

Formation Algorithmique STS SIO

Alex Esbelin

(Alex.Esbelin@math.univ-bpclermont.fr)

Malika More

(malika.more@u-clermont1.fr)

IREM Clermont-Ferrand

Stage du 29 Juin 2011

Qu'est-ce-qu'un algorithme ?

Un objet élaboré

« Un algorithme est une suite finie de règles à appliquer dans un ordre déterminé à un nombre fini de données pour arriver avec certitude (c'est-à-dire sans indétermination ni ambiguïté), en un nombre fini d'étapes, à un certain résultat, et cela indépendamment des données. Un algorithme ne résout donc pas un problème unique mais toute une classe de problèmes ne différant que par les données, mais gouvernés par les mêmes prescriptions. »

Bouvier A., George M., Le Lionnais F. (2005), *Dictionnaire des mathématiques*, P.U.F..

Commentaires

Trois notions essentielles

- Finitude
 - De l'algorithme : « ...une suite finie de règles... »
 - Des données : « ...un nombre fini de données... »
 - Du calcul : « ...un nombre fini d'étapes... »
- Certitude
 - Du résultat : « ...arriver avec certitude...à un certain résultat...sans indétermination ni ambiguïté... »
 - De l'arrêt : « ...un nombre fini d'étapes... »
- Généralité
 - « ...indépendamment des données...une classe de problèmes ne différant que par les données, mais gouvernés par les mêmes prescriptions... »

U22 : Algorithmique appliquée

Objectif : programmation

- Compétences et savoir-faire
 - Compréhension, interprétation
 - Conformité, documentation
 - Programmation, mise au point, validation
- Évaluation
 - Maîtrise pratique des mécanismes algorithmiques
 - Épreuve comportant de la programmation
- Méthodes
 - Formalisme imposé
 - Langage de programmation non imposé
- Thèmes très ouverts
 - Vie courante
 - Mathématiques
 - Informatique

Les bases de l'algorithmique

- 1 Entrées/Sorties
- 2 Variables, types et affectations
- 3 Manipulation des données
- 4 Structures conditionnelles
- 5 Structures itératives
- 6 Présentation des algorithmes

- 1 Entrées/Sorties
- 2 Variables, types et affectations
- 3 Manipulation des données
- 4 Structures conditionnelles
- 5 Structures itératives
- 6 Présentation des algorithmes

Instructions d'entrée/sortie

On ne présente que deux instructions, mais rien n'empêcherait d'en utiliser 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



Hello world !

Exemple

Algorithme 1 : Salutation

début

Afficher "Quel est votre nom ?"

Lire la valeur de *nom*

Afficher "Bonjour " *nom* " !"

fin

- 1 Entrées/Sorties
- 2 Variables, types et affectations**
- 3 Manipulation des données
- 4 Structures conditionnelles
- 5 Structures itératives
- 6 Présentation des algorithmes

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.

Instruction d'affectation

- Pour affecter une valeur à une variable :

Donner à a la valeur 12

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 .

Important

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

Une conséquence notable

Exercice (Cherchez l'erreur !)

On souhaite échanger le contenu de deux variables

Algorithme 2 : Échange faux

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

Et la solution habituelle

Solution

On souhaite échanger le contenu de deux variables

Algorithme 3 : Échange correct

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

Affectation récursive

Une différence importante avec les maths

- L'instruction `Donner à a la valeur $a + 1$` correspond à ajouter 1 à la valeur représentée par la variable a
- En Python, et dans de nombreux langages de programmation, cette instruction s'écrit `a=a+1`
- Dans ce cas, le signe `=` ne représente pas la relation d'égalité, mais l'instruction d'affectation

Type des variables

- Une variable doit être *typée*, au moins implicitement
- Principaux types élémentaires de variables :
 - Un entier
 - Un flottant (pour la vraisemblance informatique, on évitera de parler de *réels*)
 - Un texte (on dit souvent *une chaîne de caractères*)
 - Un booléen (notés par exemple `Vrai` et `Faux`)
 - Etc.
- En toute rigueur, on ne peut que recommander de définir explicitement le type des variables, même sur le papier

Type des variables

Exemples

- Donner à l'entier *a* la valeur 12
- Lire la valeur du texte *nom*
- Donner au booléen *test* la valeur `Vrai`

Typage dynamique (Python)

- Une variable prend dynamiquement le type des différents objets qu'on lui affecte.
- `a=12` : la variable *a* est du type entier
- `a=12.0` : la variable *a* est du type flottant

- 1 Entrées/Sorties
- 2 Variables, types et affectations
- 3 Manipulation des données**
- 4 Structures conditionnelles
- 5 Structures itératives
- 6 Présentation des algorithmes

Manipulation des données.

- Les instructions de manipulation des données (par exemple calcul) sont probablement les moins standardisées qui soient.
- En effet, elles dépendent fortement du type des données, et de la façon dont celles-ci sont organisées.
- En programmation, une même instruction peut donner des résultats différents selon le type des données.

Instructions de manipulation des données

Exemple

Voici un algorithme-calcul, qui effectue quelques calculs simples sur une donnée numérique :

Algorithme 4 : Calculs

début

Lire le flottant a
Donner à b la valeur a^2
Donner à b la valeur $2b$
Donner à b la valeur $b - 5a$
Donner à b la valeur $b/4$
Afficher b

fin

Suivre la trace de l'algorithme

Exercice

Exécuter l'Algorithme 4 en prenant $a = 6$.

Solution

début

Lire le flottant a

Donner à b la valeur a^2

Donner à b la valeur $2b$

Donner à b la valeur $b - 5a$

Donner à b la valeur $b/4$

Afficher b

fin

Suivre la trace de l'algorithme

Exercice

Exécuter l'Algorithme 4 en prenant $a = 6$.

Solution

début

Lire le flottant a

Donner à b la valeur a^2

Donner à b la valeur $2b$

Donner à b la valeur $b - 5a$

Donner à b la valeur $b/4$

Afficher b

fin

a	b
6	??
6	36
6	72
6	42
6	10.5

La valeur affichée
est 10.5

Attention au type en programmant

Que se passe-t-il dans ce cas en prenant $a = 6$?

début

Lire l'entier a

Donner à b la valeur a^2

Donner à b la valeur $2b$

Donner à b la valeur $b - 5a$

Donner à b la valeur $b/4$

Afficher b

fin

- Différents comportements sont possibles :
 - En Python3, on obtient 10.5 car le signe / représente la division en flottant.
 - La division en entiers s'écrit //, et dans ce cas, le résultat serait 10.

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.

- 1 Entrées/Sorties
- 2 Variables, types et affectations
- 3 Manipulation des données
- 4 Structures conditionnelles**
- 5 Structures itératives
- 6 Présentation des algorithmes

Instruction "Si... alors..."

- 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

Instruction "Si... alors... sinon..."

- 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

La machine à résultat

Exemple

Algorithme 5 : Résultat

début

Afficher "Saisissez votre note : "

Lire le nombre *note*

si *note* < 10 **alors**

| Afficher "Vous n'êtes pas admis(e)."

sinon

| Afficher "Vous êtes admis(e)."

fin

fin

Variation

Exercice

Modifier l'Algorithme 5 de façon à afficher une éventuelle mention.

Instructions conditionnelles imbriquées

Solution

début

...

si *note* < 10 **alors**

| Afficher "Vous n'êtes pas admis(e)."

sinon

| **si** *note* < 12 **alors**

| | Afficher "Vous êtes admis(e)."

| **sinon**

| | Afficher "Vous êtes admis(e) avec mention."

| **fin**

fin

fin

Combinaisons booléennes de conditions

Solution

début

...

si *note* < 10 **alors**

| Afficher "Vous n'êtes pas admis(e)."

fin

si *note* ≥ 10 **et** *note* < 12 **alors**

| Afficher "Vous êtes admis(e)."

fin

si *note* ≥ 12 **alors**

| Afficher "Vous êtes admis(e) avec mention."

fin

fin

Qu'est-ce-qu'une condition ?

Important

- Une condition doit pouvoir être évaluée à `Vrai` ou `Faux`.
- Il s'agit donc d'une **expression booléenne**, plus ou moins compliquée.
- Une **variable booléenne** est une variable qui ne peut prendre que les deux valeurs `Vrai` ou `Faux`.

Calcul booléen express

Connecteur booléens les plus courants

- **non** : négation, contraire
- **et** : conjonction
- **ou** : disjonction (toujours inclusive)
- **oux (xor)** : disjonction exclusive

Attention en programmant

Remarque

- Python et certains autres langages de programmation, mais pas tous, évaluent les **et** et les **ou** séquentiellement
- si le premier terme rend l'expression fausse (ou vraie), alors le second terme n'est pas évalué.
- Les connecteurs ne sont alors plus commutatifs, et il arrive qu'un test `A et B` qui provoque une erreur n'en provoque plus si on écrit `B et A` ...

Utilisation de variables booléennes

Exemple

Algorithme 6 : Résultat

début

Afficher "Saisissez votre note : "

Lire le nombre *note*

Donner au booléen *admis* la valeur $note \geq 10$

si *admis* alors

| Afficher "Vous êtes admis."

sinon

| Afficher "Vous n'êtes pas admis."

fin

fin

- 1 Entrées/Sorties
- 2 Variables, types et affectations
- 3 Manipulation des données
- 4 Structures conditionnelles
- 5 Structures itératives**
- 6 Présentation des algorithmes

Les boucles

C'est maintenant que l'affaire se complique...

Il s'agit de répéter un bloc d'instructions plusieurs fois de suite.
D'innombrables variantes sont possibles.

Les deux familles principales :

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

Les boucles

- Toutes les variantes 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.

Boucle `pour`

- 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 prend initialement la valeur 1 et augmente **automatiquement** de 1 à chaque tour.
- On sort de la boucle lorsque la valeur finale est dépassée.
- On peut (ou pas) utiliser la variable i dans la boucle, mais on ne doit **jamais** modifier sa valeur.
- **pour ... de ... à ... faire**
- **pour ... de ... à ... par pas de ... faire**

Boucle pour

Exemple

Algorithme 7 : Tours dans la boucle

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

Boucle pour

Exemple

Algorithme 8 : Factorielle "Pour"

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

Cherchez l'erreur !

Exercice

L'algorithme ci-dessous calcule 10 !

Algorithme 9 : Factorielle fausse

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

Remarque

Certains langages de programmation permettent malheureusement de modifier la valeur du compteur. Mais pas Python, ouf...

Boucle tant que

- 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.
- On sort de la boucle lorsque la condition n'est plus vérifiée.

Boucle tant que

Exemple

Algorithme 10 : Machine infernale

début

Donner au texte *rep* la valeur "non"

tant que *rep* ≠ "oui" **faire**

 Afficher "Voulez-vous un café ?"

 Lire la valeur de *rep*

fin

Afficher "Nous allons vous en préparer un."

fin

Boucle tant que

Exemple

Algorithme 11 : Factorielle "Tant que"

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

Pas si simple

Important

- La boucle `tant que` est beaucoup plus souple d'utilisation que la boucle `pour`
- Par conséquent, elle est aussi beaucoup plus difficile à utiliser
- Les possibilités d'erreur sont immenses
- 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.

Cherchez l'erreur

Exercice

L'algorithme ci-dessous calcule 10 !

Algorithme 12 : Factorielle fausse (2)

début

Donner à *res* la valeur 1

Donner à *i* la valeur 1

tant que $i \leq 10$ **faire**

 Donner à *res* la valeur $res * i$

fin

Afficher *res*

fin

Cherchez l'erreur

Exercice

L'algorithme ci-dessous calcule 10 !

Algorithme 13 : Factorielle fausse (3)

début

Donner à *res* 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

Cherchez l'erreur

Exercice

L'algorithme ci-dessous calcule 10 !

Algorithme 14 : Factorielle fausse (4)

début

Donner à *res* la valeur 1

Donner à *i* la valeur 1

tant que $i \leq 10$ **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

Cherchez l'erreur

Exercice

L'algorithme ci-dessous calcule 10 !

Algorithme 15 : Factorielle fausse (5)

début

Donner à *res* la valeur 1

Donner à *i* la valeur 11

tant que $i \leq 10$ **faire**

 Donner à *res* la valeur $res * i$

 Donner à *i* la valeur $i + 1$

fin

Afficher *res*

fin

Exercice

Montrer que toute boucle `pour` peut être remplacée par une boucle `tant que` équivalente.

Exercice

Montrer que toute boucle `pour` peut être remplacée par une boucle `tant que` équivalente.

Solution

```
début
  ...
  pour i de a à b par pas de c faire
    fin
  ...
fin
```

```
début
  ...
  Donner à i la valeur a
  tant que i <= b faire
    Donner à i la valeur i + c
  fin
  ...
fin
```

- 1 Entrées/Sorties
- 2 Variables, types et affectations
- 3 Manipulation des données
- 4 Structures conditionnelles
- 5 Structures itératives
- 6 Présentation des algorithmes**

Préconisations précises

Présentation textuelle

- Indentation
- Indicateurs explicites de début et de fin de blocs

En-tête

- Nom
- Rôle
- Entrées et Sorties
- Déclaration typée des variables locales

Résultat

Fonction `Euclide(a,b : entiers)`

Entrée : Deux entiers a et b

Résultat : Le *PGCD* de a et b

Description : Algorithme d'Euclide

variables locales : $x, y, temp$: entiers

début

Donner à x la valeur a % On initialise x à a %

Donner à y la valeur b % On initialise y à b %

% Début du calcul %

tant que $y > 0$ **faire**

 Donner à $temp$ la valeur y % On stocke y %

 Donner à y la valeur $x \bmod y$ % On remplace y par le reste %

 Donner à x la valeur $temp$ % x prend la valeur de y %

fin

% Fin du calcul : À la sortie $y = 0$ donc $x = \text{pgcd}(a, b)$ %

retourner : x

fin

FIN