Introduction.

1. ALGORITHMES.

1.1. Notion d'algorithme.

a-*Définition*.

Certains voient, à tort, dans l'ordinateur une machine pensante et intelligente, capable de résoudre bien des problèmes. En fait, celui-ci ne serait capable de rien si quelqu'un (le programmeur en l'occurence) ne lui avait fourni la liste des actions à exécuter. Cette description doit être faite de manière non ambigüe car il ne faut pas s'attendre à la moindre interprétation des ordres fournis. Ils seront exécutés de manière purement mécanique.

De plus, les opérations élémentaires que peut exécuter un ordinateur sont en nombre restreint et doivent être communiquées de façon précise dans un langage qu'il comprendra. Le problème principal de l'utilisateur est donc de lui décrire la suite des actions élémentaires permettant d'obtenir, à partir des données fournies, les résultats escomptés. Cette description doit être précise, envisager le moindre détail et prévoir les diverses possibilités de données.

Cette marche à suivre porte le nom d'algorithme dont l'*Encyclopaedia Universalis* donne la définition suivante:

" Un algorithme est une suite finie de règles à appliquer dans un ordre déterminé à un nombre fini de données pour arriver, en un nombre fini d'étapes, à un certain résultat, et cela indépendamment des données. "

Le mot algorithme provient du nom d'un célèbre mathématicien arabe de la première moitié du IXe siècle: Muhammad ibn Musa al Khwarizmi.

Le rôle de l'algorithme est fondamental. En effet, sans algorithme, il n'y aurait pas de programme (qui n'est jamais que sa traduction dans un langage compréhensible par l'ordinateur). De plus, les algorithmes sont fondamentaux en un autre sens: ils sont indépendants à la fois de l'ordinateur qui les exécute, des langages dans lequel ils sont énoncés et traduits.

b-*Exemple*.

Calcul de l'intérêt et de la valeur acquise par une somme placée pendant un an à intérêt simple.

L'énoncé du problème indique

les données fournies: deux nombres représentant les valeurs de la somme placée et du taux d'intérêt

les résultats désirés: deux nombres représentant l'intérêt fourni par la somme placée ainsi que la valeur obtenue après placement d'un an.

Il nous faut maintenant décrire les différentes étapes permettant de passer des donnés aux résultats. Nos connaissances générales nous permettent d'exprimer cette règle:

"Pour obtenir l'intérêt fourni par la somme, il suffit de multiplier la somme par le taux d'intérêt divisé par cent; la valeur acquise s'obtient en additionnant ce dernier montant et la somme initiale."

c-*Méthodologie*.

Dans cet exemple simple apparaissent les trois étapes qui caractérisent la résolution d'un problème sur ordinateur:

comprendre la nature du problème posé et préciser les **données** fournies ("entrées" ou "**input**" en anglais)

préciser les résultats que l'on désire obtenir ("sorties" ou "**output**" en anglais)

déterminer le processus de transformation des données en résultats.

Ces trois étapes ne sont pas indépendantes et leur ordre peut être modifié.

Si les résultats fournis par l'ordinateur ne sont pas corrects, c'est qu'une erreur s'est glissée soit dans l'analyse du problème, soit dans la mise au point de l'algorithme, soit dans sa traduction en langage de programmation car l'ordinateur ne fait qu'exécuter scrupuleusement les opérations demandées.

1.2. Formalisation de l'algorithme.

a- *En français* .

L'exemple décrit ci-dessus deviendrait:

(1) prendre connaissance de la somme initiale et du taux d'intérêt

(2) multiplier la somme par le taux; diviser ce produit par 100; le quotient obtenu est l'intérêt de la somme

(3) additionner ce montant et la somme initiale; cette somme est la valeur acquise

(4) afficher les valeurs de l'intérêt et de la valeur acquise.

Il est évident, même sur cet exemple simple, qu'une telle formalisation risque de produire un texte long, difficile à comprendre et ne mettant pas clairement en évidence les différentes étapes du traitement.

b-*Langage de description.*

Dans un langage de description, les actions sont généralement décrites par un symbole ou un verbe à l'infinitif choisi pour éviter les confusions. Ce langage est appelé soit pseudocode soit langage de description d'algorithme (LDA).

Notre exemple devient:

écrire " Introduisez la somme initiale (en francs): "

lire somme_initiale

écrire " Introduisez le taux d'intérêt (ex: 3 pour 3%): "

lire taux

intérêt <-- somme_initiale * taux / 100

valeur_acquise <-- somme_initiale + intérêt

écrire " L'intérêt fourni est de " , intérêt , "francs "

écrire " La somme après un an sera de " , valeur_acquise , "francs "

Nous pouvons remarquer deux verbes particuliers:

lire qui correspond à la saisie, à l'introduction des données;

écrire qui exécute l'affichage à l'écran ou l'impression des résultats.

Ces verbes sont soulignés pour indiquer qu'ils ont un sens particulier, qu'il est interdit de les utiliser dans un autre sens et qu'il seront traduits pour être rendus compréhensibles par la machine.

Les valeurs manipulées dans cet algorithme sont des **constantes** (100) et des **variables** (somme_initiale, taux, intérêt, valeur_acquise). Il est pratique de choisir le nom des variables de manière à rappeler la signification de la valeur qu'elles représentent. Ce nom est souvent appelé identificateur de la variable. Les variables jouent le rôle de "**tiroirs**" dans lesquels on place une valeur durant l'exécution de l'algorithme. Ainsi,

lire somme_initiale

signifie que l'on introduit dans le tiroir baptisé *somme_initiale* la valeur numérique entrée au clavier lors de l'exécution du programme.

Le contenu d'un de ces tiroirs peut être modifié en y plaçant le résultat d'un calcul. Cette instruction porte le nom d'**assignation** ou **affectation** et se représente par une flèche (<--).

Ainsi,

intérêt <-- somme_initiale * taux /100

signifie que l'on place dans le tiroir *intérêt* le résultat de l'opération figurant à droite de la flèche. Cette instruction se lit: assigner à la variable *intérêt* la valeur de l'expression de droite.

Les expressions symbolisant les calculs à effectuer sont représentées par des formules algébriques faisant intervenir les noms des variables, des symboles mathématiques ("+" pour l'addition, "-" pour la soustraction, "*" pour la multiplication, "/" pour la division, ...) et des constantes numériques.

La description d'une action et des objets qui y participent porte le nom d'**instruction**. L'ordre dans lequel les différentes opérations seront écrites indique l'ordre dans lequel elles seront exécutées: de haut en bas et de droite à gauche. Il s'agit d'une **exécution séquentielle**.

1.3. Qualités d'un algorithme, d'un programme.

Tout programme fourni à l'ordinateur n'est que la traduction dans un langage de programmation d'un algorithme mis au point pour résoudre un problème donné. Pour obtenir un bon programme, il faut partir d'un bon algorithme. Il doit, entre autres, posséder les **qualités** suivantes:

être **clair**, facile à comprendre par tous ceux qui le lisent (**structure** et **documentation**)

présenter la plus **grande généralité** possible pour répondre au plus grand nombre de cas possibles

être d'une **utilisation aisée** même par ceux qui ne l'ont pas écrit et ce grâce aux messages apparaissant à l'écran qui indiqueront quelles sont les données à fournir et sous quelle forme elles doivent être introduites ainsi que les différentes actions attendues de la part de l'utilisateur

être conçu de manière à limiter le nombre d'opérations à effectuer et la place occupée en mémoire.

Une des meilleures façons de rendre un algorithme clair et compréhensible est d'utiliser une programmation structurée n'utilisant qu'un petit nombre de structures indépendantes du langage de programmation utilisé. Une technique d'élaboration d'un bon algorithme est appelée **méthode descendante** (**top down**). Elle consiste à considérer un problème dans son ensemble, à préciser les données fournies et les résultats à obtenir puis à **décomposer** le problème en plusieurs sous-problèmes plus simples qui seront traités séparément et éventuellement décomposés eux-mêmes de manière plus fine.

Exemple: imaginons un robot domestique à qui nous devons fournir un algorithme lui permettant de préparer une tasse de café soluble. Une première version de l'algorithme pourrait être:

(1) faire bouillir de l'eau

(2) mettre le café dans la tasse

(3) ajouter l'eau dans la tasse

Les étapes de cet algorithme ne sont probablement pas assez détaillées pour que le robot puisse les interpréter. Chaque étape doit donc être affinée en une suite d'étapes plus élémentaires, chacune étant spécifiée d'une manière plus détaillée que dans la première version. Ainsi l'étape

(1) faire bouillir l'eau peut être affinée en

(1.1) remplir la bouilloire d'eau

(1.2) brancher la bouilloire sur le secteur

(1.3) attendre l'ébullition

(1.4) débrancher la bouilloire

De même,

(2) mettre le café dans la tasse pourrait être affiné en

(2.1) ouvrir le pot à café

(2.2) prendre une cuiller à café

(2.3) plonger la cuiller dans le pot

(2.4) verser le contenu de la cuiller dans la tasse

(2.5) fermer le pot à café

et (3) ajouter de l'eau dans la tasse pourrait être affinée en

(3.1) verser de l'eau dans la tasse jusqu'à ce que celle-ci soit pleine

Certaines étapes étant encore trop complexes et sans doute incompréhensibles pour notre robot, il faut les affiner davantage. Ainsi l'étape

(1.1) remplir la bouilloire d'eau

peut nécessiter les affinements suivants:

(1.1.1) mettre la bouilloire sous le robinet

(1.1.2) ouvrir le robinet

(1.1.3) attendre que la bouilloire soit pleine

(1.1.4) fermer le robinet

Quand il procède à des affinements des différentes étapes, le concepteur d'un algorithme doit naturellement savoir où s'arrêter. Autrement dit, il doit savoir quand une étape constitue une primitive adéquate au point de ne pas avoir besoin d'affinement supplémentaire. Cela signifie évidemment qu'il doit connaître quelle sorte d'étape le processeur peut interpréter. Par exemple, le concepteur de l'algorithme précédent doit savoir que le robot peut interpréter "brancher la bouilloire" ce qui de ce fait n'exige pas d'affinement, mais qu'en revanche, il ne peut pas interpréter "remplir la bouilloire" et que dès lors un affinement devient nécessaire.

Page précédente.