Aperçu sur les fonctions

- **Citation :**

« L'informatique semble encore chercher la recette miracle qui permettra aux gens d'écrire des programmes corrects sans avoir à réfléchir. Au lieu de cela, nous devons apprendre aux gens comment réfléchir » - Anonyme

- **Extrait (http://lwh.free.fr/pages/algo/algorithmes.htm)**

L'algorithmique est à la base de l'informatique. Quoiqu'on en dise, tous ces machins à la mode (POO, XML, Java et autres délires acronymiques dont on nous rabâche les oreilles) ne sont venus que bien plus tard, quand la route était déjà tracée et que - déjà - il n'y avait plus que des broutilles à inventer.

En effet, pour faire accomplir quelque chose d'à peu près utile au tas de ferraille que vous avez devant vous, il est inutile de compter sur son esprit d'initiative. Au contraire, il faut tout lui expliquer correctement. Et en détail. Patiemment. Car, malgré les apparences, l'ordinateur le plus sophistiqué n'a pas beaucoup plus d'intelligence qu'une ancienne machine à coudre. Et à pédales encore !

J'exagère à peine. Mais pour moi, **l'algorithmique** - c'est à dire **l'art de découper un problème complexe en tâches élémentaires** - est **la seule chose vraiment passionnante dans l'informatique**. Cette notion d'algorithme existait bien avant l'apparition du premier dinosaure informatique. Elle est profondément enracinée dans le désir humain de transmettre des méthodes efficaces pour résoudre des problèmes. Il s'agissait, par exemple, de procédures juridiques ou mathématiques chez les anciens Grecs. De règles linguistiques chez les Romains. Et, dans toutes les civilisations, de recettes divinatoires, médicales, culinaires...

- **Lisibilité d'un programme :**

Or, le fait d'être facilement modifiable donc lisible, y compris - et surtout - par ceux qui ne l'ont pas écrit est un critère essentiel pour un programme informatique ! Dès que l'on programme non pour soi-même, mais dans le cadre d'une organisation (entreprise ou autre), cette nécessité se fait sentir de manière aiguë. L'ignorer, c'est donc forcément grave.

- **Autres appellations**

Sous-programmes

Sous-algorithmes

Fonctions et procédures

- **Analogie**

Programme sans fonctions : une agora

Programme avec fonctions : une villa

1. Définition

Une fonction est un bloc d'instructions étiqueté ou nommé (ie auquel on a donné un nom) qui accomplit une tâche spécifique et partielle devant contribuer à l'atteinte d'un objectif général. Le nom sert à l'invoquer partout où besoin est dans le module.

Les fonctions reçoivent généralement des données en entrée et retournent généralement en sortie le résultat du traitement opéré.

Une procédure est une fonction accomplissant une tâche spécifique mais ne retournant aucun résultat.

Dans une maison : rasoir électrique (procédure) et une cafétière (fonction)

2. Fonctions préfinies

Tout langage de programmation propose ainsi un certain nombre de **fonctions** ; certaines sont indispensables, car elles permettent d'effectuer des traitements qui seraient sans elles impossibles. D'autres servent à soulager le programmeur, en lui épargnant de longs – et pénibles - algorithmes.

Partie entière

après : A ← **Ent**(3,228) A vaut 3

Modulo

A ← **Mod**(10,3) A vaut 1 car 10 = 3*3 + 1
B ← Mod(12,2) B vaut 0 car 12 = 6*2
C ← Mod(44,8) C vaut 4 car 44 = 5*8 + 4

Exemples bibliothèque stdio.h : printf, scanf

Exemples bibliothèque stdlib.h : system

Exemples bibliothèque math.h : sqrt, pow, sin, cos

A ← sin(35)

Une fonction est donc constituée de trois parties :

- le **nom** proprement dit de la fonction. Ce nom ne s'invente pas ! Il doit impérativement correspondre à une fonction proposée par le langage. Dans notre exemple, ce nom est SIN.
- deux parenthèses, une ouvrante, une fermante. Ces parenthèses sont toujours **obligatoires**, même lorsqu'on n'écrit rien à l'intérieur.

  Ces parenthèses comprennent parfois une liste de valeurs, indispensables à la bonne exécution de la fonction. Ces valeurs s'appellent des **arguments**, ou des **paramètres**. Les arguments doivent être d'un certain **type**, et qu'il faut respecter.

  Paramètres en entrée et paramètres en sortie
- le type de retour (appel et affectation : à droite et non à gauche)

3. Fonctions personnalisées

- Besoins
    - Lisibilité
    - Factorisation (éviter les répétitions inutiles)
    - Maintenance (faciliter les modifications)
    - Modularité (application = plusieurs fonctions avec plusieurs regroupements)
    - Performance

- Analyse fonctionnelle
    - Diviser pour régner
    - Découpage en modules

- Les trois moments d'une fonction
    - Déclaration
    - Définition
    - Appel

3.1 Exemples pratiques (cours)

- divisionAffichage et divisionAffichageFn (2020-2021)
- distance_1 et distance_2 (2024-2025)

3.2 Pratique en algorithmique (BALKANSKI volume 2)

- p. 21
-

3.3 Pratique en programmation (FABER)

- p. 13
- p. 34
- p. 207

4. Fonctions récursives

Une fonction récursive est une fonction qui s'appelle elle-même. Chaque appel à la fonction est indépendant des autres, avec ses propres variables. L'exemple le plus classique d'emploi de la récursivité est l'écriture de la fonction factorielle.