IREM Clermont-Ferrand
Journée d'information
sur les nouveaux programmes
de Première ES et S

Année 2010-2011
Malika More

Initiation à l'algorithmique

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, Scilab,...
  - ... 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
- 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@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/`

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 $a \mod b$
  tant que $r \neq 0$ **faire**
    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)
- Souvent schématisé : **Entrées - Algorithme - Sortie**

Dans l'exemple, les 2 premières lignes constituent le pré-traitement, les 6 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 b
16  FIN_ALGORITHME
```

Voici un programme en Scilab qui implémente l'algorithme d'Euclide :

```
1 a=input("Entrer a : ")
2 b=input("Entrer b : ")
3 r=reste(a,b)
4 while r<>0
5   a=b
6   b=r
7   r=reste(a,b)
8 end
9 afficher(b)
```

Avant de commencer. 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 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 la souris par exemple).

- 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

3 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.

Une conséquence notable

Exemple. On souhaite échanger le contenu de deux variables

début
  Donner à a la valeur 1
  Donner à b la valeur 2
  Donner à b la valeur a
  Donner à a la valeur b
  Afficher a
  Afficher b
fin

Algorithme 2 : Échange faux

Et la solution habituelle :

début
  Donner à a la valeur 1
  Donner à b la valeur 2
  Donner à $temp$ la valeur b
  Donner à b la valeur a
  Donner à a la valeur $temp$
  Afficher a
  Afficher b
fin

Algorithme 3 : Échange

Exemple. (Où l'on suit la "trace" de l'algorithme)
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

Exemple. (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} prés du résultat.

début
  Lire le nombre F
  Donner à C la valeur $0.55556 \times (F - 32)$
  Arrondir C à 10^{-1}
  Afficher C
fin

Algorithme 6 : Conversion

Ce n'est pas fini ! Le plus souvent, un algorithme ne contient pas seulement des instructions de manipulation des données à exécuter 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. Sinon, un algorithme ne serait qu'une recette de cuisine.

4 Structure alternative

- 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 7 : 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. L'utilisation de conditions complexes nécessite des compétences en calcul booléen.

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

```
début
    ...
    si a > b et a > c alors
        Donner à max la valeur a
    sinon
        si b > c alors
            Donner à max la valeur b
        sinon
            Donner à max la valeur c
        fin
    fin
    Afficher max
fin
```

Algorithme 8 : Maximum de trois

5 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 la plus courante.

- 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**
  - **pour** ... *de* ... *à* ... *par pas de* ... **faire**
  - On peut (ou pas) utiliser la variable i dans la boucle, mais il est préférable de ne pas changer sa valeur

Exemple.

```
début
    Afficher "Je commence."
    pour i de 1 à 10 faire
        Afficher "Je fais un tour dans la boucle."
    fin
    Afficher "J'ai fini."
fin
```

Algorithme 9 : Tours dans la boucle

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 10 : Factorielle "Pour"

début
  Donner à res la valeur 1
  pour *i de 1 à 10* **faire**
    Donner à res la valeur $res * i$
    Donner à i la valeur 15
  fin
  Afficher res
fin

Algorithme 11 : Factorielle fausse (1)

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

Algorithme 12 : Factorielle fausse (2)

- 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 $i \leq 10$ **faire**
    Donner à res la valeur $res * i$
    Donner à i la valeur $i + 1$
  fin
  Afficher res
fin

Algorithme 13 : 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.

Exemple. (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 14 : Factorielle fausse (3)

Exemple. (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 15 : Factorielle fausse (4)

Exemple. (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 16 : Factorielle fausse (5)

Exemple. (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 17 : Factorielle fausse (6)

Exemple. (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 18 : Inconnu

Exemple. (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 ?

```
début
    Lire la valeur de N
    si N < 1 alors
        Afficher "Impossible"
    sinon
        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
fin
```

Algorithme 19 : Logarithme corrigé