Chapitre 2

L'algorithmique

Vous prenez des décisions pour résoudre des problèmes tous les jours. Ces problèmes peuvent être aussi importants que vous trouvez un nouvel emploi ou aussi futiles que choisir l'émission à regarder à la télé ce soir. Vous résolvez des problèmes tous les jours à la maison et au travail.

De même, les développeurs informatiques résolvent des problèmes. Un algorithme est une solution logique comportant un nombre déterminé d'étapes afin de résoudre un problème donné. On peut penser à un algorithme comme à des instructions écrites en langage de tous les jours qui résoudront un problème de manière logique. Même si ces instructions sont simples, elles doivent être, autant qu'il se peut complètes et exactes.

Une réflexion et une préconception appropriées sont importantes !

OBJECTIFS

- Identifier les concepts relatifs aux algorithmes.

Notes

Leçon 2-A

Pourquoi l'algorithmique ?

L'algorithmique est un passage obligé pour les programmeurs en herbes. Avant de vouloir se lancer tête baissée dans l'écriture d'un code informatique, il est important de comprendre comment on va communiquer avec la machine.

Comment on va demander à l'ordinateur d'effectuer tel ou tel opération ?

Même si l'algorithmique peut avoir une utilisation plus large, pour la résolution de problème variées, le but de cette formation est bien de voir l'algorithmique comme un premier pas vers l'écriture du code informatique afin de résoudre un problème.

OBJECTIFS

À la fin de la présente leçon, vous serez en mesure :

- de définir c'est quoi l'algorithmique.
- de comprendre la composition d'un algorithme.
- de définir c'est quoi un programme.

Selon Wikipédia (https://fr.wikipedia.org/wiki/Algorithmique)

« L'***algorithmique*** est l'étude et la production de règles et techniques qui sont impliquées dans la définition et la conception d'algorithmes, c'est-à-dire de processus systématiques de résolution d'un problème permettant de décrire précisément des étapes pour résoudre un problème algorithmique. »

Qu'est-ce qu'un algorithme ?

Toujours selon Wikipédia (https://fr.wikipedia.org/wiki/Algorithme)

« Un ***algorithme*** est une suite finie et non ambiguë d'instructions et d'opérations permettant de résoudre une classe de problèmes. »

Le mot algorithme vient d'Al-Khwârizmî, nom d'un mathématicien persan du IX^{e} siècle.

Le domaine qui étudie les algorithmes est appelé l'algorithmique. On retrouve aujourd'hui des algorithmes dans de nombreuses applications telles que le fonctionnement des ordinateurs, la cryptographie, le routage d'informations, la planification et l'utilisation optimale des ressources, le traitement d'images, le traitement de textes, la bio-informatique, etc.

Si on ne consacre pas suffisamment de temps à la conception d'un algorithme correct, le système est destiné à devenir un cauchemar coûteux à mettre en œuvre ou un triste échec. On n'insistera jamais trop sur l'importance de la conception de l'algorithme. Avec la création de bonnes spécifications complètes, la conception de l'algorithme est l'étape la plus importante du cycle de développement.

Un algorithme est une méthode générale pour résoudre un type de problèmes. Il est dit correct lorsque, pour chaque instance du problème, il se termine en produisant la bonne sortie, c'est-à-dire qu'il résout le problème posé. Un ***algorithme*** c'est une suite d'instructions ordonnées qui a pour but de trouver un résultat à partir de données connues. Vous utilisez des algorithmes consciemment ou non dans votre quotidien.

Premier exemple : Vous avez soif !

1. Vous prenez un verre dans l'armoire
2. Vous placez le verre sous le robinet
3. Vous ouvrez le robinet
4. Vous attendez jusqu'à ce que le verre d'eau soit plein
5. Vous fermez le robinet
6. Vous buvez l'eau du verre

Cet algorithme est une suite d'instructions que vous faites régulièrement sans même vous en rendre compte, de manière inconsciente.

Serait-il logique de commencer avec l'instruction nº 6 dans l'algorithme précédent ?

Non puisque dans ce cas, vous vous trouveriez à tenter de boire un verre vide de tout contenu, donc, à boire un verre rempli d'air.

Ceci n'est qu'un exemple bien simple et, nous pourrions facilement le complexifier un peu. Par exemple, vous avez soif et vous vous demandez qu'est-ce que vous désirez boire afin d'étancher votre soif.

Selon votre goût du moment, vous pourriez vouloir boire une boisson chaude ou une boisson froide. À partir de votre réponse, vous allez devoir choisir le contenant dans lequel vous allez déposer votre boisson choisie, une tasse pour une boisson chaude ou un verre pour une boisson froide. Bien entendu, dans votre vie au quotidien, vous prenez le contenant correspondant à ce que vous désirez boire de manière automatique et sans même vous poser la question mais, un ordinateur ne le sait pas lui. Il attend que vous lui disiez ce que vous désirez faire ou, boire dans cet exemple.

Les ***algorithmes*** devraient toujours être indépendants des langages de programmation utilisés. Les ***solutions logiques*** à un problème peuvent être traduites en n'importe quel langage informatique. Les algorithmes peuvent nécessiter quelques ajustements en raison des limites d'un langage choisi, afin de conserver un reflet véritable entre le ***code*** et l'***algorithme***. Cependant, il s'agit d'une situation exceptionnelle et, dans la mesure du possible, la règle de l'indépendance à l'égard du langage doit être suivie.

Il y a normalement plusieurs ***algorithmes*** ou façons de résoudre un problème. Si un ***algorithme*** donne les bons résultats, il s'agit alors d'une solution valide. Mais certaines solutions sont plus efficaces, occupent moins d'espace et leur exécution est plus rapide. Ces solutions seraient alors privilégiées. Les programmeurs novices doivent se concentrer principalement sur la ***résolution du problème*** et se concentrer sur l'***efficacité*** que plus tard.

Des programmeurs ont déjà créé des millions d'algorithmes. Des tâches courantes comme le tri et la recherche ont été peaufinées en algorithmes serrés et efficaces, qui ne peuvent plus être optimisés davantage avec le matériel actuel. Des manuels entiers énumèrent les algorithmes courants pour certains problèmes.

Ne réinventez pas la roue !

S'il existe déjà un algorithme qui ne peut être amélioré, utilisez-le.

Avant de poursuivre, posons-nous la question suivante :

Quelle est l'importance de l'algorithmique en programmation ?

L'importance de l'algorithmique en programmation est difficile à surestimer. Elle forme le cœur de tout développement logiciel efficace et de qualité. Voici quelques raisons pour lesquelles l'algorithmique est si cruciale :

1. ***Efficacité*** : Des algorithmes efficaces peuvent résoudre des problèmes de manière plus rapide et avec moins de ressources, ce qui est essentiel pour les applications à grande échelle ou les systèmes temps réel.
2. ***Organisation*** : L'algorithmique aide à structurer et à organiser la logique d'un programme. Des algorithmes clairs et bien conçus rendent le code plus compréhensible et plus facile à maintenir.
3. ***Optimisation*** : L'optimisation des algorithmes est essentielle pour améliorer les performances des programmes. Des algorithmes optimisés peuvent réduire le temps d'exécution, la consommation de mémoire et d'autres ressources.
4. ***Solutions aux problèmes complexes*** : L'algorithmique fournit des outils pour résoudre des problèmes complexes de manière systématique et efficace. Elle permet de décomposer des problèmes en sous-problèmes plus gérables et de concevoir des solutions élégantes.
5. ***Adaptabilité*** : Les algorithmes bien conçus sont souvent plus flexibles et adaptables aux changements de spécifications ou de besoins. Cela permet aux logiciels de s'adapter plus facilement à l'évolution des exigences.
6. ***Fondation pour d'autres domaines*** : L'algorithmique forme la base de nombreux autres domaines de l'informatique, tels que l'intelligence artificielle, l'apprentissage automatique, la cryptographie, etc. Une compréhension solide de l'algorithmique est donc essentielle pour progresser dans ces domaines.

En résumé, l'algorithmique est fondamentale pour la programmation car elle fournit les outils nécessaires pour concevoir, analyser et optimiser des solutions à une variété de problèmes informatiques.

L'intelligence des ordinateurs

Les ordinateurs ne sont pas intelligents. Ils ne peuvent pas raisonner, penser par eux-mêmes, ni même deviner quoi que ce soit. Ils peuvent uniquement exécuter un ensemble d'instructions prédéfini. Ces instructions sont exécutées à des vitesses mesurées en millionièmes de seconde, mais il n'y a aucune forme d'intelligence. Un ordinateur est simplement une machine qui ***exécute*** un programme, des instructions, pour résoudre un problème de façon très rapide.

Avant de commencer à programmer, vous devez oublier la notion d'***ordinateur intelligent***.

L'analogie suivante illustre à quel point un ordinateur est intelligent.

Imaginez que vous venez de rencontrer un voyageur de l'espace. Ce voyageur a appris quelques mots de votre langue. Les mots compris de vous deux sont ***gauche***, ***droite***, ***soulever***, ***lâcher***, ***jambe*** et ***avancer***.

Vous voulez voir le voyageur de plus près. Le problème auquel vous faites face, c'est que le voyageur est à dix pas de vous. Lorsque vous dites « ***Venez ici s'il vous plaît*** », il vous regarde comme si c'était vous qui arriviez d'une autre planète. Il n'a aucune idée de ce que vous lui demandez de faire.

Vous devez lui donner des instructions détaillées, étape par étape, sur ce que vous désirez qu'il exécute.

- Soulever jambe gauche.
- Avancer jambe gauche.
- Lâcher jambe gauche.
- Soulever jambe droite.
- Avancer jambe droite.
- Lâcher jambe droite.
- Soulever jambe gauche.
- Etc.

Bien entendu, il vous faudrait répéter ces commandes plusieurs fois, jusqu'à ce que le voyageur soit à la distance que vous voulez. Une séquence d'instructions ainsi répétées de nombreuses fois est appelée une boucle. Si votre ami voyageur connaissait quelques mots de plus, vous pourriez réduire le nombre d'instructions nécessaires.

- Soulever jambe gauche.
- Avancer jambe gauche.
- Lâcher jambe gauche.
- Soulever jambe droite.
- Avancer jambe droite.
- Lâcher jambe droite.
- Répéter.

En ajoutant le mot ***Répéter*** à votre répertoire, le nombre d'instructions est réduit de 30 à 7 environ. Un problème important demeure cependant.

Comment le voyageur saura quand s'arrêter ?

Les instructions précédentes illustrent un problème classique de la création de boucles, la ***boucle sans fin***. Chaque boucle doit inclure un ***point d'arrêt***.

Grâce à l'inclusion d'un ***point d'arrêt*** dans la boucle, le voyageur saura quand arrêter d'exécuter la série d'instructions à répéter.

- Soulever jambe gauche.
- Avancer jambe gauche.
- Lâcher jambe gauche.
- Soulever jambe droite.
- Avancer jambe droite.
- Lâcher jambe droite.
- Répéter jusqu'à ce que je dise ***Arrête***.

Un programme

Un ***programme*** c'est un ou plusieurs algorithmes destinés à être exécutés par une machine, écrit dans un langage qu'elle comprend. Un algorithme se doit d'être indépendant du langage de programmation utilisé, c'est-à-dire qu'un algorithme peut être transcrit en langage compréhensible par l'ordinateur en utilisant différents langages informatiques comme le C#, le Java, le JavaScript, le PHP, etc.

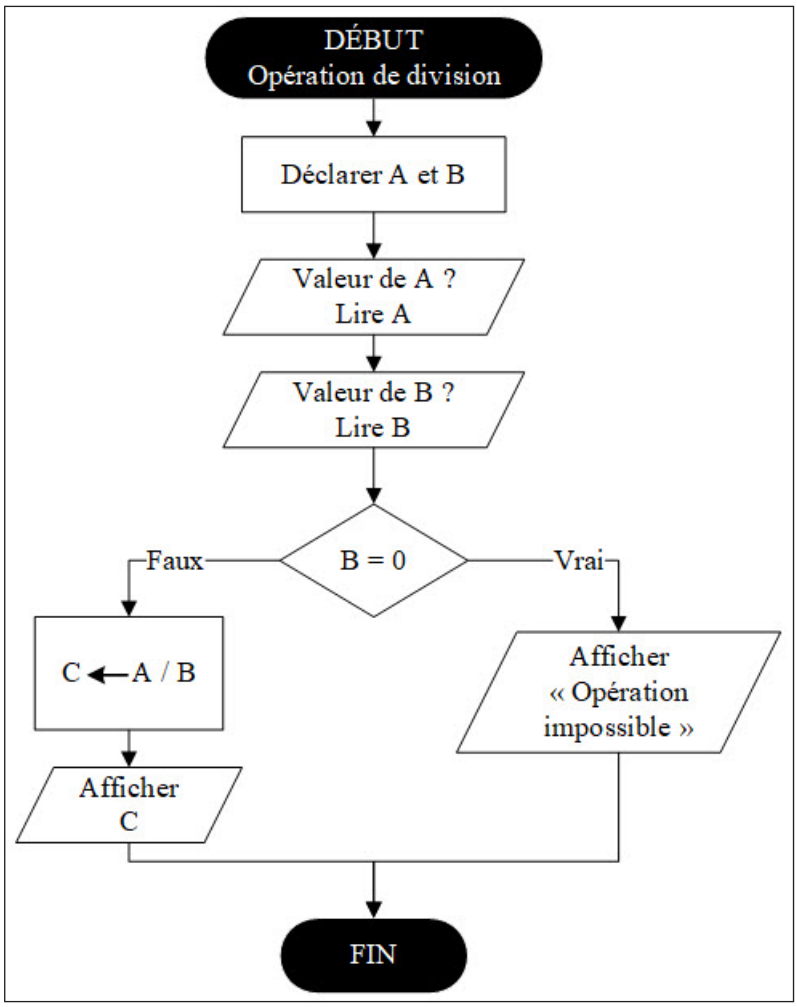
Un langage informatique n'est qu'un véhicule que l'on utilise afin de communiquer les instructions qui vont résoudre le problème soumis et, nous vous en ferons la preuve, le moment venu. Pour l'instant, nous allons nous concentrer sur la résolution de problèmes en utilisant l'algorithmique, une façon d'écrire les étapes nécessaires à la résolution du problème soumis.

Notions de base de l'élaboration d'un algorithme

En programmation informatique, un algorithme peut être représenté en utilisant deux formes :

- L'ordinogramme
- Le pseudocode

Un ordinogramme est un organigramme de traitement de l'information, représentant l'enchaînement des opérations d'un programme. La prochaine figure vous présente un algorithme sous la forme d'un ordinogramme.



Le pseudocode, également appelé ***LDA***, pour ***L***angage de ***D***escription d'***A***lgorithmes, est une façon de décrire un algorithme en langage presque naturel, sans référence à un langage de programmation en particulier.

L'écriture en pseudocode permet souvent de bien prendre toute la mesure de la difficulté de la mise en œuvre de l'algorithme, et de développer une démarche structurée dans la construction de celui-ci. En effet, son aspect descriptif permet de décrire avec plus ou moins de détail l'algorithme, permettant de ce fait de commencer par une vision très large et de passer outre temporairement certains aspects complexes, ce que n'offre pas la programmation directe. La prochaine figure vous présente l'algorithme de l'ordinogramme précédent sous forme de pseudocode.

```
ALGORITHME Opération de division

VARIABLES
        A, B, C : entier

DÉBUT
        AFFICHER("Quelle est la valeur de A ?")
        LIRE(A)
        AFFICHER("Quelle est la valeur de B ?")
        LIRE(B)
        SI B = 0 ALORS
                AFFICHER("Opération impossible.")
        SINON
                C ← A / B
                AFFICHER(C)
        FIN SI
FIN
```

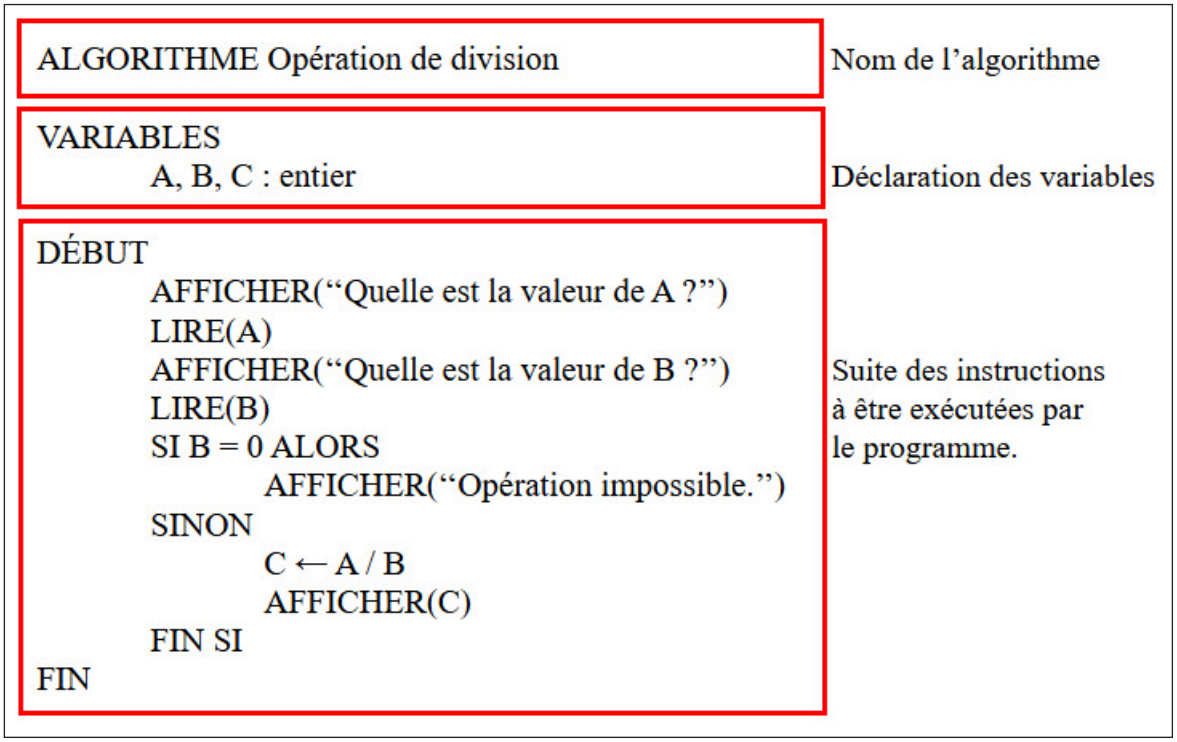
Dans le cadre de votre formation, nous opterons pour la création d'algorithme en utilisant le pseudocode, une représentation de la logique beaucoup plus facile à composer que l'utilisation d'un ordinogramme.

Un ***algorithme*** bien conçu se compose de trois sections. La première ligne doit spécifier le nom de votre algorithme, à quoi il servira. Cette ligne est suivie de la section de déclaration des variables et des constantes, les éléments connus d'un problème, que nous allons couvrir dans une prochaine leçon. Cette section est identifiée avec les mots clé ***VARIABLES*** ou ***CONSTANTES***. Nous complétons le tout avec la section des instructions proprement dite. Cette section débute avec le mot clé ***DÉBUT*** et se termine avec le mot clé ***FIN***.

Que représente cette flèche pointant vers la gauche dans ce pseudocode ?

Cette flèche dit simplement, prend le résultat de l'opération qui se situe à ma droite et affecte ce résultat à la variable à ma gauche. Ce résultat peut être le résultat d'une opération ou tout simplement une valeur saisie par l'utilisateur, comme nous le verrons plus tard.

La prochaine figure vous montre comment l'algorithme précédent est composé.



L'algorithmique est de plus en plus présent, dans notre vie quotidienne. Une recette de cuisine peut être réduite à un algorithme si on peut réduire sa spécification aux éléments constitutifs :

- Des entrées (les ingrédients, le matériel utilisé) ;
- Des instructions élémentaires simples (frire, flamber, rissoler, braiser, blanchir, etc.), dont les exécutions dans un ordre précis amènent au résultat voulu ;
- Un résultat : le plat préparé.

Reprenons cette conception de base en la transformant en une recette de cuisine.

ALGORITHME Gâteau au fromage

CONSTANTES
Croûte
Chapelure de biscuit Graham ← 310 ml
Sucre ← 30 ml
Beurre non salé fondu ← 60 ml
Papier d'aluminium de 46 cm de large
Garniture
Sucre ← 310 ml
Farine tout usage non blanchie ← 30 ml
Fromage à la crème, tempéré ← 4 paquets de 250 gr
Crème sure ← 180 ml
Œuf ← 4
Jaune d'œuf ← 2
Extrait de vanille ← 5 ml

DÉBUT

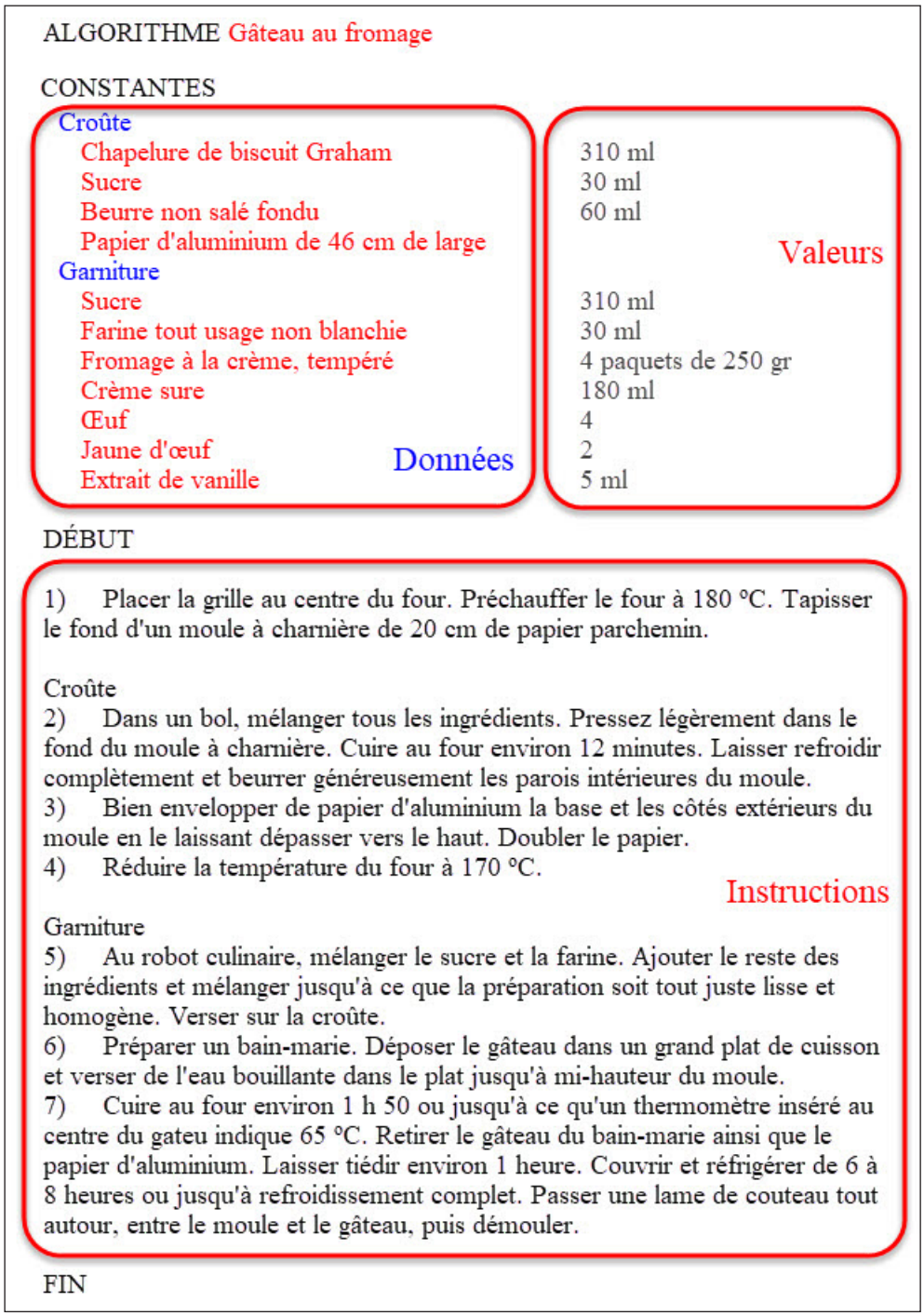
1) Placer la grille au centre du four. Préchauffer le four à 180 °C. Tapisser le fond d'un moule à charnière de 20 cm de papier parchemin.

Croûte
2) Dans un bol, mélanger tous les ingrédients. Pressez légèrement dans le fond du moule à charnière. Cuire au four environ 12 minutes. Laisser refroidir complètement et beurrer généreusement les parois intérieures du moule.
3) Bien envelopper de papier d'aluminium la base et les côtés extérieurs du moule en le laissant dépasser vers le haut. Doubler le papier.
4) Réduire la température du four à 170 °C.

Garniture
5) Au robot culinaire, mélanger le sucre et la farine. Ajouter le reste des ingrédients et mélanger jusqu'à ce que la préparation soit tout juste lisse et homogène. Verser sur la croûte.
6) Préparer un bain-marie. Déposer le gâteau dans un grand plat de cuisson et verser de l'eau bouillante dans le plat jusqu'à mi-hauteur du moule.
7) Cuire au four environ 1 h 50 ou jusqu'à ce qu'un thermomètre inséré au centre du gateu indique 65 °C. Retirer le gâteau du bain-marie ainsi que le papier d'aluminium. Laisser tiédir environ 1 heure. Couvrir et réfrigérer de 6 à 8 heures ou jusqu'à refroidissement complet. Passer une lame de couteau tout autour, entre le moule et le gâteau, puis démouler.

FIN

Décomposons cet algorithme afin d'en comprendre le sens. Examinez la prochaine figure.



En premier lieu, nous retrouvons le nom de notre algorithme, dans ce cas-ci, ***Gâteau au fromage***, immédiatement après le mot clé ***ALGORITHME***. Suit une première section, la section ***CONSTANTES***, dans laquelle nous identifions les ***données*** et les ***valeurs*** des données nécessaires à l'exécution de notre algorithme. Pour terminer, dans la dernière section qui se trouve entre les mots clés ***DÉBUT*** et ***FIN*** se trouve les instructions à exécuter pour atteindre le but désiré. Il est normal de donner un nom à notre algorithme puisque ce dernier identifie de manière spécifique son but. Pour ce qui est des données identifiées dans la section ***CONSTANTES***, ce sont des valeurs qui ne font pas l'objet d'une saisie quelconque par l'utilisateur. Dans le cas de notre gâteau, ces valeurs ont déjà été identifiées et sont considérées comme des valeurs constantes, ce qui nous permet de pouvoir les insérer dans cette section.

Si vous suivez les instructions qui sont contenues entre les mots clés ***DÉBUT*** et ***FIN***, en utilisant les ***données*** identifiées dans la section ***CONSTANTES*** selon les ***valeurs*** qu'elles contiennent, vous devriez obtenir le résultat souhaité soit, celui de la prochaine figure, un magnifique gâteau au fromage.

Dans certaines situations, il est possible de pouvoir modifier l'ordre d'exécution des instructions sans que le résultat souhaité ne soit modifié. Prenons un nouvel exemple afin de bien comprendre le tout.

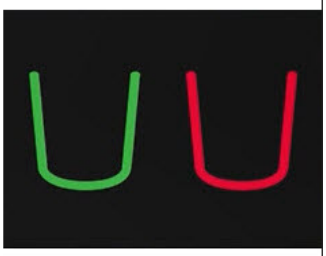
Nous avons deux seaux, un vert et un rouge qui sont tous les deux vides. Ce sont nos données d'entrée, nos ***données*** connues.

```
ALGORITHME Seau

VARIABLES
    Seau vert vide
    Seau rouge vide

DÉBUT

FIN
```



Par la suite, nous exécutons une suite d'***instructions*** en commençant par le remplissage de notre seau vert suivi du remplissage de notre seau rouge.

```
ALGORITHME Seau

VARIABLES
    Seau vert vide
    Seau rouge vide

DÉBUT
    Emplir seau vert
    Emplir seau rouge
FIN
```

Nous complétons notre suite d'instructions en vidant notre seau rouge.

```
ALGORITHME Seau

VARIABLES
    Seau vert vide
    Seau rouge vide

DÉBUT
    Emplir seau vert
    Emplir seau rouge
    Vider seau rouge
FIN
```

Nous pourrions très bien intervertir nos instructions de remplissage en commençant par le seau rouge et terminer par le seau vert et on obtiendrait le même résultat au final.

```
ALGORITHME Seau

VARIABLES
    Seau vert vide
    Seau rouge vide

DÉBUT
    Emplir seau rouge
    Emplir seau vert
    Vider seau rouge
FIN
```

Quant est-il si nous modifions l'ordre de nos instructions pour débuter avec Vider le seau rouge ?

Après avoir exécuté notre séquence d'instructions, nous allons nous retrouver avec deux seaux pleins de liquide, contrairement au résultat obtenu précédemment mais, ***est-ce là le résultat que nous désirions obtenir ?***

```
ALGORITHME Seau

VARIABLES
    Seau vert vide
    Seau rouge vide

DÉBUT
    Vider seau rouge
    Emplir seau rouge
    Emplir seau vert
FIN
```

L'idée de vouloir commencer par vider un seau vide est totalement dénuée de sens, j'en conviens. Par contre, c'est l'idée présentée dans cet exemple qui prime sur la logique.

Vous venez tout juste de constater à travers tous ces exemples que dans certaines situations, l'ordre d'exécution des instructions peut ne pas avoir d'effet sur le résultat final de la solution à un problème par contre, un ordre différent pourrait aussi avoir des répercussions qui pourraient être néfaste dans le résultat obtenu.

Récapitulation

Un algorithme c'est une suite d'instructions ordonnées qui a pour but de trouver un résultat à partir de données connues.

Dans certaines situations, l'ordre des instructions n'a aucune incidence sur le résultat final par contre, dans d'autre cas, cet ordre pourrait produire un résultat non désiré.

Leçon 2-B

Ce qu'un ordinateur peut faire

Il est faux de prétendre qu'un ordinateur est intelligent, il ne fait qu'exécuter les instructions qui lui ont été fournies par un développeur informatique.

Un ordinateur est limité dans ce qu'il peut comprendre et, vous allez devoir composer avec ces instructions limitées afin de lui faire exécuter ce que vous désirez qu'il exécute.

Il n'est donc pas faux d'affirmer que, par défaut, l'intelligence d'un ordinateur c'est l'intelligence de ceux et celles qui le programment.

OBJECTIFS

À la fin de la présente leçon, vous serez en mesure :

- d'identifier les instructions de base qu'un ordinateur peut exécuter.

Qu'est-ce qu'un ordinateur peut faire ?

Une ***instruction*** est une action effectuée dans le but d'atteindre un résultat désiré. Les instructions sont décrites à l'aide de verbes ou mots d'action, comme ***attacher*** ses souliers, ***retirer*** sa cravate ou se ***peigner*** les cheveux. Comme vous vous concentrez sur des problèmes en utilisant un ordinateur, vous devez tenir compte des types d'instructions que peut exécuter un ordinateur. Il existe six types d'instructions de base que l'ordinateur peut effectuer :

Les ordinateurs peuvent recevoir de l'information

Les ordinateurs peuvent recevoir de l'information de diverses sources, le ***clavier***, un ***fichier***, un ***microphone***, la ***souris***, etc. On désigne habituellement l'information ainsi reçue par un ordinateur sous le nom de ***données de saisie***. L'instruction de saisie est généralement décrite par les mots ***LIRE*** ou ***EXTRAIRE***.

Par exemple,

LIRE MATRICULE EMPLOYÉ
EXTRAIRE DONNÉES DE TRANSFERT

Les ordinateurs peuvent envoyer de l'information

Les ordinateurs peuvent aussi envoyer de l'information à diverses sources, l'***écran de l'ordinateur***, un ***fichier***, un ***haut-parleur***, une ***imprimante***, un ***modem***, etc. On désigne habituellement l'information envoyée par l'ordinateur sous le nom de ***données de sortie***. Les mots ***AFFICHER***, ***ÉCRIRE*** et ***IMPRIMER*** décrivent généralement l'***opération de sortie***.

Par exemple,

IMPRIMER « ALLÔ »
ÉCRIRE ADRESSE
AFFICHER RAPPORT CLIENT

Les ordinateurs peuvent faire des calculs

Les ordinateurs peuvent calculer. Ils peuvent faire des ***opérations mathématiques***.

Par exemple,

PAIE NETTE = PAIE BRUTE – IMPÔT
IMPÔT = SOUS-TOTAL * TAUX IMPÔSITION
MOYENNE DE LA CLASSE = TOTAL DES NOTES / NOMBRE D'ÉTUDIANTS

Les ordinateurs peuvent manipuler les données

Les ordinateurs peuvent manipuler de l'information en mémoire.

Par exemple,

INITIALISER TOTAL À ZÉRO
AFFECTER « JEAN » À DÉVELOPPEUR

Les ordinateurs peuvent comparer des données

Les ordinateurs peuvent comparer des données et effectuer une série d'actions selon le résultat de la comparaison. Les décisions sont décrites par les mots ***SI*** et ***SINON***.

Par exemple,

```
SI HEURES TRAVAILLÉES PLUS GRAND QUE 40 ALORS
    TEMPS SUPPLÉMENTAIRE = HEURES TRAVAILLÉES – 40
SINON
    TEMPS SUPPLÉMENTAIRE = 0
FIN SI
```

Les ordinateurs peuvent répéter une tâche

Les ordinateurs peuvent accomplir la même série d'instructions de manière répétitive, jusqu'à ce qu'une condition soit atteinte. L'opération de répétition peut prendre différentes formes comme vous allez le découvrir.

Par exemple,

```
TANT QUE PAS LA FIN DU FICHIER
    LIRE ENREGISTREMENT
    AFFICHER PRÉNOM
FIN TANT QUE
```

Les limitations de l'ordinateur

L'ordinateur a ses ***limitations***. Il est inconcevable qu'un ordinateur puisse fonctionner toujours sans erreurs. Pour les besoins des cours de votre programme d'études, nous n'allons pas considérer les erreurs de sources matérielles. Lorsqu'un ordinateur fait une erreur ce sont des erreurs de traitement qui sont généralement causés par :

- Une logique déficiente
- Une mauvaise conception du programme

Les ***erreurs*** sont généralement causées par des individus, des programmeurs ou des utilisateurs, pas nécessairement par l'ordinateur. D'où la phrase ***Garbage In Garbage Out*** qui signifie ***à données inexactes, résultats erronés***. Puisque c'est le programmeur qui est responsable de donner les instructions à l'ordinateur pour l'exécution d'une tâche, toute ***erreur de logique*** est sa responsabilité et non celle de l'ordinateur. Ce dernier ne fait qu'exécuter exactement ce que le programmeur lui demande. Prenez l'exemple suivant. Dans un programme de calcul de paie, on sait que le salaire brut est calculé en multipliant le nombre d'heures travaillées par le taux horaire.

Salaire brut = Nombre d'heures * taux horaire

Donc un employé qui travaille 40 heures à un taux de $ 20.00 l'heure, devra avoir un salaire brut de $ 800.00. Maintenant, si vous faites une erreur et que dans votre programme, vous utilisez l'opérateur d'addition (+) au lieu de l'opérateur de multiplication (*) le résultat sera complètement différent.

Salaire brut = Nombre d'heures + taux horaire
Salaire brut = 40 + 20.00
Salaire brut = 60.00

Les exemples précédents sont des exemples simples de type faute de frappe qui sont faciles à détecter et à corriger. Cependant, il y a des erreurs beaucoup plus profondes et difficiles à détecter. Les erreurs de ***conception*** et de ***logique*** d'un programme sont souvent responsables de la perte de plusieurs heures précieuses. Il n'est pas rare de passer quelques heures de travail à essayer de trouver un ***bogue*** qui provoque des comportements non sollicités dans vos programmes.

Un exemple de ce type de ***bogue*** est une ***boucle sans fin***. Un programme entre dans un bloc de code qui lui demande de répéter une série d'instructions un certain nombre de fois ou pendant qu'une certaine condition persiste. Par contre dans la logique de ce bloc de code, les instructions fournis par le programmeur, il n'y a aucune instruction qui permet de mettre fin à la boucle. Prenez l'exemple suivant.

```
ALGORITHME Nom de l'algorithme

VARIABLES
        A ← 0 : entier
        B ← 0 : entier                  Déclaration des variables
        X ← 0 : entier

DÉBUT
        TANT QUE X < 10 FAIRE           (Début de la boucle)
                A ← B + X               (Instruction 1)
                B ← B + 1               (Instruction 2)
        FIN TANT QUE                    (Fin de la boucle)
        AFFICHER(X)
FIN
```

Dans la logique ci-dessus, la ***boucle*** comporte deux instructions, ***Instruction 1*** et ***Instruction 2***. Nous voulons répéter ces instructions jusqu'au moment où la variable ***X*** atteint une valeur plus grande ou égale à 10. Lorsque ***X est plus grand ou égal à 10***, la boucle prend fin puisque sa condition d'entrée ne sera plus vraie. Mais dans ce cas, à l'intérieure de la boucle, il n'y a aucune instruction qui permet de ***modifier X*** donc, cette boucle continue son exécution sans jamais s'arrêter. C'est ce qu'on appelle une ***boucle sans fin***. Un programme contenant une telle boucle va certainement ***geler***. De là l'importance de valider la logique avec des essais avant de produire le code final.

Note : les boucles et le pseudocode seront traités en profondeur plus loin dans ce cours.

Le point important dans ces exemples est que l'ordinateur va toujours exécuter exactement ce qu'on lui demande. Alors si vous créez un programme et que le résultat obtenu n'est pas exactement ce que vous attendiez, ne changez pas d'ordinateur, changez plutôt la logique de votre algorithme.

Récapitulation

Un ordinateur peut ...

- recevoir de l'information
- envoyer de l'information
- faire des calculs
- manipuler des données
- comparer des données
- répéter une tâche

Notes