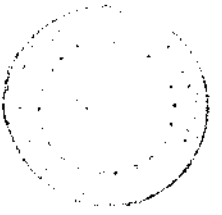
808
C. 157

BIBLIOTHEQUE DU CERIST

LA TECHNIQUE INFORMATIQUE

I

PRINCIPES GÉNÉRAUX ET PROGRAMMATION



IST 180

LA TECHNIQUE INFORMATIQUE

TOME I

PRINCIPES GÉNÉRAUX ET PROGRAMMATION

par

Hélène BESTOUGEFF

Maître de Conférences
à l'Université de Paris VII

Christian GUILPIN et Michel JACQUES

Maître-Assistant
à l'Université de Paris VII

Maître-Assistant
à l'Université de Paris VII

PRÉFACE DU Pr J. ARSAC

MASSON ET CIE, ÉDITEURS
120, BOULEVARD SAINT-GERMAIN, PARIS, VI^e^
1975

BIBLIOTHEQUE DU CERIST

PRÉFACE

Le travail d'un enseignant en Informatique est bien difficile. L'évolution est trop rapide. On n'a pas le temps de s'installer confortablement et de répéter le même cours pendant des années ! Toujours sur le métier, il faut remettre son ouvrage.

Les années 60 ont vu s'installer l'informatique en tant que technique de calcul. On a mis en place des cours techniques de programmation, orientés surtout vers les langages à grande diffusion : Fortran, Algol, Cobol, Basic, langages d'assemblage de telles machines largement répandues. Enseignement technique : on insistait plus sur l'outil à manipuler que sur le tour de main de l'artisan.

Vers les années 68-70, on a pris conscience qu'il y avait aussi une science informatique. On en a dégagé le contenu. On en a tiré les conséquences sur le plan de l'enseignement. Le colloque international sur l'enseignement de l'Informatique tenu à Amsterdam en 1970 consacrait les thèses françaises en ce domaine. Bien sûr, cette science n'est pas sortie toute armée de la cuisse de quelque Jupiter universitaire. Ce sont davantage des réflexions sur un certain nombre de cours, bâtis empiriquement, qui l'ont peu à peu dégagée. L'enseignement a alors quitté la technique pour essayer d'atteindre le niveau plus fondamental de la connaissance en soi, intéressante même si elle n'a pas tout de suite une application. Il n'est pas certain que tous les étudiants comprennent et admettent cette mutation. Il y aurait beaucoup à dire sur l'efficacité à court terme obtenue par un cours très technique, opposée à une efficacité plus lointaine bâtie sur une connaissance moins liée aux contingences technologiques, et plus à même de faire face à des changements importants dans ce domaine. Il faut se persuader que l'Informatique n'est pas originale dans son développement : elle évolue comme toutes les autres sciences expérimentales. Partie de l'empirisme, elle ira de plus en plus vers la modélisation et la formalisation. Comment l'enseignement pourrait-il éviter cette évolution ?

L'époque actuelle est celle de deux mutations très profondes. Après avoir été affaire de technicien, puis de spécialistes, l'Informatique diffuse dans toute la culture scientifique et devient un enseignement de masse. Elle est au programme de tous les premiers cycles universitaires scientifiques. Elle entre au lycée, petitement, lentement, mais, à mon avis, irréversiblement. Elle est une composante importante de la culture scientifique, et, je le crois, de la culture générale : elle a trop de liens avec la linguistique pour rester affaire de scientifiques (au sens ancien ou restreint du terme). Il faut adapter

BIBLIOTHEQUE DU CERIST

l'enseignement à cette mutation. Le public n'est plus le même, la façon de l'intéresser n'est plus la même.

Par ailleurs, l'informatique se "scientifisant" ne peut plus se contenter des "recettes de cuisine" qui ont prévalu pendant la dernière décennie. Je sais bien que Claude Bernard compare la science à un salon brillamment illuminé dans lequel on ne parvient qu'après avoir traversé une longue et affreuse cuisine. Encore faut-il sortir de la cuisine. Le mal a été abondamment dénoncé. Les remèdes sont encore mal connus, surtout au niveau de l'enseignement. Malgré mes efforts, je n'ai encore pu faire inscrire l'enseignement de la programmation au programme d'un colloque international, tant cela paraît un sujet redoutable.

Le présent ouvrage s'inscrit heureusement dans la perspective de cette double mutation. Il est destiné à un large public pour lequel l'informatique n'est pas une fin, mais qui veut être informé des démarches de l'esprit qu'elle met en jeu. Il ne fait aucune concession à la facilité. La résolution d'un problème par l'informatique y est discutée très à fond. On n'enseigne pas plus la programmation en informatique que l'on n'enseigne la démonstration en mathématiques. Les dossiers de programmation contenus dans cet ouvrage ont la même valeur d'exemple que des théorèmes démontrés dans un cours de mathématiques. C'est, pour le moment, ce que l'on peut faire de mieux. Il faudrait être devin pour savoir quels seront les aboutissements des recherches actuelles en programmation et en architecture de systèmes informatiques. Mais ce dont je suis sûr, c'est que ce livre sera longtemps utile à tous ceux qui veulent avoir un minimum de culture en informatique.

J. ARSAC
Professeur titulaire
à l'Université Paris VI.
Directeur scientifique
de l'Institut de Programmation.

BIBLIOTHEQUE DU CERIST

BIBLIOTHEQUE DU CERIST

TABLE DES MATIÈRES DU TOME I

BIBLIOTHEQUE DU CERIST

BIBLIOTHEQUE DU CERIST

Pages

BIBLIOTHEQUE DU CERIST

BIBLIOTHEQUE DU CERIST

BIBLIOTHEQUE DU CERIST

AVANT-PROPOS

BIBLIOTHEQUE DU CERIST

L'ouvrage *La Technique Informatique* se présente en deux Tomes :

Tome I : *Principes généraux et programmation.*
Tome II : *Algorithmes numériques et non numériques.*

Il est destiné d'une part aux étudiants des premiers cycles universitaires (au niveau d'une initiation) et d'autre part aux ingénieurs et chercheurs qui, sans être informaticiens, désirent connaître les principes de base et les techniques de programmation pour utiliser l'ordinateur dans leur spécialité.

Cet ouvrage est donc conçu d'un point de vue résolument pratique et seules les considérations théoriques indispensables sont abordées. Le langage de programmation choisi est le FORTRAN IV, version de base. Ce langage a l'avantage d'être le plus répandu et d'être rapidement mis en œuvre par un débutant. En outre, les remarques particulières faites au niveau de cette programmation se transposent aisément à un autre langage dans la mesure où nous nous sommes attachés à la discussion détaillée de l'algorithme.

Le premier tome est divisé en trois chapitres complétés par une annexe FORTRAN :

Chapitre I : généralités sur la notion d'algorithme
Chapitre II : structure fonctionnelle d'un système informatique
Chapitre III : comment aborder un problème ?

Le chapitre I est une discussion élémentaire sur la notion d'algorithme et l'écriture d'un programme FORTRAN correspondant. La description de l'algorithme est faite en termes d'organigramme ; cette représentation sous forme de schémas, bien que comportant certains inconvénients, reste à l'heure actuelle la forme la plus accessible à tous.

Le chapitre II introduit des éléments sur la structure fonctionnelle des ordinateurs : représentation et manipulation des informations. Ces connaissances sont indispensables à la compréhension des limitations de la machine et à l'examen critique des résultats.

Le chapitre III présente une discussion complète de cinq exemples. Les problèmes choisis sont d'une part suffisamment simples pour permettre une analyse détaillée tant du point de vue théorique que pratique et d'autre part d'un caractère suffisamment général pour que la démarche puisse être transposée à d'autres cas.

Dans chaque chapitre, de nombreux exercices partiellement ou complètement corrigés doivent permettre au lecteur de vérifier que les concepts présentés ont été bien assimilés.

Bien que les spécifications du langage FORTRAN soient regroupées en Annexe, les éléments du langage et son utilisation sont présentés progressivement au cours des exemples.

Le lecteur pourra lire rapidement les deux premiers chapitres et aborder aussitôt avec profit le chapitre III. Lors de la discussion des problèmes, il se reportera aux deux premiers chapitres afin d'approfondir les points délicats.

Les Auteurs

BIBLIOTHEQUE DU CERIST

BIBLIOTHEQUE DU CERIST

CHAPITRE I

GÉNÉRALITÉS SUR LA NOTION D'ALGORITHME

Le mot algorithme vient de la déformation du nom d'un mathématicien Ouzbek du IX^{ème} siècle, Muhammed ibn Musa-Al-Kharezmi. Il fut l'un des premiers à indiquer les règles du calcul algébrique et c'est notamment dans un de ses ouvrages que figurent tous les éléments de la théorie des équations du second degré.

A l'heure actuelle, d'une manière générale, le terme d'algorithme définit un procédé de calcul. Des définitions plus précises des algorithmes sont apparues à la suite des recherches sur les fonctions calculables et les automates, notamment par A. Markov.

Dans le cadre de cet ouvrage, nous définirons un algorithme comme : une suite finie d'instructions indiquant de façon précise l'ordre dans lequel doit être effectué un ensemble d'opérations pour obtenir la solution de tous les problèmes d'un type donné. Le nombre d'opérations décrites et le nombre d'opérations exécutées doivent être finis. Pour chaque algorithme, il existe alors un ensemble de données auxquelles nous pouvons "appliquer" l'algorithme, c'est-à-dire pour lesquelles l'algorithme a un sens.

La recherche d'algorithmes est une préoccupation ancienne ; qui ne connaît l'algorithme d'Euclide permettant de trouver le plus grand commun diviseur p.g.c.d. de deux nombres entiers positifs ? L'intérêt actuel provient du fait que nous avons maintenant à notre disposition de nouveaux outils pour mettre en œuvre les algorithmes : les ordinateurs.

Le développement pragmatique de l'outil a entraîné dans certains cas une confusion regrettable entre informatique et ordinateur. Pour un problème donné, nous pouvons imaginer divers algorithmes permettant d'aboutir à la solution cherchée. Un même algorithme par ailleurs peut être décrit de diverses façons. Les considérations qui nous amènent à choisir tel algorithme et telle description dépendent du point de vue envisagé : pratique, économique, formel, etc.

Etant donné que dans cet ouvrage, nous nous plaçons au niveau d'une initiation aux techniques de l'informatique, nous avons choisi les organigrammes comme étape intermédiaire de description des algorithmes. Ce choix est également motivé par le langage de programmation utilisé : le FORTRAN. Il est certain que compte tenu de l'évolution des langages et des structures des ordinateurs, ce choix peut être remis en cause.

BIBLIOTHEQUE DU CERIST

1. – DESCRIPTION D'UN ALGORITHME. L'ORGANIGRAMME

Dans la définition, nous avons supposé qu'un algorithme était indépendant de l'instrument ou plus généralement du processeur(*) (homme ou machine) qui effectue les opérations ; en fait, lorsque nous voulons décrire un algorithme, il est impossible d'ignorer le processeur, car le niveau de détail nécessaire en dépend éventuellement.

Prenons un exemple extrêmement simple :

Décrire l'algorithme permettant d'additionner deux nombres de deux chiffres a et b. Nous supposerons pour l'instant qu'il n'y a pas de retenue sur les unités.

a) Si le processeur est un adulte, l'énoncé seul suffit, l'addition étant considérée comme *une opération élémentaire*.

b) Si le processeur est un enfant de la classe préparatoire qui ne sait faire que l'addition de chiffres, il faut préciser la description :

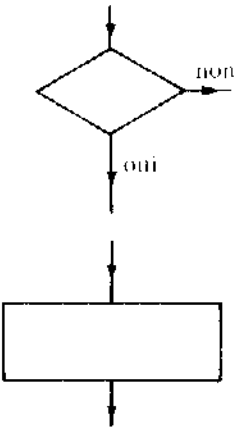
additionner les chiffres des unités de a et b, écrire le résultat,

additionner les chiffres des dizaines de a et b, écrire le résultat à gauche du précédent.

c) Si le processeur est une machine, nous devons connaître *les opérations élémentaires* qu'elle peut effectuer, afin de décrire l'algorithme à ce niveau.

Considérons, pour le moment, le "processeur humain". Le langage courant est en général imprécis et souvent ambigu ; son utilisation exclusive pour la description d'algorithmes paraît donc peu appropriée dès que l'enchaînement des opérations à effectuer devient complexe. Nous pourrions chercher une formulation plus proche du langage mathématique mais cette approche est trop abstraite dans le cadre d'une initiation ; aussi préférons-nous introduire une notation plus "imagée", sous forme de schémas appelés *organigrammes*.

Pour cela, nous séparons arbitrairement les opérations en deux types : les opérations de décision en fonction d'un résultat partiel (test) d'une part, les autres opérations d'autre part, et nous associons à chaque type un symbole particulier :



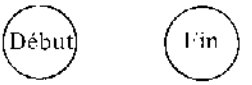
opération de test

autre opération

(*) Néologisme provenant de l'anglais *processor* (ou *process* : traiter).

La nature des opérations est indiquée à l'intérieur du losange et du rectangle ; les flèches indiquent l'enchaînement des opérations. L'opération de test revient à poser une question pour laquelle il y a deux réponses qui s'excluent (oui-non) ; pour chacune des réponses, l'opération qui suit est différente, l'enchaînement étant toujours matérialisé par les flèches.

Un algorithme doit toujours avoir un début et au moins une fin, sur l'organigramme nous utiliserons les deux symboles suivants :



Nous pouvons maintenant représenter l'organigramme de l'algorithme (*b*) de l'addition de deux nombres de deux chiffres (fig. I.1).

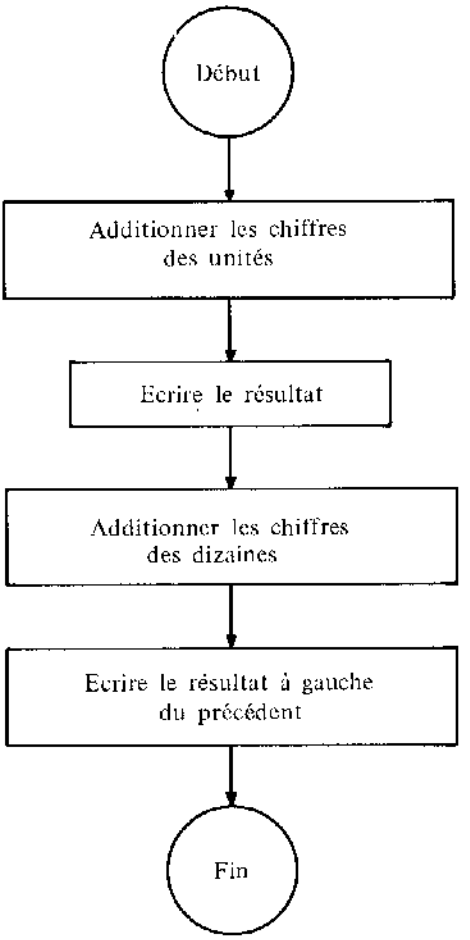


Fig. I.1.

BIBLIOTHEQUE DU CERIST

Nous avons supposé précédemment qu'il n'y avait pas de retenue sur les dizaines ; si nous enlevons cette restriction, nous devons reformuler l'algorithme en faisant un test sur la retenue (fig. I.2).

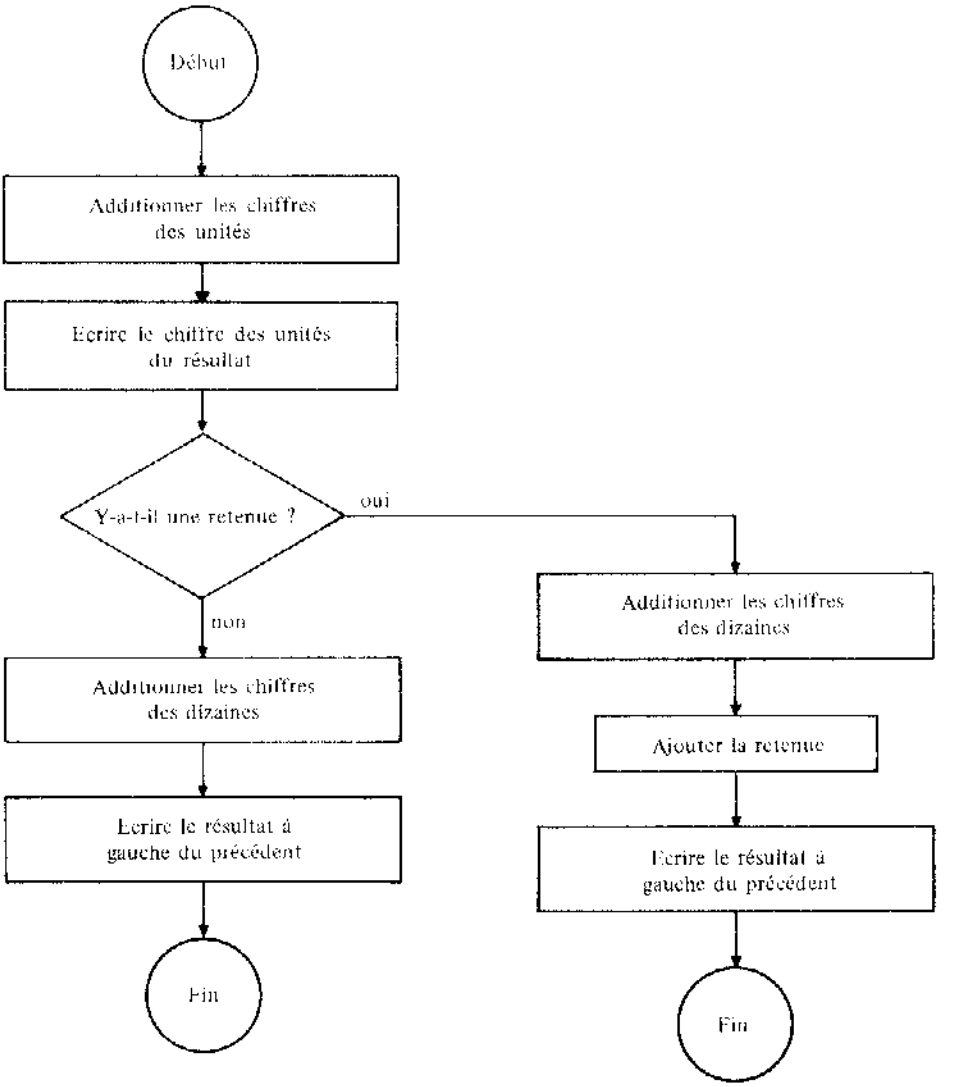


Fig. I.2

Nous constatons qu'il y a deux "fin d'algorithme" possibles suivant qu'il y a ou non une retenue. Les deux organigrammes précédents présentent clairement les enchaînements des opérations ; par contre, la définition des opérations est encore donnée en langage naturel. Dans un premier temps, ce type d'organigramme peut être utile surtout dans le cas d'enchaînements complexes, mais il est important de préciser également les opérations.

BIBLIOTHEQUE DU CERIST

Les opérations envisagées (arithmétiques, logiques) sont en général définies sur des objets auxquels nous pouvons attacher deux attributs : le *nom* et la *valeur*.

I.1. — Opération d'affectation

Définissons une opération entre deux objets m et n, notée $m \leftarrow n$ et appelée *opération d'affectation* qui consiste à remplacer la valeur de m par la valeur de n. Par exemple, si m et n sont deux variables ayant respectivement pour valeur 15 et 7, après l'opération "$m \leftarrow n$" les deux variables auront la même valeur (soit 7 dans ce cas). S'il s'agit d'une constante, par exemple $m \leftarrow 255$, la valeur à affecter est explicitement donnée.

Il faut bien noter que cette opération est définie statiquement sur les *noms* de variables et que c'est seulement l'exécution — aspect dynamique de l'opération — qui modifie la valeur. La constante est un cas particulier où le *nom* et la *valeur* sont confondus, ce qui entraîne que la *valeur ne peut être modifiée* ; l'écriture $225 \leftarrow m$ *n'a pas de sens.*

Sauf pour les constantes, la valeur d'une variable n'est pas connue a priori ; il faut donc faire précéder toute exécution d'un calcul d'une étape d'initialisation qui va affecter à chaque variable une valeur initiale.

D'une façon générale, un algorithme est une suite d'opérations définies sur des noms de variables. L'exécution d'un algorithme modifie dynamiquement les valeurs des variables, ce qui explique qu'une variable puisse avoir plusieurs valeurs différentes en *cours d'exécution*.

Nous verrons plus loin que, pour faire exécuter un algorithme par ordinateur, le concept de variable est traduit en machine par celui de *case-mémoire*. Chaque case a une adresse (le nom) et un contenu (la valeur). Les opérations-machine sont définies sur les adresses des cases, l'exécution modifiant les contenus.

Adoptons la numération habituelle de position et reprenons l'algorithme de la figure I.2 en utilisant l'opération d'affectation. Nous désignons par x et y d'une part, et w et z d'autre part, les variables qui auront pour valeur les chiffres des unités et des dizaines des nombres a et b, respectivement (fig. I.3).

Nous avons généralisé l'utilisation de l'opération d'affectation en permettant un calcul d'une valeur à droite du signe $\leftarrow$. Il est évident que ce calcul n'a de sens que si les valeurs des variables sur lesquelles se fait le calcul sont connues. L'écriture $u \leftarrow u - 10$ par exemple signifie : prendre la valeur de u (qui est supérieure à 9 puisque nous sommes dans la branche de droite), retrancher la valeur de 10, qui est égale à 10 par définition, et donner cette *nouvelle valeur* à u. L'ancienne valeur de u est maintenant perdue.

Si nous examinons la figure I.3, nous constatons que les opérations d'écriture sont identiques dans les deux branches et qu'il peut être intéressant de les regrouper pour avoir un organigramme plus concis (fig. I.4).

Nous pouvons encore remarquer qu'après le test, les deux branches comportent l'addition des chiffres des dizaines ; il peut donc sembler judicieux de faire cette opération avant le test (fig. I.5).

BIBLIOTHEQUE DU CERIST

BIBLIOTHEQUE DU CERIST

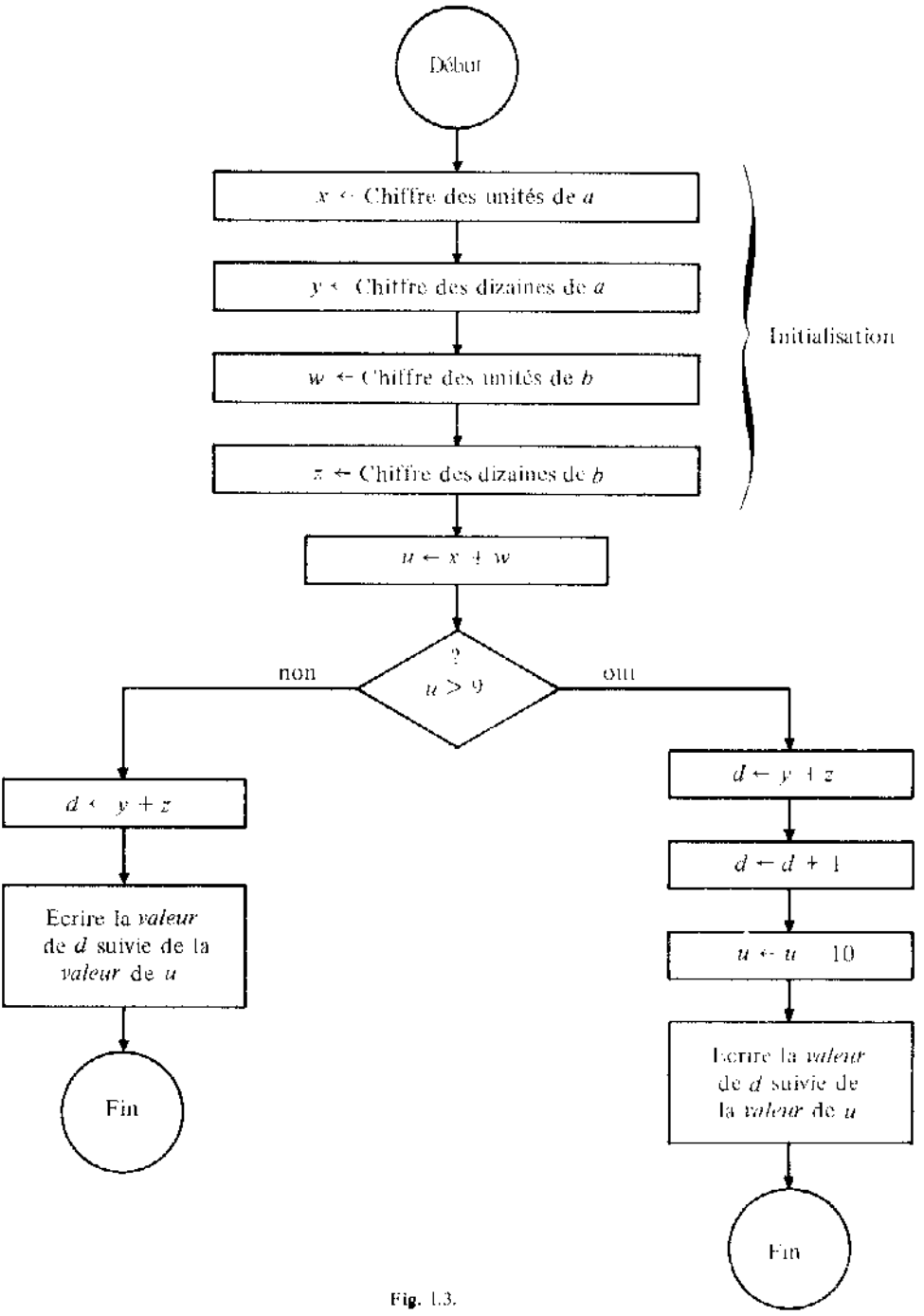


Fig. I.3.

BIBLIOTHEQUE DU CERIST

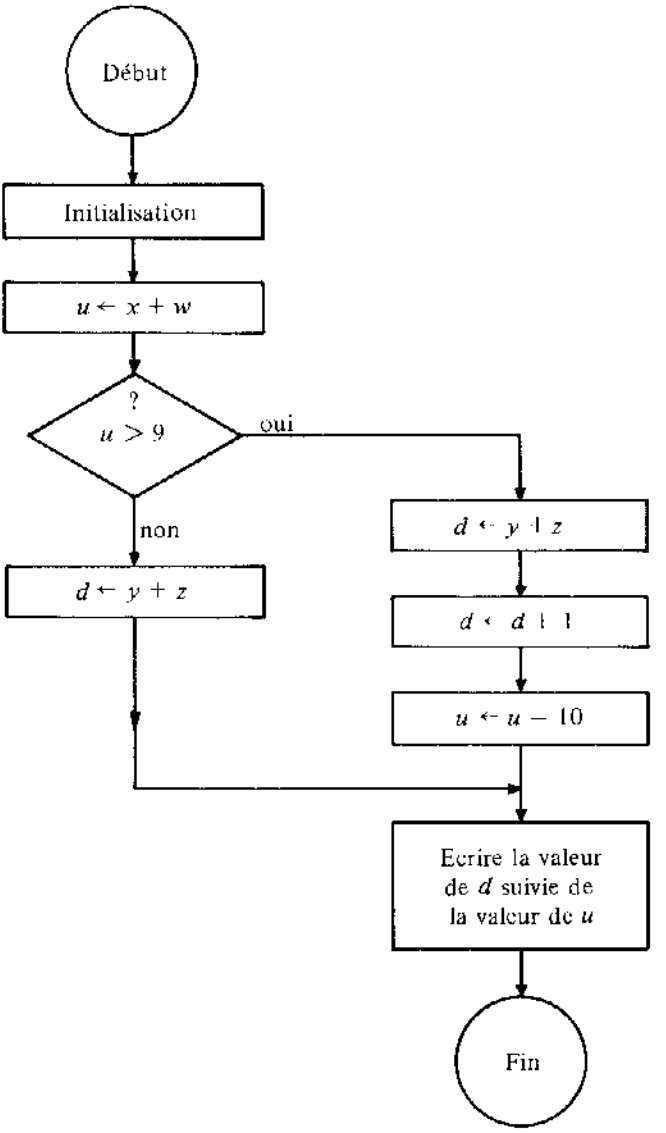


Fig. I.4.

BIBLIOTHEQUE DU CERIST

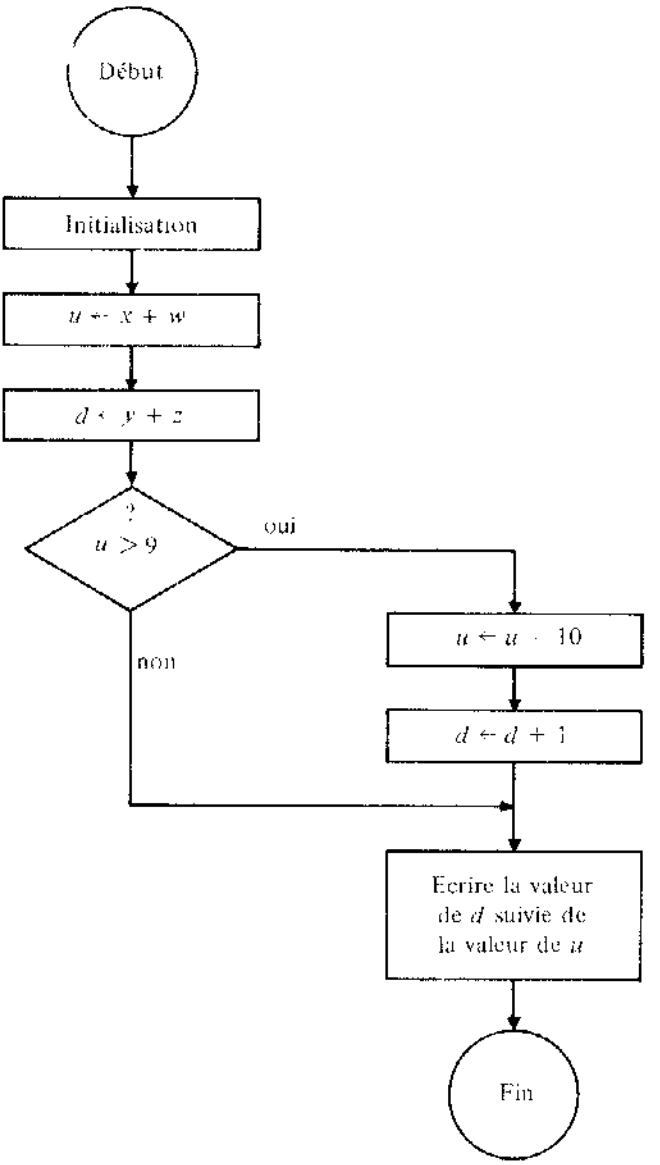


Fig. I.5.

Les enchaînements d'opérations que nous avons présentés jusqu'à maintenant sont simples. Si le nombre d'opérations *à effectuer* pour obtenir le résultat était toujours le même, en revanche le nombre d'opérations *décrites* a été diminué en faisant des regroupements.

Donnons schématiquement la structure des différents organigrammes rencontrés (*fig. I.6*) :

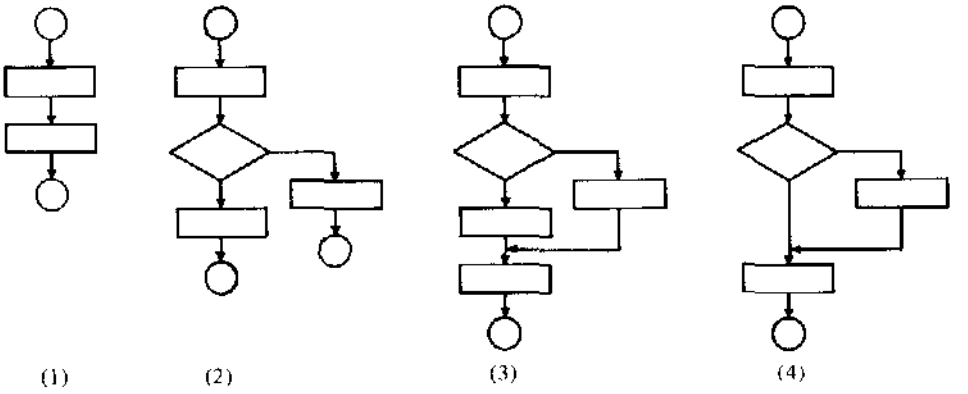


Fig. I.6.

(1) : traitement séquentiel.

(2) : à partir de l'opération de test deux traitements distincts sont possibles suivant *la valeur* des données.

(3) et (4) : les traitements ne se différencient que pour un certain nombre d'opérations intermédiaires :

Ces organigrammes sont en fait des éléments de base pour la construction des organigrammes en général. Dans les exemples précédents chaque opération est exécutée *au plus une fois* pour un groupe de données. En cherchant à généraliser à deux nombres de n chiffres le processus d'addition, nous rencontrerions un dernier élément de base d'organigramme qui est à vrai dire le plus fondamental : il traduit la notion de répétition qui est le propre des automatismes. Pour le mettre en relief, nous préférons présenter cet élément de façon isolée dans un exemple plus spécifique.

I.2. — Algorithme d'Euclide

Considérons maintenant l'algorithme d'Euclide qui permet de trouver le PGCD de deux nombres entiers positifs m et n. Admettons ici que la valeur de m est toujours supérieure ou égale à la valeur de n. Nous pouvons d'abord décrire l'algorithme par la suite d'instructions suivantes exécutées séquentiellement sauf mention contraire :

1/ Diviser m par n tel que r soit le reste (nous aurons $0 \leqslant r < n$).

2/ Si r est nul alors le p.g.c.d. de m et de n est n, fin de l'algorithme.

Sinon aller en 3.

3/ Prendre pour nouvelle valeur de m la valeur de n, et pour nouvelle valeur de n la valeur de r.

4/ Recommencer le traitement en 1.

L'instruction (4) ne correspond pas à une opération sur les données (objets); elle indique seulement qu'il faut recommencer les opérations *avec des valeurs* de m et n qui ont été modifiées par la première exécution des opérations.

Construisons l'organigramme correspondant (fig. I.7).

BIBLIOTHEQUE DU CERIST

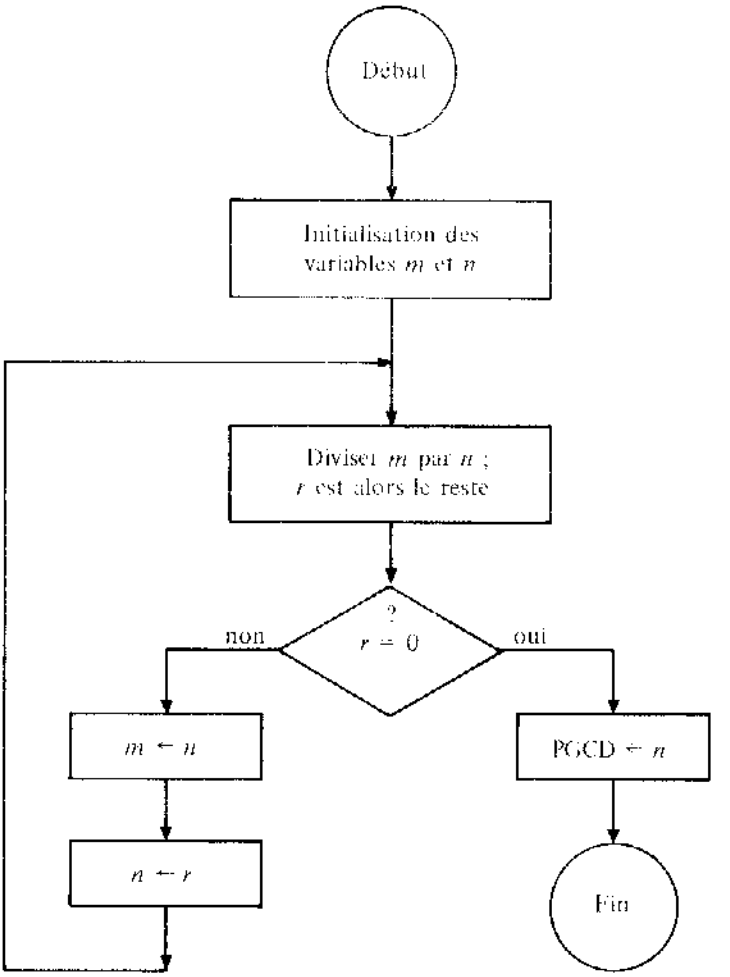


Fig. I.7.

Nous constatons que l'instruction 4 a été traduite par une flèche indiquant à quel niveau il faut recommencer les opérations. Lorsqu'il y a une *réexécution* des mêmes opérations, nous dirons qu'il y a une *boucle*.

La définition de l'algorithme précise que le nombre d'opérations doit être fini ; or, à partir du moment où il existe une boucle dans un organigramme, cette condition dépend du jeu de données utilisées. Il faut donc s'assurer que, pour toutes données, la sortie d'un test, à un certain moment, provoque un arrêt de l'exécution. Nous reviendrons sur ce point au § I.4.

L'utilisation d'une boucle permet de limiter le nombre de variables utilisées. En effet, nous aurions pu essayer de décrire l'algorithme d'Euclide en utilisant en plus de m et de n une suite de variables r_i en construisant par exemple le schéma suivant (fig. I.8) :

BIBLIOTHEQUE DU CERIST

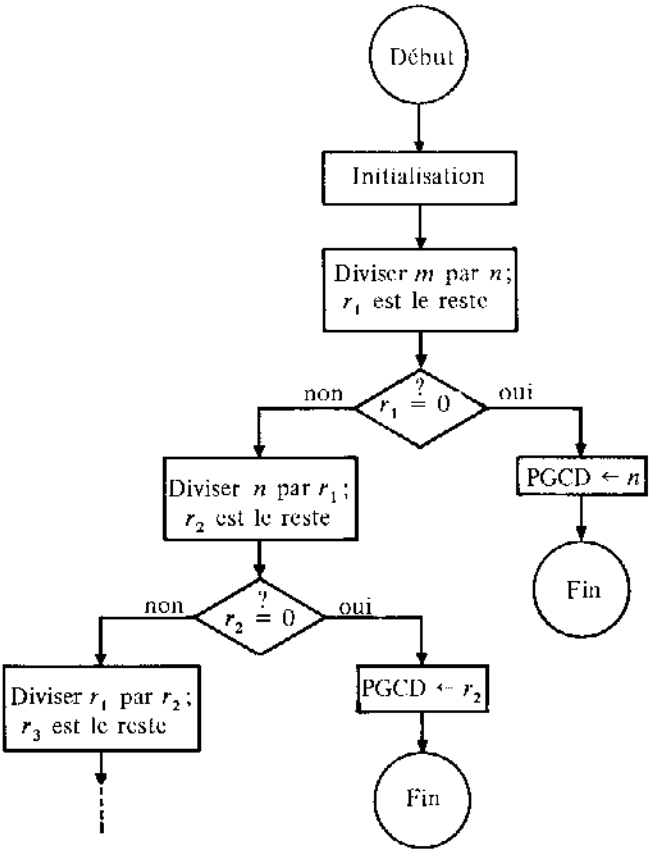


Fig. I.8.

Nous constatons d'abord qu'il est impossible a priori de prévoir le nombre de variables r_i nécessaires pour un couple quelconque de nombres entiers (m, n). De plus, les valeurs de tous les r_i intermédiaires ne servent à rien dans la suite du calcul ; à chaque étape de calcul, seules 3 variables sont nécessaires, dont les valeurs vont évoluer au cours de l'exécution.

Plus généralement, si nous pouvons décrire une partie d'un algorithme par une suite récurrente, comme par exemple celle sous-jacente à l'algorithme d'Euclide

$$r_1 = m$$
$$r_2 = n$$
$$\ldots\ldots$$
$$r_{i+2} = r_i \quad \text{modulo} \quad r_{i+1} ,$$

il faut déterminer le nombre de variables nécessaires pour obtenir une valeur r_{i+2} et *se limiter* à ce nombre pour le calcul d'un quelconque $i^{\text{ème}}$ terme.

BIBLIOTHEQUE DU CERIST

Appliquons l'algorithme d'Euclide aux deux nombres entiers 297 et 63, soit pour l'initialisation $m \leftarrow 297$, $n \leftarrow 63$.

1. $m/n = 4 + \frac{45}{63}$ d'où $r \leftarrow 45$

2. $r = 0$? non

3. $m \leftarrow 63$, $n \leftarrow 45$

1'. $m/n = 1 + \frac{18}{45}$ d'où $r \leftarrow 18$

2'. $r = 0$? non

3'. $m \leftarrow 45$, $n \leftarrow 18$

1''. $m/n = 2 + \frac{9}{18}$ d'où $r \leftarrow 9$

2''. $r = 0$? non

3''. $m \leftarrow 18$, $n \leftarrow 9$

1'''. $m/n = 2$ d'où $r \leftarrow 0$

2'''. $r = 0$? oui PGCD $\leftarrow n$ *Fin*

2. – LE PROGRAMME

Dans le cadre de cet ouvrage, nous nous préoccupons principalement d'algorithmes destinés à être exécutés par un ordinateur.

Nous appellerons *Programme*, un algorithme *décrit dans un langage tel* qu'il puisse être exécuté par l'ordinateur. Etant donné la structure actuelle des ordinateurs, que nous verrons succintement dans le chapitre suivant, le langage utilisé devra être précis et ne comporter ni ambiguïtés, ni redondances. Un tel langage est dit *langage de programmation*.

La construction d'organigrammes se fera alors en termes d'opérations ou d'instructions que comporte le langage.

Il existe de très nombreux langages de programmation et le choix d'un langage particulier dépend à la fois du type de données traitées et de l'ordinateur utilisé. Pour l'ensemble de cet ouvrage, nous avons choisi le FORTRAN IV dans sa version de base pour les raisons suivantes :

- c'est à l'heure actuelle le langage le plus répandu parmi les utilisateurs, et tous les constructeurs le fournissent ;

- c'est un langage qui s'apprend très rapidement car il a peu d'instructions ;

– les possibilités des autres langages peuvent presque toujours être simulées en FORTRAN (la programmation de ces simulations est un excellent exercice) ;

les instructions sont relativement proches des instructions-machine.

BIBLIOTHEQUE DU CERIST