemets dans les chaînes de caractères et sur la confusion numéro 2 signalée plus haut, comparons maintenant deux algorithmes suivants :

```
Exemple n°1
Début
Riri ← "Loulou"
Fifi ← "Riri"
Fin

Exemple n°2
Début
Riri ← "Loulou"
Fifi ← Riri
Fin
```

La seule différence entre les deux algorithmes consiste dans la présence ou dans l'absence des guillemets lors de la seconde affectation.
Et l'on voit que cela change tout !

Dans l'exemple n°1, ce que l'on affecte à la variable Fifi, c'est la suite de caractères R – i – r - i. Et à la fin de l'algorithme, le contenu de la variable Fifi est donc « Riri ».

Dans l'exemple n°2, en revanche, Riri étant dépourvu de guillemets, n'est pas considéré comme une suite de caractères, mais comme un nom de variable.
Le sens de la ligne devient donc : « affecte à la variable Fifi le contenu de la variable Riri ». A la fin de l'algorithme n°2, la valeur de la variable Fifi est donc « Loulou ».

Ici, l'oubli des guillemets conduit certes à un résultat, mais à un résultat différent.

A noter, car c'est un cas très fréquent, que généralement, lorsqu'on oublie les guillemets lors d'une affectation de chaîne, ce qui se trouve à droite du signe d'affectation ne correspond à aucune variable précédemment déclarée et affectée.

Dans ce cas, l'oubli des guillemets se solde immédiatement par une erreur d'exécution.

Ceci est une simple illustration. Mais elle résume l'ensemble des problèmes qui surviennent lorsqu'on oublie la règle des guillemets aux chaînes de caractères.

3.3.2 Ordre des instructions

Il va de soi que l'ordre dans lequel les instructions sont écrites va jouer un rôle essentiel dans le résultat final.

Considérons les deux algorithmes suivants :

```
Exemple 1
Variable A en Numérique
Début
A ← 34
A ← 12
Fin

Exemple 2
Variable A en Numérique
Début
A ← 12
A ← 34
Fin
```

Il est clair que dans le premier cas la valeur finale de A est 12, dans l'autre elle est 34.

Il est tout aussi clair que ceci ne doit pas nous étonner. Lorsqu'on indique le chemin à quelqu'un, dire « prenez tout droit sur 1km, puis à droite » n'envoie pas les gens au même endroit que si l'on dit « prenez à droite puis tout droit pendant 1 km ».

Enfin, il est également clair que si l'on met de côté leur vertu pédagogique, les deux algorithmes ci-dessus sont parfaitement idiots ; à tout le moins ils contiennent une incohérence.

Il n'y a aucun intérêt à affecter une variable pour l'affecter différemment juste après.

En l'occurrence, on aurait tout aussi bien atteint le même résultat en écrivant simplement :

```
Exemple 1
Variable A en Numérique
Début
A ← 12
Fin

Exemple 2
Variable A en Numérique
Début
A ← 34
Fin
```

3.3.3 Mise en pratique.

Exercice 1.1

Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

```
Variables A, B en Entier
Début
A ← 1
B ← A + 3
A ← 3
Fin
```

Réponse :

```
Après          La valeur des variables est :
A ← 1          A = 1          B = ?
B ← A + 3      A = 1          B = 4
A ← 3          A = 3          B = 4
```

Exercice 1.2

Quelles seront les valeurs des variables A, B et C après exécution des instructions suivantes ?

```
Variables A, B, C en Entier
Début
A ← 5
B ← 3
C ← A + B
A ← 2
C ← B - A
Fin
```

Réponse :

```
Après          La valeur des variables est :
A ← 5          A = 5          B = ?          C = ?
B ← 3          A = 5          B = 3          C = ?
C ← A + B      A = 5          B = 3          C = 8
A ← 2          A = 2          B = 3          C = 8
C ← B - A      A = 2          B = 3          C = 1
```

Exercice 1.3

Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

```
Variables A, B en Entier
Début
A ← 5
B ← A + 4
A ← A + 1
B ← A - 4
Fin
```

Réponse :

Après	La valeur des variables est :	
A ← 5	A = 5	B = ?
B ← A + 4	A = 5	B = 9
A ← A + 1	A = 6	B = 9
B ← A – 4	**A = 6**	**B = 2**

Exercice 1.4

Quelles seront les valeurs des variables A, B et C après exécution des instructions suivantes ?

```
Variables A, B, C en Entier
Début
 A ← 3
B ← 10
C ← A + B
B ← A + B
A ← C
Fin
```

Réponse :

Après	La valeur des variables est :		
A ← 3	A = 3	B = ?	C = ?
B ← 10	A = 3	B = 10	C = ?
C ← A + B	A = 3	B = 10	C = 13
B ← A + B	A = 3	B = 13	C = 13
A ← C	**A = 13**	**B = 13**	**C = 13**

Exercice 1.5

Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

```
Variables A, B en Entier
Début
A ← 5
B ← 2
A ← B
B ← A
Fin
```

Moralité : les deux dernières instructions permettent-elles d'échanger les deux valeurs de B et A ? Si l'on inverse les deux dernières instructions, cela change-t-il quelque chose ?

Réponse :

Après	La valeur des variables est :	
A ← 5	A = 5	B = ?
B ← 2	A = 5	B = 2
A ← B	A = 2	B = 2
B ← A	**A = 2**	**B = 2**

Les deux dernières instructions ne permettent donc pas d'échanger les deux valeurs de B et A, puisque l'une des deux valeurs (celle de A) est ici écrasée.
Si l'on inverse les deux dernières instructions, cela ne changera rien du tout, hormis le fait que cette fois c'est la valeur de B qui sera écrasée.

Exercice 1.6

Plus difficile, mais c'est un classique absolu, qu'il faut absolument maîtriser : écrire un algorithme permettant d'échanger les valeurs de deux variables A et B, et ce quel que soit leur contenu préalable.

Réponse :

```
Début
…
C ← A
A ← B
B ← C
Fin
```

On est obligé de passer par une variable dite temporaire (la variable C).

Exercice 1.7

Une variante du précédent : on dispose de trois variables A, B et C. Ecrivez un algorithme transférant à B la valeur de A, à C la valeur de B et à A la valeur de C (toujours quels que soient les contenus préalables de ces variables).

Réponse :

```
Début
…
D ← C
C ← B
B ← A
A ← D
Fin
```

En fait, quel que soit le nombre de variables, une seule variable temporaire suffit…

3.4 Expressions et opérateurs

Si on fait le point, on s'aperçoit que dans une instruction d'affectation, on trouve :

À gauche de la flèche, un nom de variable, et uniquement cela. En ce monde empli de doutes qu'est celui de l'algorithmique, c'est une des rares règles d'or qui marche à tous les coups : si on voit à gauche d'une flèche d'affectation autre chose qu'un nom de variable, on peut être certain à 100% qu'il s'agit d'une erreur.

À droite de la flèche, ce qu'on appelle une expression. Voilà encore un mot qui est trompeur ; en effet, ce mot existe dans le langage courant, où il revêt bien des significations. Mais en informatique, le terme d'expression ne désigne qu'une seule chose, et qui plus est une chose très précise :

Une expression est un ensemble de valeurs, reliées par des opérateurs, et équivalent à une seule valeur.

Cette définition vous paraît peut-être obscure. Mais réfléchissez-y quelques minutes, et vous verrez qu'elle recouvre quelque chose d'assez simple sur le fond.

Par exemple, voyons quelques expressions de type numérique.

Ainsi :

```
7
5+4
123-45+844
Toto-12+5-Riri
```

Sont toutes des expressions valides, pour peu que Toto et Riri soient bien des nombres.

Car dans le cas contraire, la quatrième expression n'a pas de sens. En l'occurrence, les opérateurs que j'ai employés sont l'addition (+) et la soustraction (-).

Revenons pour le moment sur l'affectation. Une condition supplémentaire (en plus des deux précédentes) de validité d'une instruction d'affectation est que :

L'expression située à droite de la flèche soit du même type que la variable située à gauche. C'est très logique : on ne peut pas ranger convenablement des outils dans un sac à provision, ni des légumes dans une trousse à outils… sauf à provoquer un résultat catastrophique.

Si l'un des trois points énumérés ci-dessus n'est pas respecté, la machine sera incapable d'exécuter l'affectation, et déclenchera une erreur (est-il besoin de dire que si aucun de ces points n'est respecté, il y aura aussi erreur !)

On va maintenant détailler ce que l'on entend par le terme d'opérateur.

Un opérateur est un signe qui relie deux valeurs, pour produire un résultat.

Les opérateurs possibles dépendent du type des valeurs qui sont en jeu.

3.4.1 Opérateurs numériques

Ce sont les quatre opérations arithmétiques tout ce qu'il y a de classique.

+ : Addition

- : Soustraction

* : Multiplication

/ : Division

Mentionnons également le ^ qui signifie « puissance ». 45 au carré s'écrira donc 45 ^ 2.

Enfin, on a le droit d'utiliser les parenthèses, avec les mêmes règles qu'en mathématiques. La multiplication et la division ont « naturellement » priorité sur l'addition et la soustraction. Les parenthèses ne sont ainsi utiles que pour modifier cette priorité naturelle.

Cela signifie qu'en informatique, 12 * 3 + 5 et (12 * 3) + 5 valent strictement la même chose, à savoir 41. Pourquoi dès lors se fatiguer à mettre des parenthèses inutiles ?

En revanche, 12 * (3 + 5) vaut 12 * 8 soit 96. Rien de difficile là-dedans, que du normal.

3.4.2 Opérateur alphanumérique : &

Cet opérateur permet de concaténer, autrement dit d'agglomérer, deux chaînes de caractères.

Par exemple :

```
Variables A, B, C en Caractère
Début
A ← "Gloubi"
B ← "Boulga"
C ← A & B
Fin
```

La valeur de C à la fin de l'algorithme est "GloubiBoulga"

3.4.3 Opérateurs logiques (ou booléens) :

Il s'agit du ET, du OU, du NON et du mystérieux (mais rarissime XOR). Nous les laisserons de côté… provisoirement, soyez-en sûrs.

3.4.4 Mise en pratique.

Exercice 1.8

Que produit l'algorithme suivant ?

```
Variables A, B, C en Caractères
Début
A ← "423"
B ← "12"
C ← A + B
Fin
```

Réponse : Il ne peut produire qu'une erreur d'exécution, puisqu'on ne peut pas additionner des caractères.

Exercice 1.9

Que produit l'algorithme suivant ?

```
Variables A, B, C en Caractères
Début
A ← "423"
B ← "12"
C ← A & B
Fin
```

Réponse : En revanche, on peut les concaténer. A la fin de l'algorithme, C vaudra donc "42312".

3.5 Deux remarques pour terminer

Maintenant que nous sommes familiers des variables et que nous les manipulons les yeux fermés (mais les neurones en éveil, toutefois), j'attire votre attention sur la trompeuse similitude de vocabulaire entre les mathématiques et l'informatique.

En mathématiques, une « variable » est généralement une inconnue, qui recouvre un nombre non précisé de valeurs. Lorsque j'écris :

$$y = 3\,x + 2$$

Les « variables » x et y satisfaisant à l'équation existent en nombre infini (graphiquement, l'ensemble des solutions à cette équation dessine une droite). Lorsque j'écris :

$$ax^2 + bx + c = 0$$

La « variable » x désigne les solutions à cette équation, c'est-à-dire zéro, une ou deux valeurs à la fois…

En informatique, une variable possède à un moment donné une valeur et une seule.

A la rigueur, elle peut ne pas avoir de valeur du tout (une fois qu'elle a été déclarée, et tant qu'on ne l'a pas affectée.

A signaler que dans certains langages, les variables non encore affectées sont considérées comme valant automatiquement zéro).

Mais ce qui est important, c'est que cette valeur justement, ne « varie » pas à proprement parler. Du moins ne varie-t-elle que lorsqu'elle est l'objet d'une instruction d'affectation.

La deuxième remarque concerne le signe de l'affectation.

En algorithmique, comme on l'a vu, c'est le signe ←. Mais en pratique, la quasi-totalité des langages emploient le signe égal.

Et là, pour les débutants, la confusion avec les maths est également facile.

En maths, A = B et B = A sont deux propositions strictement équivalentes.

En informatique, absolument pas, puisque cela revient à écrire A ← B et B ← A, deux choses bien différentes.

De même, A = A + 1, qui en mathématiques, constitue une équation sans solution, représente en programmation une action tout à fait licite (et de surcroît extrêmement courante).

Donc, attention ! ! ! La meilleure des vaccinations contre cette confusion consiste à bien employer le signe ← en pseudo-code, signe qui a le mérite de ne pas laisser place à l'ambiguïté.

Une fois acquis les bons réflexes avec ce signe, vous n'aurez plus aucune difficulté à passer au = des langages de programmation.

4 Lecture et Ecriture

4.1 De quoi parle-t-on ?

Trifouiller des variables en mémoire vive par un chouette programme, c'est vrai que c'est très marrant, et d'ailleurs on a tous bien rigolé au chapitre précédent.

Cela dit, à la fin de la foire, on peut tout de même se demander à quoi ça sert.

En effet. Imaginons que nous ayons fait un programme pour calculer le carré d'un nombre, mettons 12.
Si on a fait au plus simple, on a écrit un truc du genre :

```
Variable A en Numérique
Début
A ← 12^2
Fin
```

D'une part, ce programme nous donne le carré de 12. C'est très gentil à lui. Mais si l'on veut le carré d'un autre nombre que 12, il faut réécrire le programme. Bof.

D'autre part, le résultat est indubitablement calculé par la machine. Mais elle le garde soigneusement pour elle, et le pauvre utilisateur qui fait exécuter ce programme, lui, ne saura jamais quel est le carré de 12. Re-bof.

C'est pourquoi, heureusement, il existe des d'instructions pour permettre à la machine de dialoguer avec l'utilisateur.

Dans un sens, ces instructions permettent à l'utilisateur de rentrer des valeurs au clavier pour qu'elles soient utilisées par le programme.

Cette opération est la lecture.

Dans l'autre sens, d'autres instructions permettent au programme de communiquer des valeurs à l'utilisateur en les affichant à l'écran.

Cette opération est l'écriture.

Remarque essentielle : A première vue, on peut avoir l'impression que les informaticiens étaient beurrés comme des petits lus lorsqu'ils ont baptisé ces opérations ; puisque quand l'utilisateur doit écrire au clavier, on appelle cette instruction la lecture, et quand il doit lire sur l'écran on l'appelle l'écriture.

Mais avant d'agonir d'insultes une digne corporation, il faut réfléchir un peu plus loin.

Un algorithme, c'est une suite d'instructions qui programme la machine, pas l'utilisateur !

Donc quand on dit à la machine de lire une valeur, cela implique que l'utilisateur va devoir écrire cette valeur.

Et quand on demande à la machine d'écrire une valeur, c'est pour que l'utilisateur puisse la lire.

Lecture et écriture sont donc des termes qui comme toujours en programmation, doivent être compris du point de vue de la machine qui sera chargée de les exécuter, et non de l'utilisateur qui se servira du programme.

Et là, tout devient parfaitement logique.

4.2 Les instructions de lecture et d'écriture

Tout bêtement, pour que l'utilisateur entre la (nouvelle) valeur de Titi, on mettra :

```
Lire Titi
```

Dès que le programme rencontre une instruction Lire, l'exécution s'interrompt, attendant la frappe d'une valeur au clavier.

L'interruption peut durer quelques secondes, quelques minutes ou plusieurs heures : la seule chose qui fera exécuter la suite des instructions, c'est que la touche Entrée (Enter) ait été enfoncée.

Aussitôt que c'est le cas, il se passe deux choses.

Pour commencer, tout ce qui a été frappé avant la touche Entrée (une suite de lettres, de chiffres, ou un mélange des deux) est rentré dans la variable qui suit l'instruction Lire (ici, Titi).

Et ensuite, immédiatement, la machine exécute l'instruction suivante.

Lire est donc une autre manière d'affecter une valeur à une variable.
Avec l'instruction d'affectation, c'est le programmeur qui choisit à l'avance quelle doit être cette valeur.
Avec l'instruction Lire, il laisse ce choix à l'utilisateur.

Dans le sens inverse, pour écrire quelque chose à l'écran, c'est aussi simple que :

```
Ecrire Toto
```

Parenthèse : « les bonnes manières du programmeur » : avant de Lire une variable, il est très fortement conseillé d'écrire des libellés à l'écran, afin de prévenir l'utilisateur de ce qu'il doit frapper (sinon, le pauvre utilisateur passe son temps à se demander ce que l'ordinateur attend de lui… et c'est très désagréable !) :

```
Ecrire "Entrez votre nom : "
Lire NomFamille
```

Lecture et Ecriture sont des instructions algorithmiques qui ne présentent pas de difficultés particulières, une fois qu'on a bien assimilé ce problème du sens du dialogue (homme → machine, ou machine ← homme).

4.3 Mise en pratique.

Exercice 2.1

Quel résultat produit le programme suivant ?

```
Variables val, double numériques
Début
Val ← 231
Double ← Val * 2
Ecrire Val
Ecrire Double
Fin
```

Réponse : On verra apparaître à l'écran 231, puis 462 (qui vaut 231 * 2)

Exercice 2.2

Ecrire un programme qui demande un nombre à l'utilisateur, puis qui calcule et affiche le carré de ce nombre.

Réponse :

```
Variables nb, carr en Entier
Début
Ecrire "Entrez un nombre :"
Lire nb
carr ← nb * nb
Ecrire "Son carré est : ", carr
Fin
```

En fait, on pourrait tout aussi bien économiser la variable carr en remplaçant les deux avant-dernières lignes par :

```
Ecrire "Son carré est : ", nb*nb
```

C'est une question de style ; dans un cas, on privilégie la lisibilité de l'algorithme, dans l'autre, on privilégie l'économie d'une variable.

Exercice 2.3

Ecrire un programme qui demande son prénom à l'utilisateur, et qui lui réponde par un charmant « Bonjour » suivi du prénom. On aura ainsi le dialogue suivant :

machine : Quel est votre prénom ?
utilisateur : Marie-Cunégonde
machine : Bonjour, Marie Cunégonde ! ».

Réponse :

```
Variable prenom en Caractere
Début
Ecrire "Quel est votre prenom ?"
Lire Prenom
Ecrire "Bonjour ", Prenom, " !"
Fin
```

Exercice 2.4

Ecrire un programme qui lit le prix HT d'un article, le nombre d'articles et le taux de TVA, et qui fournit le prix total TTC correspondant. Faire en sorte que des libellés apparaissent clairement.

Réponse :

```
Variables nb, pht, ttva, pttc en Numérique
Début
Ecrire "Entrez le prix hors taxes :"
Lire pht
Ecrire "Entrez le nombre d'articles :"
Lire nb
Ecrire "Entrez le taux de TVA :"
Lire ttva
pttc ← nb * pht * (1 + ttva)
Ecrire "Le prix toutes taxes est : ", pttc
Fin
```

Là aussi, on pourrait squeezer une variable et une ligne en écrivant directement. :

```
Ecrire "Le prix toutes taxes est : ", nb * pht * (1 + ttva)
```

C'est plus rapide, plus léger en mémoire, mais un peu plus difficile à relire (et à écrire !)