FICHE ALGORITHME ET PROGRAMMATION

BOOSTMACOM - Théo Gamory

1. Qu'est-ce qu'un Algorithme ?

C'est une séquence d'étapes à effectuer pour une sortie requise d'une certaine entrée donnée. Son objectif est d'obtenir un résultat une fois que l'ensemble du processus est terminé.

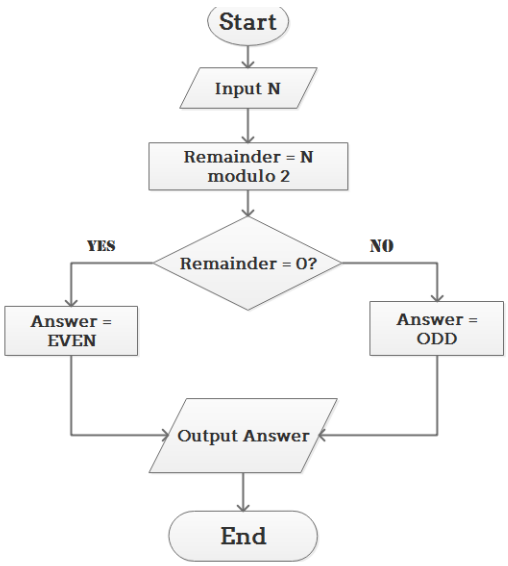
2. Domaines d'application

- Dans la vie quotidienne avec les recettes de cuisines ou les distributeurs automatique.
- En mathématique pour résoudre des problèmes.
- En informatique pour donner des instructions à un ordinateur (un programme)

3. Représentation

Le logigramme (ordinogramme / organigramme) : outil qui permet de visualiser de façon séquentielle et logique les actions à mener et les décisions à prendre pour atteindre un objectif défini.

Exemple : pair ou impair ?



4. La programmation informatique

C'est la mise en application de l'algorithmie dans le but de communiquer avec une machine (un ordinateur)

Un ordinateur ne comprend que des instructions **binaires** (ensemble de 0 et de 1), c'est ce qu'on appelle **le langage machine**.

Afin de communiquer plus facilement avec les ordinateurs on utilise des langages de programmation. Ils servent d'interprète et permettent de traduire des instructions claires en langage machine.

Exemple de langages de programmation : PHP, C#, Java, C, C++, Python, Javascript, etc...

ATTENTION : HTML et CSS ne sont en aucun cas des langages de programmation.

5. Écrire un Algorithme

5.1 Les variables

Pour mémoriser/lire des valeurs avec un algo on a besoin de **variables**. Celles-ci doivent être identifiées avec un nom qui ne doit pas contenir d'espace ni d'accents. Un nombre de variable ne commence jamais par un chiffre.

Affectation : c'est le fait d'attribuer un contenu à une variable

Exemple : N ← 5

Pour tester un algorithme vous pouvez l'évaluer à la main sur papier :

Début
A ← 2
B ← 5
C ← 12
D ← A + B
C ← A + C
B ← B + C
A ← C − A
D ← B × D − C
Fin

instructions	A	B	C	D
début	n.a.	n.a.	n.a.	n.a.
A ← 2	2	n.a.	n.a.	n.a.
B ← 5	2	5	n.a.	n.a.
C ← 12	2	5	12	n.a.
D ← A + B	2	5	12	7
C ← A + C	2	5	14	7
B ← B + C	2	19	14	7
A ← C − A	12	19	14	7
D ← B × D − C	12	19	14	119
fin	12	19	14	119

Le but ici est de suivre l'évolution des valeurs de nos variables. Chaque ligne représente le contenu de chaque variable à chaque ligne de l'algorithme (instruction).

5.2 Les types

Il existe **4 types** de variables : Entier, Réel, Chaîne de caractères, Booléen; c'est ce qu'on appelle le **Typage.**

- Entiers : tous les nombre "rond" (sans virgule) négatif ou positif
- Réels : les nombres à virgules
- Chaîne de caractères : "entre guillemets"
- Booléens : représente que deux valeurs possibles (Vrai / Faux) (1 / 0) (Oui / Non)

Avant leur utilisation, les variables doivent toujours être initialisées ! (par une affectation ou une saisie)

5.3 Lecture / Ecriture

But ici est de communiquer avec l'utilisateur (celui qui utilise le programme) soit en lui demandant des informations à saisir (Lecture) soit en lui en affichant (Ecriture).

Pour cela 2 méthodes :

- **saisir() / saisir("message")** : Le programme se met en attente d'une entrée pour pouvoir continuer et stock cette entrée dans une variable
  Exemple : N <-- saisir("Veuillez saisir un nombre")
  Note : on peut également afficher un message à l'utilisateur pour ajouter des précisions sur les informations demandées.

- **afficher("message")** : le programme affiche un message ou le contenu d'une variable
  Note : on peut afficher plusieurs valeurs en les séparant par des virgules.

5.4 Schéma d'écriture

- Un nom // même contraintes que pour les variables
- Son rôle // que fait l'algorithme ?
- Ses entrées / sorties
- La déclaration des variables utilisées (noms et types)
- Un Début et une Fin // encadre les instructions
- Des commentaires // signalés par le symbole "#" pour rendre lisible et compréhensible l'algorithme

Exemple :

Algorithme: Affiche_Double
Rôle: calculer le double d'un entier donné
Entrées: Un nombre entier A
Sortie: affichage de 2 x A
Variables: entier : A
Début :
    A ← saisir("Donnez-moi un nombre entier : ")
    Afficher("le double de votre nombre vaut ",2 x A)
Fin

5.5 Les opérateurs

- Sur les variables réelles :
  - Les opérations classiques : +, -, ×, ÷, a^x
  - Avec respect des priorités usuelles : puissance avant multiplications/divisions, multiplications/divisions évaluées avant les additions/soustractions
- Sur les variables entières :
  - Addition, soustraction et multiplication OK

- Cas particulier pour la division :
    - On utilisera *div* pour obtenir le quotient
    - Et *mod* pour obtenir le reste (division euclidienne)

      Exemple : (17 div 3) renvoie 5 et (17 mod 3) fournit 2

- Sur les chaines de caractères : une seule opération possible
    - La concaténation avec "+"

      Exemple :

      A ← "Salut, "
      B ← A + "tu vas bien ?"
      B contiendra : "Salut, tu vas bien ?"

Et sur les booléens : les opérateurs logiques **ET / OU**

Table de vérité de ET		
a	**b**	**a ET b**
0	0	0
0	1	0
1	0	0
1	1	1

Table de vérité de OU		
a	**b**	**a OU b**
0	0	0
0	1	1
1	0	1
1	1	1

L'opérateur **NON** : donne le "contraire" d'un booléen

Exemple :

A ← true
Non_A ← NON(A)
Afficher(Non_A) # affiche 'false'

6. Structures de contrôle

6.1 Traitements conditionnels

Il existe 2 types de traitements conditionnels :

- Traitement simple : Si (*condition*) Alors (*action*) FinSi
  Dans le cas où la condition n'est pas réalisée, l'action est ignorée et on passe directement à l'instruction suivant le Fin Si.

- Traitement étendue : Si (*condition*) Alors (*action*) Sinon (action) FinSi
  On exécute une autre action si la condition n'est pas vérifiée, ensuite on passe aux instructions suivantes.

Dans les deux cas on effectue la première action si et seulement si la première condition est vérifiée (si elle est vraie).

Exemple traitement simple :

```
N ← saisir("Saisir un entier")
Reminder ← N mod 2                    # Résultat de la division Euclidienne
Res ← "Le nombre est impair"
Si (Reminder = 0) Alors
        Res ← "Le nombre est paire"
Fin Si
Afficher(Res)
```

Exemple traitement étendue :

```
N ← saisir("Saisir un entier")
Reminder ← N mod 2                    # Résultat de la division Euclidienne
Si (Reminder = 0) Alors
        Afficher("Le nombre est paire")
Sinon
        Afficher("Le nombre est impair")
Fin Si
```

Dans cet exemple on préfèrera la première méthode (simple) car moins d'instructions.

6.2 Les boucles

Deux types de boucle :

- Tant que (while) : on exécute une instruction Tant que la condition évaluée est **vraie !**
  Lorsque la condition est fausse (n'est pas vérifiée) alors on passe directement aux instructions suivantes.

  IMPORTANT : les variables utilisées dans la condition doivent évoluer au risque de tomber sur une boucle infinie.

  Exemple :

  ```
  N ← 0
  Tant que (N < 10)   # N vaut toujours 0
          action …
  Fin Tant que
  ```

Ce qu'il faut faire :
N ← 0
Tant que (N < 10)
    action …
    N ← N + 1   # on incrémente N dans le but de rendre la condition fausse
Fin Tant que   et donc de sortir de la boucle

- Pour (for) : on exécute une action un nombre de fois défini
  Structure:
  Pour *VariableComptage* allant de *ValeurDébut* à *ValeurFin*
      action …
  Fin Pour

  1. La boucle "Pour" initialise la variable de comptage
  2. Elle s'occupe de l'incrémenter automatiquement (+ 1 à chaque fois)
  3. Jusqu'à ce que VariableComptage = ValeurFin (la boucle s'arrête à ce moment-là)

7. Les tableaux

7.1 Qu'est-ce à dire que ceci ?

Dans un programme informatique, les tableaux servent à stocker plusieurs valeurs dans un même conteneur (variable).

On peut les comparer à une commode faite de plusieurs tirroirs, dans lesquelles ont peut stocker des valeurs.

7.2 Les tableaux simple (1 dimensions)

On les représente comme ceci (exemple en python) :

```
tableau = [1, 2, 3, 4] # un tableau de 4 éléments
```

A savoir :

- La taille d'un tableau est le nombre d'éléments qu'il contient

- Un tableau commence toujours à l'indice 0, càd qu'ici la valeur 1 se situe à l'indice 0, 2 à l'indice 1, 3 à l'indice 2, etc...

- Pour obtenir la valeur d'un tableau à un certain indice on utilise cette syntaxe : val = tableau[0] (donc val vaut 1 dans notre exemple)

- Le dernier indice d'un tableau est toujours égale à la taille du tableau - 1

7.3 Les tableaux à double dimensions

Un tableau peut également contenir des tableaux, c'est un tableau à double dimension.

Exemple :

tableau = [[1, 2, 3], [4, 5, 6]]

Ici la variable "tableau" est un tableau qui contient à l'indice 0 un tableau [1, 2, 3] et à l'indice 1 un tableau [4, 5, 6].

Chaque tableau est indépendant au niveau de ces indices, càd qu'ici les deux tableaux commencent chacun à l'indice 0.

- tableau[0] vaut [1, 2, 3]

- tableau[1] vaut [4, 5, 6]

- tableau[0][2] vaut 3

- tableau[1][1] vaut 5

7.4 Généralités

- Un tableau est extensible (ce qu'on appellera une liste en python) : on peut toujours ajouter un élément à la suite d'un tableau.

Exemple en python :

```
tab = [1,5]
tab = tab + [3] # on concatène deux tableaux ici (on ajoute une nouvelle « cellule »
print(tab) # affiche [1,5,3]
```

- On peut modifier la valeur d'un tableau à un indice donné :

Exemple en python :

```
tab = [1, 5, 3]
tab[1] = 2 # on remplace la valeur à l'indice 1 par 2
print(tab) # affiche [1, 2, 3]
```