INTRODUCTION METHODOLOGIQUE A L' ALGORITHMIQUE

I. DEFINITIONS PRELIMINAIRES

I.1. Action

Une *action* est un évènement produit par un exécutant (acteur). L'évènement prend place pendant une période de temps finie et produit un résultat bien défini et prévu.

I.2. Information

Une action aura un début (temps t_0) et une fin (temps t_1). Pour pouvoir reconnaître le résultat d'une action, il faut que le système observé pendant ce laps de temps, soit pourvu d'indicateurs prenant des valeurs différentes. La valeur d'un tel indicateur est appelée *information*.

I.3. Etat

L'ensemble des valeurs des différents indicateurs en un instant t ($t_0 \leq t \leq t_1$) du déroulement de l'évènement est appelée *état* du système observé à l'instant t.

I.4. Résultat

Le *résultat* bien défini est l'état du système à l'instant t_1.
Le fait que le résultat soit prévisible signifie que si l'on connaît l'état du système à l'instant t_0, alors on peut prédire avec précision quel sera l'état du système à l'instant t_1, avant même que l'action envisagée ne se produise.
L'état du système à l'instant t_0 définit les données de l'action.

I.5. Processus

Un *processus* est un ensemble de phénomènes organisé dans le temps et conçu comme actif.

Découvrir un processus, dans ce contexte informatique, c'est interpréter un évènement comme un ensemble d'actions plus élémentaires dont l'effet cumulatif est le même que celui produit par l'évènement complet. Si une action de ce processus ne peut commencer avant que l'action en cours ne soit totalement terminée, alors le processus est dit séquentiel. Sinon il est dit parallèle.

Dans ce cours, nous nous intéressons exclusivement aux processus séquentiels. Ainsi, à tout instant, une action et une seule pourra s'exécuter.

Comparer deux processus revient à chercher les différences sur les actions mises en jeu. On perçoit alors l'activité de programmation qui a pour objectif de décrire ces divers processus en un seul texte. La notation algorithmique permet le nombre d'exécution d'une action en fonction des informations observées (ou objectifs). L'état initial du système fixe ainsi pour chaque action du processus le nombre de ses exécutions. La notion de processus est donc essentielle pour la mise en œuvre des algorithmes. En effet, elle va permettre d'organiser le travail de programmation (description de plusieurs évènements en un seul texte) sur la base de la découverte des actions élémentaires qu'il faut organiser dans le temps (décomposition du problème) et du choix de la manière d'organiser ces actions (composition des actions) pour obtenir le résultat voulu par effet cumulé.

I.6. Schéma

C'est une description ou une représentation mentale réduite aux traits essentiels d'un objet ou d'un processus.

Il est fondamentale pour un programmeur de pouvoir déceler dans un seul d'évènements ou d'informations, les pus grandes lignes communes. En effet, lors de la conception des solutions informatiques, on est amené à faire abstraction à un certain niveau, d'un ensemble de détails semblant différencier deux évènements, pour pouvoir trouver comment ces détails se ressemblent. Une fois cette unification réalisée, les détails seront traités dans des niveaux considérés plus bas. Cette activité de schématisation est courante et s'appuie sur la connaissance de techniques et d'outils permettant d'analyser ce qui se passe (pouvoir d'abstraction) et de

notations permettant de communiquer à d'autres le résultat de ces analyses (schémas diagrammes, consignes).

II. ALGORITHMIQUE

II.1. Définition

L'algorithmique est l'ensemble des techniques utilisées pour concevoir des processus de calcul permettant à partir de données (en entrée) d'obtenir des résultats déterminés (en sortie).

Définition de l'*Encyclopedia Universalis* : « un algorithme est la spécification d'un schéma de calcul, sous forme d'une suite d'opérations élémentaires obéissant à un enchaînement déterminé ».

Un algorithme est donc une séquence d'étapes de calcul permettant de résoudre un problème combinatoire.

Autre définition : un algorithme est une procédure générale permettant de résoudre un problème : appliqué à n'importe quelle instance du problème, il produit sûrement une solution en un nombre de pas fini.

II.2. Elaboration d'un algorithme

Exemples d'algorithmes :

- les opérations arithmétiques
- l'algorithme d'Euclide : pour calculer le PGCD de deux nombres entiers …

II.2.1. Démarche algorithmique

Résolution dans IR, d'une équation du second degré : $aX^2+bX+c=0$

- Données (entrée) : coefficients du trinôme a, b et c
- Processus de calcul : du discriminant Δ, des racines R_1 et R_2
- Résultats : R_1 et R_2 (cas d'existence de deux racines réelles)

II.2.2. Phase conceptuelle ou Formalisation

a- Construction démonstrative

Il existe une ancienne pratique dans certains milieux professionnels, de contrôler la correction d'un programme à posteriori, en comparant les résultats qu'il fournit avec des résultats théoriques connus à l'avance.

Malheureusement, ce test ne permet jamais de conclure que le modèle de programme proposé est exact, mais seulement que les résultats observés ne permettent pas de le rejeter. Cette méthode de test est incapable de prouver qu'un programme est juste (sauf à considérer l'ensemble des valeurs possibles de test d'un domaine, ce qui est impensable).

D'où il est nécessaire de démontrer la validité d'un programme lors de sa conception. Ainsi, la construction d'un programme commence par une phase de formalisation démonstrative qui se place dans le domaine mathématique (outils et raisonnement mathématiques).
Les outils les plus utilisés sont les quantificateurs :

- universel ($\forall$), exprimant la répétition d'un calcul
- existentiel ($\exists$), exprimant l'existence certaine, unique ou hypothétique d'un élément dans un ensemble possédant une certaine propriété.

b- Les étapes de formalisation

La formalisation d'un algorithme comporte :

- ✓ Proposition d'une méthode de résolution. C'est l'étape préliminaire. Elle consiste à définir exactement ce que l'on veut faire. Ainsi, on doit détailler son domaine d'application (ses données ou arguments) et faire de même avec le ou les résultats calculés , en utilisant au mieux et au besoin des équations, des quantificateurs, des intervalles etc. Ces informations constituent la spécification de la fonction recherchée : c'est en quelque sorte un cahier des charges.

- ✓ Démonstration immédiate que la méthode proposée conduit aux résultats recherchés. C'est dans cette étape que l'on vérifie la complétude, la cohérence de la méthode spécifiée (on s'assure que toutes les situations ont été envisagées et qu'il ne subsiste pas de contradiction).

- ✓ Structuration des données. On doit choisir la représentation informatique des objets manipulés par la fonction (arguments, résultats, objets intermédiaires…). C'est la mise en oeuvre pratique de la méthode spécifiée.

- ✓ Définition des conditions initiales à réaliser pour valider la méthode proposée.

- ✓ Preuve de correction. Une fois le programme écrit, on le prouve comme on le fait avec une démonstration mathématique. Ainsi, on démontre la convergence de la méthode en un temps fini raisonnable.

- ✓ Regroupement et présentation de tous les éléments de calculs dégagés dans les étapes 2 et 4, nécessaires à l'obtention des résultats.

Ces différentes étapes constituent le formalisme algorithmique.
Illustration des différentes étapes :

1. Calcul de la somme des inverses des n premiers entiers
   a. Données : n
   b. Résultats : H

Définissons une suite numérique $\{u_n\}_{n \geq 1}$ par :

$$U_k = \sum_{i=1}^{k} \frac{1}{i}$$

2. Démonstration de la méthode : récurrence
   $H = u_n$
3. Le procédé de calcul effectif est obtenu grâce à la définition récurrente de U_k.
   $U_k = U_{k-1} + 1/k$
4. Conditions initiales : le calcul des termes de $\{U_k\}$ requiert la connaissance du premier d'entre eux, soit : U_1
5. l'algorithme est nécessairement convergent, puisque le nombre d'étapes de calcul n est connu au départ.
6. résumé du formalisme algorithmique :
   a. donnée : n ; résultat : H
   b. k=2, … n $U_k = U_{k-1} + 1/k$
   c. Résultat : $H = U_n$
   d. condition initiale : $U_1 = 1$

c- L'analyse

Il existe deux méthodes principales d'analyse de problèmes pour la construction d'un algorithme :

- la méthode descendante qui consiste à partir du problème initial, à descendre étape par étape, vers des sous-problèmes de plus en plus simples résoudre ou déjà résolus. L'idée est d'isoler les problèmes en résolvant d'abord les plus généraux avant de «descendre » aux particuliers. On suppose à priori résolubles les problèmes de niveau inférieur : ils sont résolus lorsque ceux du niveau supérieur sont entièrement résolus.
- La méthode ascendante qui part du niveau le plus bas et remonte jusqu'au problème à résoudre.

En fait, les deux méthodes sont toujours combinées et dans la plupart des cas, l'analyse n'est ni entièrement descendante, ni entièrement ascendante.

II.2.3. Phase technique (ou le programme)

L'étape conceptuelle établissait la faisabilité, la calculabilité d'un résultat en n'impliquant à priori aucune action, aucune contrainte d'ordre. Le formalisme bénéficiait de toutes les libertés autorisées par l'écriture mathématique. L'algorithme le reprend, l'ordonne selon la séquence des calculs et le structure, à l'aide des schémas de programme.
Un algorithme sera formalisé (simple recommandation et non une obligation) comme suit :

- Un en-tête :

- nom donné à l'algorithme
- données de l'algorithme
- les résultats fournis par l'algorithme

- Un corps

- début
  suite ordonnée d'équations (instructions) et de schémas de programmes
- fin

La mise en œuvre pratique sur un ordinateur, d'un algorithme (fruit d'une démarche logique de résolution d'un problème), nécessite de passer par une codification c'est-à-dire par l'intermédiaire d'un langage codifié (connu de l'ordinateur) ou langage de programmation qui comporte beaucoup de restrictions.

Pour s'affranchir de ces restrictions (syntaxe, codage méticuleux …), on utilise un 'pseudo-code' traduisible en n'importe quel langage usuel. Ainsi, on identifierait un programme à un algorithme codifié. Techniquement, le programme s'obtient en remplaçant les êtres mathématiques de l'algorithme par des êtres informatiques (avec en particulier, la notion de variable) et les équations par des affectations.

REMARQUE :

.

Un algorithme s'exprime de façon plus ou moins formelle : langage naturel, ou pseudo-code plus ou moins précis, ou langage de programmation de haut niveau. On peut se contenter des grandes lignes, en négligeant les problèmes d'implémentation fine, de façon à privilégier la lisibilité et la compréhension. En général, la description d'un algorithme tient en quelques lignes, et dépasse rarement une page.

On néglige à ce stade les problèmes de génie logiciel tels que protection, visibilité, modularité, traitement des erreurs . . . : ces aspects importants mais hors du champ de l'algorithmique sont traités en programmation.

III. ENSEMBLES ET SUITES

Les objets manipulés dans les exemples précédents appartiennent à des ensembles bien définis. C'est des collections de valeurs que peuvent prendre ces objets. Un ensemble peut être fini ou infini (les éléments sont distincts).

Une suite est une liste d'objets pas nécessairement distincts, indexables par des entiers. Elle peut être fini ou non. Un mot, par exemple, peut être considéré comme une suite finie de lettres appartenant à un alphabet.

La notion de suite est fondamentale en informatique si bien que les langages de programmation présentent des facilités de manipulation.

Il revient à l'implémenteur de fixer les conventions de représentations des ensembles.

Exercice3 :

a. Ecrire un pseudo-code qui factorise tout entier *p>1* en produit de nombres premiers.

b. Soit l'ensemble *{2, 3, …, n}*. On se propose d'extraire de cet ensemble la liste de tous les nombres premiers inférieurs à *n.*
   - Proposer une méthode
   - Vérifier sa validité
   - Ecrire l'algorithme correspondant.

-

III.1. Types de données

Les objets utilisés peuvent être des constantes ou des variables. A chaque objet est associé implicitement ou explicitement un type qui caractérise l'ensemble des valeurs possibles pour cet objet ainsi que les actions auxquelles celui-ci peut être soumis.

Les types de données suivants :

- entier
- réel
- booléen
- caractère

sont qualifiés d'élémentaires' ou 'scalaires'.

III.2. Structures de données

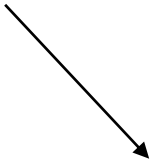
Une structure de données est aussi un type de données, seulement, on utilise le terme de structure de données pour désigner les types de données complexes construits à l'aide des types élémentaires.
Exemples : tableaux, structure, pointeurs, liste, …

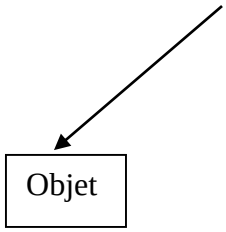
III.3. Les Actions

- **Affectation :** opération qui consiste à attribuer (affecter) une valeur à un objet. Elle est fréquemment utilisée.

Exemple :

V ← e





Cette action correspond une primitive (instruction) dans tous les langages de programmation.

- **Instructions d'entrée et de sortie**
  - Lire (o_1, o_2, ..., o_n) lit dans les objets o_1, o_2, ..., o_n les valeurs frappées au clavier.
  - Afficher (o_1, o_2, ..., o_n) permet d'envoyer à l'écran ou d'imprimer les valeurs des objets o_1, o_2, ..., o_n

Ces instructions sont facilement codés dans la plupart des langages de programmation.

III.3.1. Structures conditionnelles

Soit ***a*** et ***s*** des suites d'actions et ***p*** un prédicat, c'est-à-dire une expression logique. On peut avoir un schéma conditionnel ou un schéma alternatif.

- schéma conditionnel

  Si ***p***

  Alors ***a***

  finsi

- schéma alternatif

  Si ***p***

Alors ***a***

Sinon ***s***

Finsi

Si ***p*** est vraie, alors la suite d'actions ***a*** est exécutée dans les deux schémas. Par contre si ***p*** prend la valeur fausse, alors la suite d'actions ***s*** est exécutée pour le second schéma.

III.3.2. Choix multiple

Cas ***identificateur*** parmi

< val_1 > : a_1

. .

. .

. .

< val_n > : a_n

Sinon : ***s***

FinCas

Le terme ***identificateur*** désigne une variable ou une expression de même type que les constantes des ensembles **<val_i>**, i=1, …, n.

a_i et **s** sont des suites d'actions.

III.3.3. Structures itératives

Attention : Toute répétition d'actions doit avoir une fin contrôlée au moyen d'un prédicat.

- Si le nombre d'itérations est connu à l'avance, on utilise le schéma suivant :

Pour ***iddébut*** *à* ***idfin*** *pas de <incrément> Faire*

a

FinPour

OU

Pour ***iddébut*** ***à*** ***idfin*** *pas de <décrément> Faire*

a

FinPour

- Si on ne connaît pas le nombre d'itérations, on peut utiliser l'un des deux schémas généraux suivants :

  Schéma 1

  Itérer

  a

  sortirsi **p**

  b

  FinItérer

  1. Si **a** et **b** sont des suites non vides d'actions, **a** est exécutée une fois de plus que **b.**
  2. Si **a** est vide, la structure est équivalente à la boucle :

     Tant que non **p** faire **b**

  3. Si **b** est vide, la structure est équivalente à la boucle :
     Répéter **a** *jusqu'à* p

  Schéma 2

  Itérer

  a

  sortirsi **p**

b

sortirsi **q**

c

FinItére