IREM Clermont-Ferrand
Journée d'information
sur les nouveaux programmes de Seconde

Année 2009-2010
Malika More

# Initiation à l'algorithmique

... et à la programmation

## Contenu de l'atelier

- Des algorithmes (à destination des êtres humains) - Outils : Papier/Crayon
  - ... parce-que c'est le cœur du sujet
- Et des programmes (à destination des ordinateurs) - Outil : AlgoBox
  - ... parce-que l'un ne va pas sans l'autre
- En insistant sur les faits suivants :
  - les programmes informatiques ne sont là que pour voir fonctionner les algorithmes
  - la maîtrise d'un langage de programmation n'est pas l'objectif visé dans les classes ni dans cet atelier
  - ... mais un objectif secondaire de cet atelier est une initiation à la programmation
- À propos des exemples et des exercices : j'ai essayé d'en varier la nature et les domaines pour donner des idées de choses à faire en classe, en suivant les recommandations officielles.
  - analyser le fonctionnement ou le but d'un algorithme existant (trace, algorithme mystère, cherchez l'erreur)
  - modifier un algorithme existant pour obtenir un résultat précis
  - créer un algorithme en réponse à un problème posé
- Document ressources Éduscol
- Concernant l'après-stage :
  - Ne pas hésiter à me contacter (`malika.more@iut.u-clermont1.fr`) en cas de difficultés ou de questions
  - De nombreux documents et informations sont disponibles à partir de la rubrique Évolutions au lycée du portail des IREM :

    `http ://www.univ-irem.fr/`

## Installation du logiciel

- Site officiel d'AlgoBox

  `http ://www.xm1math.net/algobox/`
- Libre gratuit et multiplateforme
- Écrit par un professeur de mathématiques de lycée pour l'algoritmique au lycée

# Table des matières

# 1 Introduction

Vous connaissez déjà tous les exemples que nous verrons aujourd'hui et tous ceux que vous serez amenés à enseigner au lycée. Il n'y a pas de contenu nouveau, il s'agit seulement d'un changement de présentation pour mettre en évidence la nature algorithmique des méthodes utilisées.

*Exemple.* Voici (encore) une version de l'algorithme d'Euclide :

**début**
  Lire le nombre $a$
  Lire le nombre $b$
  Donner à $r$ la valeur $a \mod b$
  **répéter jusqu'à** $r = 0$
    Donner à $a$ la valeur $b$
    Donner à $b$ la valeur $r$
    Donner à $r$ la valeur $a \mod b$
  **fin**
  Afficher $b$
**fin**

**Algorithme 1** : Euclide

Un algoritme est en général décomposable en trois parties :

- Un **pré-traitement** : entrée des données au clavier, initialisation des valeurs, ... (il faut bien commencer)
- Un **traitement** : calculs, manipulation des données, ... (c'est le cœur de l'algorithme)
- Un **post-traitement** : affichage des résultats sur l'écran, écriture dans un fichier, ... (sans cela les résultats seraient perdus)

Dans l'exemple, les 3 premières lignes constituent le pré-traitement, les 5 lignes suivantes le traitement proprement dit et la dernière ligne est le post-traitement.

*Exemple.* (suite) Voici un programme AlgoBox qui implémente l'algorithme d'Euclide :

```
1   VARIABLES
2     a EST_DU_TYPE NOMBRE
3     b EST_DU_TYPE NOMBRE
4     r EST_DU_TYPE NOMBRE
5   DEBUT_ALGORITHME
6     LIRE a
7     LIRE b
8     r PREND_LA_VALEUR a%b
9     TANT_QUE (r!=0) FAIRE
10      DEBUT_TANT_QUE
11      a PREND_LA_VALEUR b
12      b PREND_LA_VALEUR r
13      r PREND_LA_VALEUR a%b
14      FIN_TANT_QUE
15    AFFICHER "le pgcd est "
16    AFFICHER b
17  FIN_ALGORITHME
```

Raisons du choix d'AlgoBox pour cet atelier :

- Investissement de départ assez faible pour pouvoir commencer à programmer

- L'interface graphique limite les risques d'erreurs de syntaxe
- Langage dédié à l'algorithmique au lycée

Mais en contrepartie :

- Le langage est assez limité
- L'interface est sommaire

Moralité : Aucun langage de programmation n'est parfait pour tout, mais ils sont tous utilisables pour (presque) tout.

*Exemple.* Voici un algorithme très simple, dans lequel on remarque que la structure en trois partie est implicite :

**début**
  Lire le nombre $a$
  Afficher "L'image de" $a$ "est" $2a^2 - 5a + 3$
**fin**

**Algorithme 2** : Valeur d'une fonction (1)

On peut réécrire l'algorithme pour la faire apparaître :

**début**
  Lire le nombre $a$
  Donner à $b$ la valeur $2a^2 - 5a + 3$
  Afficher "L'image de" $a$ "est" $b$
**fin**

**Algorithme 3** : Valeur d'une fonction (2)

On écrit un programme en AlgoBox qui réalise ce calcul.

# 2 Les bases

Comme dans les exemples précédents, nos algorithmes seront écrits en "pseudo-code", c'est-à-dire sous une forme (relativement) standardisée et à l'aide d'un certain nombre de mots-clés (relativement) standardisés. Ceci permettra à la fois de faire apparaître les structures communes et de faciliter la programmation éventuelle en se rapprochant de la syntaxe d'un langage de programmation. Toutefois, il faut garder à l'esprit que le choix exact de cette forme et de ces mots-clés est en grande partie arbitraire, et peut donc légitimement varier.

## 2.1 Entrées/Sorties

On n'a besoin aujourd'hui que de deux instructions, mais rien n'empêcherait d'en inclure d'autres si le besoin s'en faisait sentir.

- Pour obtenir une donnée entrée au clavier : `Lire la valeur de a`
- Pour afficher un message et/ou un résultat à l'écran : `Afficher "Le résultat est :" b`

## 2.2 Variables, affectations et manipulation des données

**Variables et affectations.** Pour mémoriser les données initiales, ou les résultats intermédiaires des calculs, on utilise des "variables". Du point de vue de l'ordinateur, une variable est une zone de mémoire au contenu de laquelle on accède via un identificateur. Du point de vue algorithmique, une variable a un nom fixe et une valeur qui peut changer au cours de l'exécution de l'algorithme. La nature et le rôle des variables en informatique et en mathématique sont donc différents, bien qu'on utilise le même mot.

- Pour affecter une valeur à une variable : Donner à $a$ la valeur 12

*Important* : Quand on affecte une nouvelle valeur à une variable, la valeur précédente disparaît et n'est plus accessible.

*Remarque* : L'instruction "Lire la valeur de $a$" non seulement lit une valeur tapée au clavier, mais aussi affecte cette valeur à la variable $a$.

**Exercice 1.** (le jeu du bonneteau) Quel est le résultat affiché par l'algorithme ci-dessous :

**début**
| Donner à $a$ la valeur 1
| Donner à $b$ la valeur 2
| Donner à $c$ la valeur 3
| Donner à $c$ la valeur $a$
| Donner à $a$ la valeur $b$
| Donner à $b$ la valeur $c$
| Afficher $b$
**fin**

**Algorithme 4** : Bonneteau

**Manipulation des données.** Les instructions de manipulation des données (par exemple calcul) sont probablement les moins standardisées qui soient. En effet, elles dépendent fortement de la nature des données, et de la façons dont celles-ci sont organisées. On restera donc assez flous à ce sujet, et on verra au fur et à mesure de quoi on aura besoin et ce dont il sera raisonnable de se doter.

*Exemple.* Voici un algorithme qui effectue quelques calculs simples sur une donnée numérique :

**début**
| Lire le nombre $a$
| Donner à $b$ la valeur $a^2$
| Donner à $b$ la valuer $2b$
| Donner à $b$ la valeur $b - 5a$
| Donner à $b$ la valeur $b + 3$
| Afficher $b$
**fin**

**Algorithme 5** : Calculs

**Exercice 2.** (où l'on suit la "trace" de l'algorithme) Exécuter l'Algorithme 5 en prenant $a = 6$.

**Exercice 3.** (où l'on ne s'embête pas avec les instructions) Pour convertir des degrés Fahrenheit en degrés Celsius, on a la formule suivante :

$$C \approx 0.55556 \times (F - 32).$$

Écrire un algorithme qui convertit une temprérature entrée au clavier des degrés Fahrenheit en degrés Celsius, et affiche une valeur approchée à $10^{-1}$ près du résultat.

**Exercice 4.** (un meilleur algorithme ?)

1. Écrire un algorithme qui lit deux nombres $a$ et $b$ entrés au clavier et calcule $a^2 + 2ab + b^2$.
2. Écrire un autre algorithme qui donne le même résultat.

## 2.3 Structure alternative

Le plus souvent, un algorithme ne contient pas seulement des instructions de manipulation des données à éxécuter les une après les autres, mais aussi des instructions dites de contrôle ou de structure (conditions et boucles), qui ont un effet sur l'exécution des autres instructions.

Le premier type de telles instruction est celle permettant une exécution conditionnelle.

- Pour exécuter des instructions seulement dans le cas où une condition est réalisée :

```
si condition alors
|  instructions à effectuer
|  si la condition est réalisée
fin
```

- Pour exécuter certaines instructions dans le cas où une condition est réalisée et d'autres dans le cas où elle ne l'est pas :

```
si condition alors
|  instructions à effectuer
|  si la condition est réalisée
sinon
|  instructions à effectuer
|  si la condition n'est pas réalisée
fin
```

*Exemple.* L'algorithme ci-dessous a pour but de lire deux nombres au clavier, puis de les afficher dans l'ordre croissant.

```
début
|  Lire le nombre a
|  Lire le nombre b
|  si a<b alors
|  |  Afficher a
|  |  Afficher b
|  sinon
|  |  Afficher b
|  |  Afficher a
|  fin
fin
```

**Algorithme 6** : Maximum de deux

*Important* : La condition doit pouvoir être évaluée à vrai ou faux. Il s'agit donc d'une expression logique, plus ou moins compliquée.

**Exercice 5.** (modifier l'exemple précédent) Écrire un algorithme qui lit trois valeurs au clavier et affiche le maximum des trois.

**Exercice 6.** Soit une équation de droite $3x + 4y + 1 = 0$. Écrire un algorithme qui lit au clavier les coordonnées d'un point et étudie l'appartenance du point à la droite.

## 2.4 Structures répétitives

Il s'agit de répéter un bloc d'instructions plusieurs fois de suite. D'innombrables variantes sont possibles.
Les deux familles principales consistent à :

- répéter un bloc d'instructions un nombre de fois donné,
- répéter un bloc d'instructions jusqu'à ce qu'une condition soit vérifiée (ou tant qu'une condition est vérifiée).

Toutes les version sont légitimes quand on écrit un algorithme sur papier. Dans chaque langage de programmation certaines versions sont implémentées et pas d'autres, c'est pourquoi la connaissance a priori du langage dans lequel on entend programmer les algorithmes peut (mais non doit) orienter le choix pour l'écriture des algorithmes. Dans notre cas, j'ai choisi de coller à la syntaxe AlgoBox.

- Pour répéter un bloc d'instructions un nombre de fois donné :

```
pour i de 1 à 10 faire
| instructions à effectuer
fin
```

  - la variable $i$ est un compteur
  - elle augmente automatiquement de 1 à chaque tour
  - **pour** ... *de* ... *à* ... **faire**

*Exemple.* Calcul de 10 !

```
début
| Donner à res la valeur 1
| pour i de 1 à 10 faire
| | Donner à res la valeur res * i
| fin
| Afficher res
fin
```

**Algorithme 7** : Factorielle "Pour"

*Remarque* On peut (ou pas) utiliser la variable $i$ dans la boucle, mais il est préférable de ne pas changer sa valeur.

- Pour répéter un bloc d'instructions tant qu'une condition est vérifiée :

```
tant que condition faire
| instructions à effectuer
fin
```

  - Le test de la condition est effectué **avant** d'entrer dans la boucle.
  - Par conséquent, si la condition n'est pas vérifiée avant l'entrée dans la boucle, **on n'y entre pas**, les instructions à l'intérieur de la boucle ne sont pas effectuées, et on passe à l'instruction suivant la boucle.

*Exemple.* Un autre calcul de 10 !

```
début
    Donner à res la valeur 1
    Donner à i la valeur 1
    tant que i ≤ 10 faire
        Donner à res la valeur res ∗ i
        Donner à i la valeur i + 1
    fin
    Afficher res
fin
```

**Algorithme 8** : Factorielle "Tant que"

*Important* : D'un point de vue algorithmique, dans tous les cas, il est important de bien réfléchir à l'entrée et à la sortie de la boucle.

**Exercice 7.** (cherchez l'erreur) L'algorithme ci-dessous calcule 10 !

```
début
    Donner à res la valeur 1
    Donner à i la valeur 1
    tant que i ≤ 10 faire
        Donner à res la valeur res ∗ i
    fin
    Afficher res
fin
```

**Algorithme 9** : Factorielle fausse (1)

**Exercice 8.** (cherchez l'erreur) L'algorithme ci-dessous calcule 10 !

```
début
    Donner à res la valeur 1
    tant que i ≤ 10 faire
        Donner à res la valeur res ∗ i
        Donner à i la valeur i + 1
    fin
    Afficher res
fin
```

**Algorithme 10** : Factorielle fausse (2)

**Exercice 9.** (cherchez l'erreur) L'algorithme ci-dessous calcule 10 !

```
début
    Donner à res la valeur 1
    Donner à i la valeur 1
    tant que i ≤ 10 faire
        Donner à i la valeur 1
        Donner à res la valeur res ∗ i
        Donner à i la valeur i + 1
    fin
    Afficher res
fin
```

**Algorithme 11** : Factorielle fausse (3)

**Exercice 10.** (cherchez l'erreur) L'algorithme ci-dessous calcule 10 !

```
début
    Donner à res la valeur 1
    Donner à i la valeur 11
    tant que i ≤ 10 faire
        Donner à res la valeur res ∗ i
        Donner à i la valeur i + 1
    fin
    Afficher res
fin
```

**Algorithme 12** : Factorielle fausse (4)

**Exercice 11.** (Algorithme mystère) Que fait l'algorithme ci-dessous ?

```
début
    Donner à a la valeur 0    Donner à b la valeur 10
    Donner à N la valeur 50    Donner à pas la valeur (b − a)/N
    Donner à x la valeur a    Donner à max la valeur 2x^2 − 5x + 3
    pour i de 1 à N faire
        Donner à x la valeur x + pas
        Donner à y la valeur 2x^2 − 5x + 3
        si max < y alors
            Donner à max la valeur y
        fin
    fin
    Afficher max
fin
```

**Algorithme 13** : Inconnu

**Exercice 12.** (Algorithme mystère) Que fait l'algorithme ci-dessous ?

```
début
    Lire la valeur de N
    Donner à i la valeur 0
    Donner à n la valeur 1
    tant que n ≤ N faire
        Donner à n la valeur 2 ∗ n
        Donner à i la valeur i + 1
    fin
    Afficher i − 1
fin
```

**Algorithme 14** : Autre Inconnu

**Exercice 13.** (où l'on voit que le diable est dans les détails) Que se passe-t-il si on choisit $N = 0$ ? Comment réparer le problème ?

**Exercice 14.** Écrire un algorithme qui détermine si un nombre lu au clavier est premier.

# 3 Pour aller plus loin

## 3.1 Documentation des algorithmes

Quelques habitudes de rigueur permettent de se simplifier la vie quand les algorithmes deviennent longs et/ou nombreux :

- Décrire précisément l'objectif de l'algorithme
- Écrire des commentaires décrivant le déroulement de l'algorithme
- Choisir des noms de variables explicites
- Afficher des messages à l'écran pour dire à l'utilisateur du programme ce qu'il doit faire

*Exemple.* Voici une version commentée de l'algorithme d'Euclide :

```
début
    Afficher "Entrez le nombre a"
    Lire le nombre a                          % a et b sont les nombres %
    Afficher "Entrez le nombre b"             % dont on calcule le pgcd %
    Lire le nombre b
    Donner à r la valeur a mod b                        % r est le reste %
    répéter jusqu'à r = 0
        Donner à a la valeur b                % le pgcd de a est de b est %
        Donner à b la valeur r            % égal au pgcd de b et du reste %
        Donner à r la valeur a mod b                  % et on recommence %
    fin
    Afficher "Le pgcd est" b    % le dernier reste non nul est le pgcd %
fin
```

**Algorithme 15** : Euclide commenté

**Exercice 15.** (algorithme mystère)

1. Que fait l'algorithme 15 ?
2. Commenter cet algorithme.

```
début
    Donner à nb la valeur 1000
    Donner à somme la valeur 0
    répéter nb fois
        Donner à tir la valeur 1
        Donner à d une valeur entière aléatoire entre 1 et 6
        répéter jusqu'à d = 6
            Donner à tir la valeur tir + 1
            Donner à d une valeur entière aléatoire entre 1 et 6
        fin
        Donner à somme la valeur somme + tir
    fin
    Afficher somme/nb
fin
```

**Algorithme 16** : inconnu

## 3.2 Listes

On utilise des listes ou des tableaux pour manipuler des variables indicées. Les instructions associées permettent d'ajouter ou de supprimer des éléments, de connaitre la longueur d'une liste, etc. Selon les langages de programmation, la gestion des listes et des tableaux est très variable.

*Exemple.* Voici un algorithme qui détermine si un triangle dont les coordonnées des sommets $M_1$, $M_2$ et $M_3$ sont entrées au clavier est isocèle en $M_1$.

```
début
    Donner à i la valeur 1
    répéter 3 fois
        Lire la valeur de abs[i]                % liste des abscisses %
        Lire la valeur de ord[i]                % liste des ordonnées %
        Donner à i la valeur i + 1
    fin
    Donner à M1M2 la valeur (abs[2] − abs[1])² + (ord[2] − ord[1])²
    Donner à M1M3 la valeur (abs[3] − abs[1])² + (ord[3] − ord[1])²
    si M1M2 = M1M3 alors
        Afficher "C'est un triangle isocèle en M1"
    sinon
        Afficher "Ce n'est pas un triangle isocèle en M1"
    fin
fin
```

**Algorithme 17** : Triangle isocèle

**Exercice 16.** (fonctionnalités graphiques) Écrire un algorithme qui dessine un triangle à l'écran. Programmer cet algorithme avec AlgoBox

**Exercice 17.** (au hasard) Écrire un algorithme qui remplit une liste de 10 nombres entiers entre 0 et 9 au hasard, puis calcule la somme des éléments de la liste.

## 3.3 Exercices supplémentaires

**Exercice 18.** (encore une boucle) Écrire un algorithme qui recherche une solution approchée de l'équation $x^2 + x - 1$ sur l'intervalle $[0; 2]$ par dichotomie, avec une précision de $10^{-3}$.

**Exercice 19.** (les chaînes de caractères) Le code de César consiste à crypter un message en remplaçant chaque lettre par celle qui se trouve 3 rangs à droite dans l'alphabet (et bien sûr "x", "y" et "z" deviennent respectivement "a", "b" et "c"). Par exemple "exemple" devient "hahpsoh". Écrire un algorithme qui crypte un mot entré au clavier en utilisant le code de César.

**Exercice 20.** (un peu de tout) Écrire un algorithme de machine à voter : on fait voter des électeurs tant qu'il y en a entre un candidat A et un candidat B. À la fin, on affiche le nom du vainqueur (s'il y en a un). L'algorithme doit être fiable à 100% et ne doit pas permettre d'erreur de saisie.