Conception et compilation d'un langage pour l'écriture de cours

Camille Bellissant

N° d'ordre

THESE

présentée à

LA FACULTE DES SCIENCES DE GRENOBLE

pour obtenir

LE GRADE DE DOCTEUR DE TROISIEME CYCLE

"INFORMATIQUE"

par

Camille BELLISSANT

Conception et compilation d'un langage pour l'écriture de cours

Thèse soutenue le 18 Mars 1970 devant la commission d'examen

Monsieur	J. KUNTZMANN	Président
Messieurs	N. GASTINEL	
	Y. LE CORRE	Examinateurs
	J.C. BOUSSARD	

L I S T E D E S P R O F E S S E U R S

Doyen honoraire : Monsieur M. MORET
Doyen : Monsieur E. BONNIER

PROFESSEURS TITULAIRES

MM.	NEEL Louis	Physique Expérimentale
	KRAVTCHENKO Julien	Mécanique Rationnelle
	CHABAUTY Claude	Calcul différentiel et intégral
	BENOIT Jean	Radioélectricité
	CHENE Marcel	Chimie Papetière
	FELICI Noël	Electrostatique
	KUNTZMANN Jean	Mathématiques Appliquées
	BARBIER Reynold	Géologie Appliquée
	SANTON Lucien	Mécanique des Fluides
	OZENDA Paul	Botanique
	FALLOT Maurice	Physique Industrielle
	KOSZUL Jean-Louis	Mathématiques
	GALVANI Octave	Mathématiques
	MOUSSA André	Chimie Nucléaire
	TRAYNARD Philippe	Chimie Générale
	SOUTIF Michel	Physique Générale
	CRAYA Antoine	Hydrodynamique
	REULOS René	Théorie des Champs
	BESSON Jean	Chimie Minérale
	AYANT Yves	Physique Approfondie
	GALLISSOT François	Mathématiques
Melle.	LUTZ Elisabeth	Mathématiques
MM.	BLAMBERT Maurice	Mathématiques
	BOUCHEZ Robert	Physique Nucléaire
	LLIBOUTRY Louis	Géophysique
	MICHEL Robert	Minéralogie et pétrographie
	BONNIER Etienne	Electrochimie et Electrométallurgie
	DESSAUX Georges	Physiologie animale
	PILLET Emile	Physique Industrielle-Electrotechnique
	YOCCOZ Jean	Physique Nucléaire théorique
	DEBELMAS Jacques	Géologie Générale
	GERBER Robert	Mathématiques
	PAUTHENET René	Electrotechnique
	MALGRANGE Bernard	Mathématiques Pures
	VAUQUOIS Bernard	Calcul Electronique
	BARJON Robert	Physique Nucléaire

MM.	BARBIER Jean-Claude	Physique
	SILBER Robert	Mécanique des Fluides
	BUYLE-BODIN Maurice	Electronique
	DREYFUS Bernard	Thermodynamique
	KLEIN Joseph	Mathématiques
	VAILLANT François	Zoologie et Hydrobiologie
	ARNAUD Paul	Chimie
	SENGEL Philippe	Zoologie
	BARNOUD Fernand	Biosynthèse de la Cellulose
	BRISSONNEAU Pierre	Physique
	GAGNAIRE Didier	Chimie Physique
Mme.	KOFLER Lucie	Botanique
MM.	DEGRANGE Charles	Zoologie
	PEBAY-PEROULA Jean-Claude	Physique
	RASSAT André	Chimie Systématique
	DUCROS Pierre	Cristallographie Physique
	DODU Jacques	Mécanique Appliquée I. U. T.
	ANGLES D'AURIAC Paul	Mécanique des Fluides
	LACAZE Albert	Thermodynamique
	GASTINEL Noël	Analyse numérique
	GIRAUD Pierre	Géologie
	PERRET René	Servo-mécanisme
	PAYAN Jean-Jacques	Mathématiques Pures

PROFESSEURS SANS CHAIRE

MM.	GIDON Paul	Géologie
Mme.	BARBIER Marie-Jeanne	Electrochimie
Mme.	SOUTIF Jeanne	Physique
	COHEN Joseph	Electrotechnique
	DEPASSEL R.	Mécanique des Fluides
	GLENAT René	Chimie
	BARRA Jean	Mathématiques Appliquées
	COUMES André	Electronique
	PERRIAUX Jacques	Géologie et Minéralogie
	ROBERT André	Chimie Papetière
	BIARREZ Jean	Mécanique Physique
	BONNET Georges	Electronique
	CAUQUIS Georges	Chimie Générale
	BONNETAIN Lucien	Chimie Minérale
	DEPOMIER Pierre	Physique Nucléaire-Génie Atomique
	HACQUES Gérard	Calcul numérique
	POLOUJADOFF Michel	Electrotechnique
Mme.	KAHANE Josette	Physique
Mme.	BONNIER Jane	Chimie
MM.	VALENTIN Jacques	Physique
	REBECQ Jacques	Biologie
	DEPORTES Charles	Chimie
	SARROT-REYNAULD Jean	Géologie
	BERTRANDIAS Jean-Paul	Mathématiques Appliquées
	AUBERT Guy	Physique

PROFESSEURS ASSOCIES

MM.	RODRIGUES Alexandre	Mathématiques Pures
	MORITA Susumu	Physique Nucléaire
	RADHAKRISHNA	Thermodynamique

MAITRES DE CONFERENCES

MM.	LANCIA Roland	Physique Atomique
Mme.	BOUCHE Liane	Mathématiques
MM.	KAHANE André	Physique Générale
	DOLIQUE Jean Michel	Electronique
	BRIERE Georges	Physique
	DESRE Georges	Chimie
	LAJZEHOWICZ Joseph	Physique
	LAURENT Pierre	Mathématiques Appliquées
Mme.	BERTRANDIAS Françoise	Mathématiques Pures
MM.	LONGEQUEUE Jean-Pierre	Physique
	SOHM Jean-Claude	Electrochimie
	ZADWORNY François	Electronique
	DURAND Francis	Chimie Physique
	CARLIER Georges	Biologie végétale
	PFISTER Jean-Claude	Physique
	CHIBON Pierre	Biologie animale
	IDELMAN Simon	Physiologie animale
	BLOCH Daniel	Electrotechnique I. P.
	MARTIN-BOUYER Michel	Chimie (C. S. U. Chambéry)
	SIBILLE Robert	Construction mécanique (I. U. T.)
	BRUGEL Lucien	Energétique I. U. T.
	BOUVARD Maurice	HYdrologie
	RICHARD Lucien	Botanique
	PELMONT Jean	Physiologie animale
	BOUSSARD Jean-Claude	Mathématiques Appliquées (I. P. G.)
	MOREAU René	Hydraulique I. P. G.
	ARMAND Yves	Chimie I. U. T.
	BOLLIET Louis	Informatique I. U. T.
	KUHN Gérard	Energétique I. U. T.
	PEFFEN René	Chimie I. U. T.
	GERMAIN Jean-Pierre	Mécanique
	JOLY Jean-René	Mathématiques Pures
Melle.	PIERY Yvette	Biologie animale
	BERNARD Alain	Mathématiques Pures
	MOHSEN Tahsin	Biologie (C. S. U. Chambéry)
	CONTE René	Mesures Physiques I. U. T.
	LE JUNTER Noël	Génie Electrique Electronique I. U. T.
	LE ROY Philippe	Génie Mécanique I. U. T.
	ROMIER Guy	Techniques Statistiques quantitatives I. U. T.
	VIALON Pierre	Géologie
	BENZAKEN Claude	Mathématiques Appliquées
	MAYNARD Roger	Physique

MM.	DUSSAUD René	Mathématiques (C. S. U. Chambéry)
	BELORIZKY Elie	Physique (C. S. U. Chambéry)
Mme.	LAJZEROWICZ Jeannine	Physique (C. S. U. Chambéry)
M.	JULLIEN Pierre	Mathématiques Pures
Mme.	RINAUDO Marguerite	Chimie
MM.	BLIMAN Samuel	E. I. E.
	BEGUIN Claude	Chimie Organique
	NEGRE Robert	I. U. T.

MAITRES DE CONFERENCES ASSOCIES

MM.	YAMADA Osamu	Physique du Solide
	NAGAO Makoto	Mathématiques Appliquées
	MAREZIO Massimo	Physique du Solide
	CHEECKE John	Thermodynamique
	BOUDOURIS Georges	Radioélectricité
	ROZMARIN Georges	Chimie Papetière

A mes parents

Je tiens à exprimer ici ma profonde gratitude à

Monsieur le Professeur Jean KUNTZMANN, *Directeur de l'Institut de Mathématiques Appliquées de Grenoble, dont l'intérêt pour une rénovation pédagogique en Mathématiques m'a été un précieux encouragement, et qui m'a fait l'honneur de présider le jury de cette thèse,*

Monsieur le Professeur Noël GASTINEL, *Directeur de l'Institut de Programmation de Grenoble, dont les conseils éminents ont grandement amélioré le fond et la forme de mon travail,*

Monsieur le Professeur Yves LE CORRE, *Directeur du Laboratoire "Ordinateur Pour Etudiants" de la Faculté des Sciences de Paris, qui le premier en France, a ouvert la voie de la diffusion de l'enseignement par ordinateur,*

Monsieur Jean-Claude BOUSSARD, *Maître de Conférences à l'Institut Polytechnique de Grenoble, qui a dirigé mon travail sans ménager ses conseils et ses encouragements, et lui a imprimé, par la hauteur de vue de ses remarques, un caractère général qu'il n'avait pas au début.*

Messieurs Roger SACCO *et* Georges FAFIOTTE, *Ingénieurs à l'I.M.A.G., pour l'aide efficace qu'ils m'ont apportée dans l'écriture du système,*

Mes collègues de l'Equipe de Programmation pour leurs judicieux conseils, en particulier, MM. Olivier LECARME, Edouard CLEEMANN, Michel LUCAS, Dominique CLAUZEL,

Mademoiselle Josiane QUACCHIA, *MM.* Paul MOUNET, Claude ANGUILLE, *et* Claude LABORIE, *pour l'extraordinaire performance qui a conduit en un temps record à la réalisation matérielle de cet ouvrage.*

INTRODUCTION

I N T R O D U C T I O N

Les hommes se rangent en deux catégories : ceux qu'on appelle "informaticiens" selon un mot à la mode, et les autres. Ces derniers en ont généralement des complexes, jusqu'au jour où à la faveur d'un article de vulgarisation, ils entrent la tête haute dans la première catégorie.

Parmi les informaticiens, on trouve des optimistes invétérés qui sont persuadés que l'ordinateur peut tout faire. Ce sont les plus dangereux, car ils entretiennent des légendes propres à effaroucher les non-spécialistes.

Une de ces légendes, particulièrement tenace, consiste à dire que la machine remplacera peu à peu le professeur dans son métier, et bien sûr, fera mieux que lui. De pareilles affirmations provoquent, on s'en doute, des sentiments de méfiance de la part des enseignants, des éducateurs et des parents, et limitent considérablement la portée des expériences en cours.

Ces dernières ont d'abord vu le jour aux Etats-Unis et, une fois de plus, on est tombé dans le travers courant en informatique qui consiste à refaire avec un ordinateur ce qu'on faisait "à la main" auparavant; Les travaux de SKINNER et de CROWDER sur l'enseignement programmé ont ainsi donné lieu à des transpositions plus ou moins heureuses sur matériel électronique. L'ordinateur dans ce cas, tourne les pages d'un livre, avec ou sans élégance, selon le savoir-faire du réalisateur.

L'avènement des systèmes de programmation conversationnels a permis d'utiliser plus rationnellement la machine, en lui confiant la tâche de poser des questions à l'étudiant. La majeure partie des expériences actuelles en France ont cet aspect de questionnaire lié à la vérification des

connaissances dans une seule discipline. Les résultats de ces expériences sont généralement fragmentaires et ne permettent pas de dégager des stratégies d'ensemble pour l'utilisation de ce nouveau matériel pédagogique qu'est l'ordinateur.

Seul un système général, non lié à l'apprentissage d'une seule matière, ni à un processus didactique figé, doit déboucher sur des résultats significatifs permettant de valider telle ou telle stratégie pédagogique pour telle ou telle discipline enseignée à telle ou telle population d'élèves.

Ces objectifs nous ont conduits à concevoir un système aussi souple que possible pour en faire un outil nouveau à la disposition des enseignants. Comme dans l'utilisation des techniques audiovisuelles, ceux-ci doivent adopter une attitude rénovatrice et bâtir une pédagogie adaptée à ce nouveau moyen. Le langage MAGISTER présenté ici, a été conçu en vue de cette recherche par l'enseignant d'une pédagogie, à travers l'écriture de ses cours.

Le premier module du système, destiné au professeur pour la rédaction de son cours, est un compilateur conversationnel. Le second module est un interpréteur qui contrôle la séance d'apprentissage proprement dite. Le troisième module, destiné aux élèves, est un programme d'assistance qui remédie à une défaillance momentanée.

L'objet du présent ouvrage est de présenter le langage MAGISTER **(Chapitre II)** et d'exposer la méthode de compilation utilisée. **(Chapitre III).** Le **Chapitre I,** après une étude bibliographique, décrit brièvement le système global d'enseignement assisté par ordinateur.

CHAPITRE I

LE SYSTEME GENERAL

endroit quelconque du processus pédagogique, le plus souvent pour "traiter" les réponses des étudiants.

L'Université de Grenoble a sans doute été une des premières à mettre sur pied un procédé de correction automatique de tests {16}. Dès 1960, les étudiants de licence se sont vu proposer des questionnaires comportant des questions à choix multiple. Les réponses étaient portées sur une carte au moyen d'un crayon magnétique. L'ordinateur -un GAMMA ET- après un traitement approprié des cartes, les corrigeait et perforait en dernière analyse une note de 0 à 20 sur la carte elle-même. Les réponses attendues étaient soit une alternative ou un enchaînement, soit un choix entre 10 possibilités au plus, soit une valeur numérique pondérée. Ces tests, d'une fréquence de 15 jours, étaient utilisés conjointement avec d'autres types de contrôle de connaissances, tels qu'interrogations écrites ou orales. Le principe d'un programme général de correction était donc posé, puisque divers types de réponses étaient envisagés.

Il est curieux de constater que, 10 ans plus tard, ce sont les mêmes caractéristiques que l'on retrouve dans nombre de systèmes d'enseignement assisté par ordinateur. Les progrès considérables accomplis aussi bien du côté de la technologie que du côté de la programmation n'ont finalement servi qu'à parfaire la présentation extérieure de l'enseignement. Cela est surtout vrai aux Etats-Unis où l'attrait de la technologie a quelque peu masqué les problèmes de fond qu'on est en droit de se poser dans ce domaine. On a ainsi vu fleurir des expériences spectaculaires comportant tous les raffinements audio-visuels du moment, mais particulièrement paúvres sur des points aussi essentiels que l'analyse des réponses.

1 - Etats-Unis

Quelques laboratoires américains, cependant, développent des systèmes généraux d'enseignement assisté par ordinateur, sans tenir compte de cet engouement, source d'un empirisme quasi-systématique dans la manière de procéder aux expérimentations.

Le laboratoire que dirige à l'Université de Stanford le Professeur SUPPES (Institute for Mathematical Studies in the Social Sciences) a mis au point depuis 1963 plusieurs systèmes différents s'adressant à des populations d'élèves différentes, portant sur des disciplines différentes et opérant sur des matériels différents {32}. C'est certainement au monde le laboratoire qui a le plus d'expériences dans ce domaine. Le premier système conçu fut fondé sur une série d'exercices de mathématiques à l'école primaire. Les enfants ont à découvrir certaines règles telles que l'associativité ou la commutativité de l'addition. Chaque interrogation d'une durée d'environ 3 minutes comporte une vingtaine d'exercices. A l'heure actuelle, environ 5000 enfants de Californie, ainsi que du Kentucky, et du Mississippi pratiquent ces exercices distribués par un ordinateur PDP10 au moyen de télétypes.
Les résultats pédagogiques pour l'année 1965/66 se révèlent très encourageants {31}. A côté de ce système d'exercices (drill-and-practice), existent, toujours à Stanford, des systèmes d'enseignement (tutorial) pour l'apprentissage de la lecture par des enfants de 6 ans et des mathématiques par des enfants de 7 ans. Ici, les terminaux sont très évolués, puisqu'ils comportent un écran cathodique avec un pointeur optique, un projecteur de diapositives et un organe de réponse vocale. Un troisième système est utilisé pour l'enseignement de l'algèbre et de la logique du russe à l'Université, un cinquième, enfin, tout à fait original propose des "colles" de mathématiques élémentaires aux étudiants en leur téléphonant chez eux. Les termes de la colle sont générés par l'ordinateur à partir d'un lexique particulier et transmis oralement à l'étudiant. Celui-ci répond au moyen du petit clavier à touches figurant sur les appareils de téléphone américains.

Une caractéristique de ces systèmes est qu'ils ont été conçus pour une application unique. Ce n'est pas le même programme qui gère l'enseignement des mathématiques et celui du russe. Il n'y a donc pas ici de système général accueillant des disciplines diverses, mais autant de systèmes particuliers qu'il y a d'expériences pédagogiques.

On retrouve cette même obligation d'écriture d'un programme pour chaque cas présenté, dans une autre application, celle dirigée par W. FEURZEIG chez BOLT, BERANEK & NEWMAN. Il s'agit là de ce que l'on est convenu d'appeler

une étude de cas. Les étudiants du Massachussets Hospital de Boston s'entraînent au diagnostic médical avec un malade fictif pour lequel ils ont une observation d'entrée à l'hôpital et une double liste comportant les examens pouvant être demandés et les diagnostics vraisemblables. Le comportement de l'étudiant, ainsi que son passé déterminent les réponses de l'ordinateur. Le degré de liberté laissé à l'étudiant est plus grand ici que dans les processus d'enseignement automatisé décrits plus haut. L'étudiant est plus ou moins libre de déterminer l'ordre des phases de son apprentissage qui doivent le mener au diagnostic judicieux.

Cette liberté est encore plus grande dans les expériences de simulation et de jeu. On demande par exemple à l'étudiant, momentanément promu Ministre de l'Economie, de prendre toutes les décisions propres à juguler une famine de tout un peuple. Ce que l'ordinateur propose ici est donc une situation type devant amener des décisions et des choix de la part de l'étudiant. Il ne s'agit plus tellement d'enseignement automatisé, mais bien plutôt d'apprentissage automatisé avec un simulateur. Ce transfert d'intérêt du professeur vers l'élève semble être une tendance générale des équipes de recherche aux Etats-Unis.

2 - URSS

Les informations fragmentaires que l'on possède sur l'automatisation de l'enseignement en URSS {29} soulignent l'importance que les pédagogues soviétiques accordent à la notion de contrôle dans l'acquisition des connaissances. La plupart des expériences tendent à la maîtrise des mécanismes d'apprentissage de l'élève. LANDA {21} s'attache à formaliser très rigoureusement les processus intellectuels qui interviennent dans la résolution des problèmes. Il tente de décrire sous forme d'algorithmes, les structures internes des mécanismes de la pensée. D'autres équipes s'intéressent également à la définition des moyens de ce contrôle : processus d'analyse de la réponse (DMITRIEW), élaboration de langages appropriés (DOWGJALLO). L'aspect technologique de l'enseignement automatisé n'a pas, ici, pris le pas sur la recherche fondamentale. Les dispositifs utilisés en URSS sont bien souvent des machines à enseigner, plutôt que des ordinateurs, et ce, à la satisfaction des utilisateurs.

3 - Royaume Uni

En Grande-Bretagne, où l'enseignement programmé est largement répandu dans les entreprises {28}, les tendances de la recherche s'orientent vers l'étude de l'apprentissage adaptatif {15, 23}. Un système adaptatif pour l'éducation se caractérise d'après PASK par une interaction individuelle entre l'élève et un mécanisme qui tout d'abord mesure et décrit la performance de l'élève, et ensuite utilise la description ainsi obtenue pour prescrire ou modifier une stratégie d'apprentissage.

4 - Belgique

En Belgique, c'est également dans cette direction que travaillent les équipes de Louvain et de Liège {18, 19}. Les résultats obtenus sont ici de très haut niveau, puisqu'on a obtenu lors d'un test final portant sur l'apprentissage du calcul 97 % de réponses exactes.

5 - Allemagne

En Allemagne, on trouve plusieurs courants de pensée traduisant les préoccupations diverses des équipes de recherche, parmi lesquelles on peut citer POHL qui reprend les idées de LANDA sur les algorithmes en les appliquant à l'enseignement des langues vivantes {27}, et le groupe de la Gesellschaft für Programmierte Instruktion, qui envisage l'utilisation de l'ordinateur, non pas comme un mécanisme contrôlant l'apprentissage mais comme une machine permettant de fabriquer des cours programmés à partir d'un inventaire des concepts et notions à enseigner.

6 - France

Cet intérêt pour l'analyse et la structuration de la matière à enseigner se retrouve en France {8, 26} où l'enseignement programmé tient une place importante à côté de l'enseignement assisté par ordinateur. La

première expérience sur calculateur a vu le jour à la Faculté des Sciences de Paris dans le laboratoire de l'OPE (Ordinateur pour Etudiants) dirigé par Y. LE CORRE {4}. Le système a été conçu non pas dans le but immédiat d'enseigner, mais de vérifier ou de renforcer des connaissances supposées acquises par un apprentissage traditionnel. La discipline de départ était l'Electricité au niveau du second cycle de l'enseignement supérieur. Peu à peu, des questionnaires portant sur les sujets les plus divers ont été élaborés. Un module d'analyse des réponses comportant une égalité accepte toutes les formes équivalentes d'une formule, ce qui est très utile pour les sciences exactes. L'intérêt des étudiants pour ce mode de vérification des connaissances a été considérable.

Les étudiants en informatique de la Faculté des Sciences de Toulouse font leur apprentissage d'ALGOL grâce à un système de temps partagé mettant en jeu un éditeur conversationnel reconnaissant le langage {6}. Le laboratoire d'informatique a d'autre part une expérience ancienne de l'enseignement programmé {7}.

Le Centre d'Etudes et de Recherches Psychologiques "Air" a sous la direction du colonel DE BRISSON mis sur pied un système d'enseignement automatisé de l'arithmétique élémentaire {22}. Un mécanisme d'analyse de formules est utilisé pour tenter de décrire un algorithme de la résolution de problème en vue d'arriver à un diagnostic adéquat.

Le Centre National d'Etudes des Télécommunications de Lannion a également mis au point un système d'enseignement assisté par ordinateur {1} portant sur l'apprentissage du dépannage en électronique, ceci pour les besoins propres des P.T.T.. D'autres sujets ont été abordés, comme le vocabulaire français à l'I.U.T. de Rennes. Les résultats semblent être encourageants et l'intérêt des étudiants considérable.

Dans le projet Jean BERNARD, il s'agit de l'enseignement de l'hématologie à la Faculté de Médecine de Paris {11}. Dans cette expérience, l'accent est mis davantage sur les modèles d'apprentissage et l'analyse statistique des résultats obtenus que sur l'étude d'un langage pour l'écriture des cours.

Cette énumération non exhaustive des expériences françaises en matière d'enseignement automatisé montre une diversification des centres d'intérêt analogue à celle des équipes étrangères. On ne peut pas dire qu'il y ait une tendance propre à la recherche française dans ce domaine. Jusqu'à présent chaque équipe a travaillé dans son laboratoire sans contacts approfondis avec d'autres chercheurs. Depuis 1969, cependant, un désir d'information et de discussion a conduit à la création à Paris d'une U.E.R. à dominante de recherche, ayant pour but l'étude de la didactique des disciplines scientifiques. On peut raisonnablement espérer aboutir en quelques années à une politique concertée et à un plan cohérent pour le développement de ce nouvel outil pédagogique.

C - CHOIX DE LA CONCEPTION DU SYSTEME SELON LES OBJECTIFS

1 - Introduction

Avant d'aborder l'étude des objectifs du système, il est bon de se demander qui va s'en servir. Deux conceptions s'opposent à ce sujet.

Les uns pensent qu'un système d'enseignement assisté par ordinateur est quelque chose qu'il ne faut pas laisser entre toutes les mains et surtout pas dans celles des enseignants. L'argument est qu'il vaut mieux confier à un rédacteur de cours l'exposé du professeur pour en faire un cours programmé. Cette façon de voir les choses a plusieurs défauts. Le principal est qu'elle isole le professeur de son outil pédagogique, un second est qu'elle nécessite un personnage intermédiaire qui se trouve être un obstacle de plus entre le maître et l'élève.

L'autre conception met le professeur en contact direct avec le système, grâce auquel il rédige lui-même son cours. Le défaut majeur est ici que le professeur n'est pas en général informaticien. Il est donc souvent rebuté par les contraintes d'utilisation du système.

Malgré cet inconvénient, nous avons choisi cette seconde manière d'imaginer le rôle du professeur. En effet, si le professeur est rebuté par l'utilisation pratique du système, c'est que le système est mauvais et non adapté à sa "clientèle". D'autre part, l'ordinateur doit être toujours considéré comme un outil pour les enseignants, au même titre que les dispositifs audio-visuels.

C'est en utilisant cet outil, puis en maîtrisant son emploi que les enseignants développeront eux-mêmes cette technique pédagogique, que certains craignent un peu, parce qu'ils la méconnaissent. Donc, l'utilisateur de notre système sera un enseignant, la plupart du temps non informaticien. Ceci va conditionner bien sûr le genre d'objectifs à atteindre.

2 - Objectifs

Le plus important est la souplesse du système. Le professeur doit trouver la rédaction de son cours agréable. Ceci ne peut être obtenu que par un système conversationnel, au niveau de l'écriture du cours. Le maître se sert alors d'un terminal pour introduire son cours en machine. Les erreurs inévitables qu'il commet doivent être signalées sur le champ et conduire à une correction immédiate. Le langage pour l'écriture des cours doit répondre lui aussi à certaines préoccupations qu'on étudiera dans le chapitre suivant. La compilation de ce langage doit être conversationnelle et permettre non seulement d'introduire un cours en machine, mais de le corriger et ceci à tout moment, même pendant une séance d'apprentissage avec les élèves. Un outil possédant ces caractéristiques est propres à notre avis, à intéresser les utilisateurs.

Vu du côté de l'élève maintenant, le système doit conserver cette souplesse. Cela nécessite des possibilités d'interruption de la séance d'apprentissage par l'élève pour effectuer certains calculs, ou poser lui-même des questions au système.

L'accès à des ressources externes est très intéressant aussi bien pour le professeur que pour l'étudiant. Par exemple on ne saurait concevoir l'apprentissage d'un langage de programmation sans permettre l'accès du professeur et de l'étudiant, au compilateur reconnaissant ce langage. Il en va de même pour des exercices de calcul, pour lequel l'accès de l'élève à un programme du genre "calculateur de bureau" est essentiel.

Un autre objectif important est de pouvoir diffuser plusieurs cours différents en même temps. En effet, il est dommage de limiter une séance d'apprentissage à une matière déterminée, ce qui va à l'encontre de la notion de libre-service pour les étudiants.

Enfin, une dernière nécessité est de ne pas perturber par les séances d'enseignement, le fonctionnement quotidien du centre de calcul. De deux choses l'une:ou bien on dispose d'un ordinateur uniquement destiné à l'enseignement, ou bien celui qu'on veut utiliser est déjà dévolu à d'autres tâches, et il faut alors s'insérer au mieux dans le cadre de travail existant. On peut réserver des tranches d'heures aux séances d'apprentissage, mais hélas, les horaires scolaires et professionnels coïncident parfaitement. Dans les pays s'étendant sur plusieurs fuseaux horaires, le partage de l'ordinateur pose moins de problèmes. Ainsi aux Etats-Unis, grâce au décalage horaire, une même machine peut servir des utilisateurs de la côte Est et d'autres de la côte Ouest, et ceci aux heures actives chez les uns et les autres. Dans un petit pays c'est impossible, et il faut donc envisager un schéma de partage de temps qui permette simultanément l'usage de l'ordinateur, aux chercheurs du centre de calcul et aux usagers (professeurs et étudiants) du système d'enseignement assisté.

Ces divers objectifs nous ont conduit à imaginer le système d'enseignement assisté, sur l'ordinateur IBM 360/67 dans l'environnement des systèmes CP et CMS {20}.

3 - CP/CMS

Ce couple de systèmes a été mis au point par le Centre Scientifique IBM de Cambridge (Massachussets) pour permettre l'accès conversationnel d'un grand nombre d'utilisateurs à un 360. La méthode, tout à fait originale, consiste à simuler à chaque terminal relié au 67, un pupitre de 360 standard.

Ainsi chaque utilisateur agit comme s'il avait devant lui un ordinateur, son ordinateur. En effet le programme CP est un générateur de machines virtuelles" et ne simule pas seulement le pupitre, mais tous les organes d'une machine (unité centrale, mémoire centrale, disques, bandes, lecteur de cartes, imprimante, etc ...). Il va de soi que les machines virtuelles sont indépendantes les unes des autres et que les utilisateurs peuvent s'ignorer totalement. Mais ils peuvent aussi correspondre et se transmettre des fichiers. Le rôle de CP consiste donc uniquement à générer et à gérer des machines virtuelles.

Ces machines peuvent être maintenant utilisées avec n'importe quel système de programmation. Par exemple le système d'exploitation classique d'un 360 : OS, peut être installé sur une machine virtuelle et gérer un flot de travaux comme sur une machine réelle. Bien entendu, les listes de programmes devant sortir sur l'imprimante virtuelle de cette machine, sortiront en définitive sur l'imprimante réelle du 67. C'est CP qui se charge de la gestion des entrées-sorties en fonction du nombre de machines virtuelles, de la taille de la mémoire réelle, du tambour et des disques, sans se préoccuper du genre de système de programmation installé sur une machine virtuelle.

Le système de programme CMS a été conçu pour permettre un dialogue avec l'utilisateur à son terminal. CMS peut fonctionner sur n'importe quel 360 standard réel, et donc ignore totalement ce que fait CP. Cette séparation des pouvoirs entre CP et CMS fait l'originalité de la méthode et accroît la sécurité et la souplesse d'emploi.

La programmation sous CMS se fait au moyen de commandes écrites au terminal et qui appellent certaines ressources telles qu'un compilateur, un éditeur, un chargeur ou tout programme de l'utilisateur. Une fois que le programme appelé a terminé son travail, l'utilisateur peut introduire une autre commande et ainsi de suite. Notre système d'enseignement assisté se trouve à ce niveau de hiérarchie. Le professeur a donc, grâce à CP, une machine virtuelle à sa disposition, dans laquelle, grâce à CMS , il appelle le système d'enseignement au moyen d'une commande. Une fois la commande reconnue, c'est la première phase de ce système qui prend le contrôle et qui assiste le professeur dans la rédaction de son cours.

L'étudiant a également une machine virtuelle à sa disposition dans laquelle, par le moyen d'une autre commande il appelle la phase du système qui gère son apprentissage. Cette seconde phase consiste évidemment à présenter le cours à l'élève et à récupérer et analyser ses réactions. Celui-ci peut, à tout moment, interrompre le déroulement de cette phase d'apprentissage pour appeler un programme d'assistance ou toute autre ressource externe au système d'enseignement (compilateur, calculateur de bureau, etc ...).

Remarque :

Cette description sommaire du système peut faire penser qu'il est très lié à l'environnement de CP/CMS. En fait, la tâche de diffuser une ressource à plusieurs utilisateurs peut être accomplie par un programme beaucoup plus simple et petit. L'adaptation de notre système à un 360 standard muni d'un tel programme distributeur prendra quelques mois. La ressource à diffuser est alors composée de l'interprète et du programme objet. La notion de machine virtuelle permet la mise au point assez rapide d'un système nouveau et c'est une des raisons de notre choix.

La conception adoptée pour notre système d'enseignement et décrite plus haut répond bien aux objectifs initiaux :

- insertion dans une organisation existante permettant aux chercheurs du centre de calcul de continuer leurs travaux sans heures creuses.

- possibilité de diffuser simultanément des cours différents.

- rédaction et modification du cours"à la demande"avec l'aide du système.

- possibilité d'appeler des ressources extérieures pour le professeur et pour l'étudiant.

Il s'agit de voir maintenant comment peuvent être techniquement réalisées ces fonctions.

D - DESCRIPTION TECHNIQUE DU SYSTEME (voir schéma page 18).

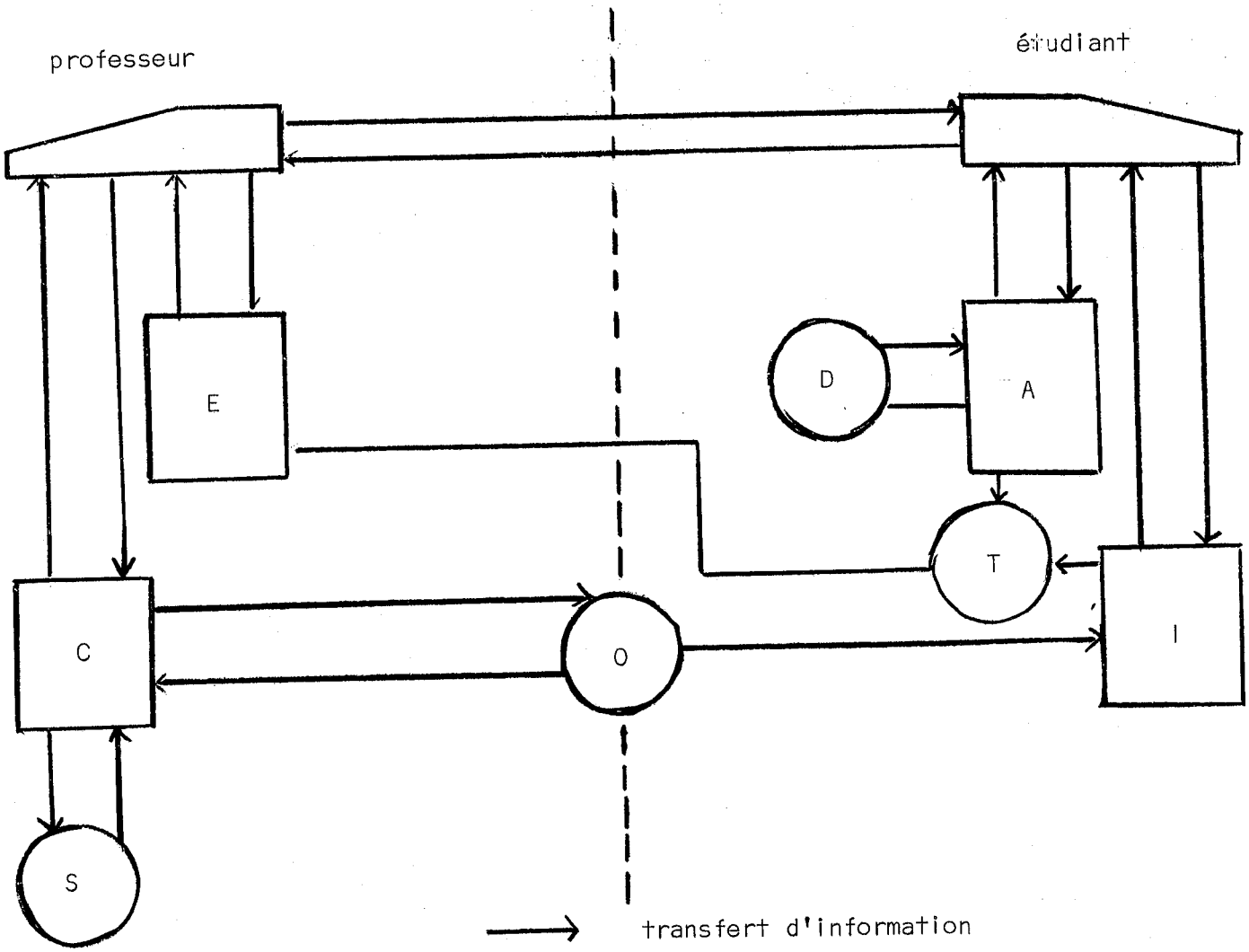
Il y a au moins deux phases à réaliser : l'écriture du cours et sa présentation. La première se déroule dans la machine virtuelle du professeur, la seconde dans la machine virtuelle de l'élève. Le cours est écrit dans un langage spécialisé, MAGISTER, étudié au chapitre suivant, un compilateur conversationnel C crée à partir des instructions écrites par le professeur, un fichier source S et un fichier objet O résultat de la compilation.

Ces fichiers source et objet peuvent être modifiés au gré du professeur. Lors de la séance d'apprentissage, le fichier objet est rendu accessible à partir des machines virtuelles des élèves dans lequelles il est interprété. Les données de ce programme objet sont constituées par les chaînes de texte fournies par l'étudiant à son terminal. C'est à ce terminal que seront imprimés les messages construits par le professeur et qui constituent donc les sorties du programme objet.

Le programme interpréteur I assure donc le dialogue avec l'étudiant et crée un fichier T qui est la trace de la conversation au terminal de l'élève. Ce dernier peut appeler un programme d'assistance A qui grâce à un dictionnaire de mots-clés D lui fournit des indications sur la discipline étudiée. Le fichier T porte la trace de l'assistance fournie. Le professeur a accès à ce fichier trace au moyen d'un programme d'examen E. D'autre part le professeur et l'étudiant peuvent correspondre directement comme tous les utilisateurs de CP. Telles sont les liaisons possibles entre les machines virtuelles du maître et de l'élève.

On reviendra au chapitre III sur la manipulation des fichiers, à propos de la méthode de compilation.

Sur le plan technologique, les terminaux utilisés sont ceux acceptés par CP/CMS, c'est-à-dire les consoles IBM 2741, les écrans de visualisation alphanumériques IBM 2260 et les télétypes ASR33. Les écrans permettent d'afficher dix lignes de texte en une seule fois, mais l'information qu'ils présentent est fugitive et l'étudiant ne peut pas emporter une trace de sa séance. Cette trace est évidemment fournie par les terminaux de type machine à écrire. En contre partie, ceux-ci sont en général très bruyants et leur utilisation n'est pas à conseiller pour de jeunes enfants. Il est donc utile de disposer de plusieurs genres de terminaux sur un même système.



C compilateur
E examen
I interprète
A assistance

S source
O objet
T trace
D dictionnaire

Liaisons entre l'écriture du cours et sa présentation

CHAPITRE II

C H A P I T R E II

DEFINITION DU LANGAGE MAGISTER

A - OBJECTIFS VISES

On peut se demander quelle est la justification des langages d'écriture de cours. En effet, il y a déjà tellement de langages de programmation qu'il suffisait d'en utiliser un dans un contexte d'enseignement automatisé. Le langage qui s'y prête le mieux semble être **SNOBOL 4.** On y retrouve en effet tout l'ensemble d'instructions dont on a besoin pour morceler, concaténer des chaînes de texte, ou rechercher des modèles. Malheureusement sa syntaxe en fait un outil peu utilisable par un non-spécialiste de l'informatique. Or, il est nécessaire de confier au professeur lui-même, la tâche de rédiger son cours si l'on veut répandre ce mode d'enseignement. On est donc conduit à définir un nouveau type de langage, adapté à cette tâche pédagogique.

Les principales qualités à attendre d'un tel langage sont développées ici :

1 - Le langage doit être le plus général possible, c'est-à-dire qu'il doit pouvoir accueillir des cours très divers quant à leur fond, leur forme et aussi leur articulation pédagogique. Il ne doit pas être lié ni à une discipline, ni à une stratégie particulières.

2 - Il doit être facile à apprendre par les professeurs qui l'utiliseront. Ceux-ci ne sont pas des informaticiens et sont vite rebutés par une syntaxe et une sémantique compliquées. Cependant, il ne faut pas non plus tomber dans l'excès contraire qui consiste, sous prétexte de simplicité, à bâtir un langage inefficace parce que trop pauvre. Un juste équilibre doit donc être trouvé entre ces deux objectifs.

3 - Le langage d'écriture de cours doit être rapidement extensible pour permettre de nouvelles structures de données ou de nouvelles opérations, nécessitées par les besoins de la stratégie pédagogique adoptée. Ceci implique que la définition du langage soit modulaire. C'est la technique d'interprétation qui semble alors la mieux adaptée à cet objectif. Il est en effet aisé, dans ce mode de traduction d'ajouter de nouveaux modules ayant d'autres finalités que ceux déjà existants. Par exemple, le professeur désire disposer pour son cours d'un module d'analyse de la réponse différent de celui qu'il utilisait jusqu'à présent. L'informaticien doit alors pouvoir rapidement étendre la puissance du langage dans le sens désiré.

4 - L'utilisation pratique du langage par le professeur doit être commode, et ceci peut être obtenu par une compilation conversationnelle. Le professeur rédige alors son cours au terminal, avec l'appui d'un compilateur incrémentiel qui détecte les erreurs syntaxiques et sémantiques au fur et à mesure de leur apparition.

5 - Le langage doit permettre la rédaction de cours dont la diffusion ne s'avèrera pas monotone à l'élève. Cela peut être obtenu par un processus aléatoire, portant sur les quantités manipulées et aussi sur les séquences d'instructions du cours lui-même.

Certains de ces objectifs, mais pas tous, se retrouvent dans quelques langages d'écriture de cours déjà existants. Presque toujours le langage a été conçu en vue de l'apprentissage d'une matière bien précise et les opérations qu'il permet s'en ressentent. Fréquemment, cet apprentissage a également été planifié par le réalisateur et le langage ne peut être utilisé que dans une stratégie pédagogique donnée (le plus souvent, il s'agit de questions à choix multiple). C'est certainement le plus grave défaut, car l'enseignant ne peut pas sortir du cadre rigide qui lui est imposé. L'outil qu'on lui propose alors est plus un frein qu'un moteur. D'autre part, il n'y a presque jamais de compilation conversationnelle, ce qui interdit au professeur de vérifier

son cours immédiatement, en jouant le jeu de l'élève. Enfin certains langages pèchent soit par excès de simplicité, soit au contraire par une complexité analogue à celle des langages de programmation habituels, ce qui éloigne les non-informaticiens. On trouvera dans {10, 14, 34} des exemples intéressants de langages d'écriture de cours et dans {12, 35} des exemples de langages conversationnels, destinés cette fois-ci à l'étudiant.

B - DESCRIPTION DES INSTRUCTIONS DU LANGAGE

Un cours écrit en langage MAGISTER est composé d'un ou plusieurs chapitres, chaque chapitre correspondant à une unité pédagogique à présenter en une séance d'apprentissage. Des liaisons peuvent cependant intervenir d'un chapitre à un autre, ceci pour les besoins de la progression de l'élève dans le cours. Le chapitre constitue donc l'unité de programme source, qui une fois, compilée fournit un programme objet transmis à l'interprète. Ce dernier rend actif le programme objet dont les données sont fournies par l'élève à son terminal.

Un chapitre est constitué d'une suite d'instructions dont l'exécution s'effectue en séquence, sauf indication contraire, c'est-à-dire dans le cas d'une rupture de séquence. Les instructions sont séparées les unes des autres par un signe de ponctuation qui est soit un point, soit un point-virgule.

1 - Instructions de rupture de séquence

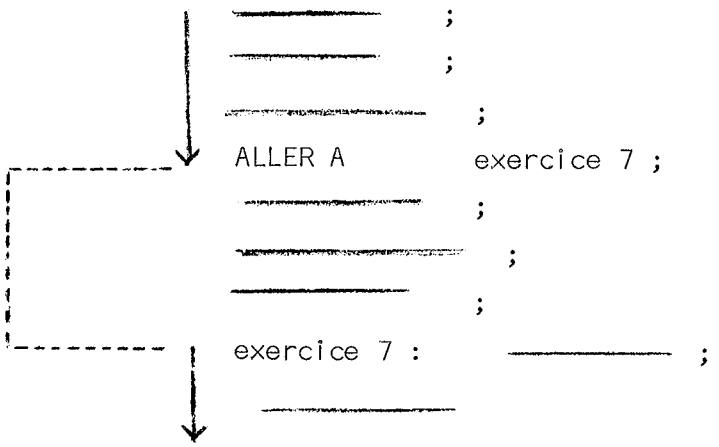
Toutes les instructions sont étiquetables, pour permettre des ruptures de séquence lors de l'exécution du programme. Les étiquettes qui sont une combinaison de lettres et de chiffres commençant obligatoirement par une lettre peuvent être simples ou multiples.

Exemples :

exercice 7 : addition : (instruction) ;

Les ruptures de séquence sont de deux types.
Le plus simple est un transfert direct vers l'instruction étiquetée. Le symbole de base correspondant est ALLER A.

Exemple :



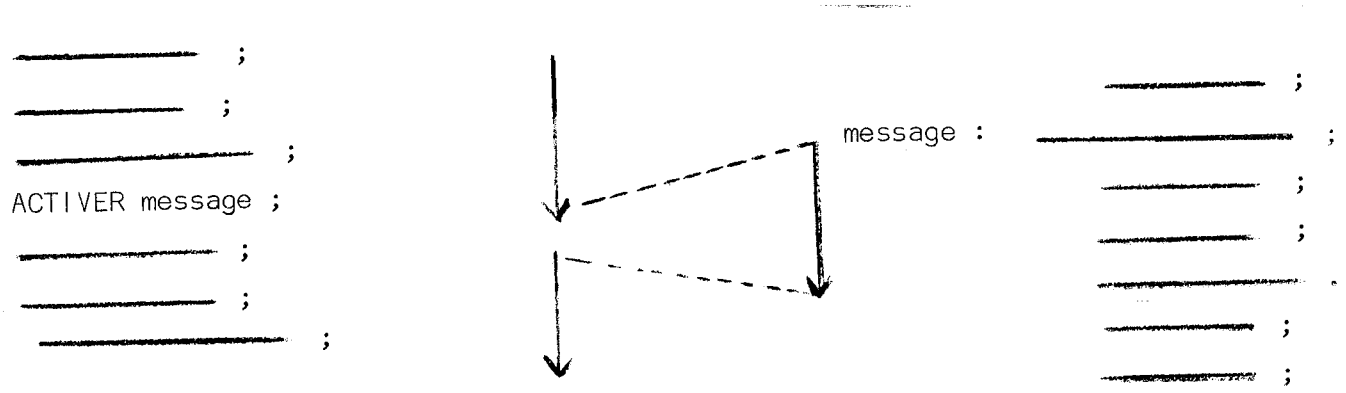
Lors de l'exécution de l'instruction de transfert, un saut se produit vers l'étiquette exercice 7 et le programme continue de se dérouler à partir de cet endroit. Une étiquette ne peut pas figurer deux fois dans le même chapitre, cela va de soi. Les ruptures de séquence peuvent s'effectuer soit vers l'avant comme c'est le cas dans l'exemple soit vers l'arrière.

Remarque :

Cette instruction est l'analogue de la directive "Allez à la page 92" ou "voir l'item 19 A" qu'on rencontre dans les livres brouillés.

Le second type de rupture de séquence est un transfert suivi d'un retour. Le symbole de base correspondant est ACTIVER.

Exemple :



Cette instruction correspond à une activation de sous-programme. En effet l'instruction

ACTIVER message ;

provoque à l'exécution, un transfert vers l'étiquette message, et le programme se déroule alors à partir de cette étiquette jusqu'à l'occurrence d'un point (.) comme symbole de ponctuation terminant une instruction. A ce moment là, un retour s'effectue vers l'instruction qui suit l'ordre ACTIVER. Le premier point rencontré joue donc le rôle de fin de sous-programme et l'instruction qu'il termine est la dernière exécutée avant le retour.

Cet ordre ACTIVER permet d'exécuter plusieurs fois dans un programme une même séquence d'instructions, ce qui se produit fréquemment dans un cours programmé.

Remarque :

Il se peut qu'un point soit rencontré sans qu'il y ait eu au préalable d'instruction ACTIVER. Dans ce cas, le rôle du point est le même que celui du point-virgule qui est une simple ponctuation. Donc, lorsqu'on rencontre un point, deux situations sont à envisager :

a) l'exécution de la séquence d'instructions dépend d'un ordre ACTIVER, et alors l'exécution se poursuit vers l'instruction qui suit cet ordre.

b) l'exécution de la séquence d'instructions ne dépend pas d'un ordre ACTIVER, et alors l'exécution se poursuit normalement en séquence, après le point.

Les instructions de rupture de séquence peuvent avoir comme opérandes non plus seulement une étiquette mais une liste d'étiquettes, séparées les unes des autres par le symbole | .

Exemples :

```
ALLER A addition | e1 | somme ;
ACTIVER erreur | faute ;
```

Lors de l'exécution, le transfert se fera vers l'une des étiquettes spécifiées dans la liste, le choix de cette étiquette se faisant au hasard à l'exécution.

Cette notion de liste d'étiquettes sera utilement mise à profit lorsque l'enseignant voudra présenter à l'élève un exercice pris au hasard parmi une classe d'exercices ayant la même finalité pédagogique. Cette notion se retrouvera dans les instructions d'affectation et d'écriture avec les listes d'expressions. Elle est destinée à rompre la monotonie du dialogue. Si le professeur a rédigé un grand nombre d'exercices portant sur le même sujet et que ces exercices sont repérés par des étiquettes figurant dans une liste aléatoire, il y a peu de chances pour que le même exercice soit présenté deux fois de suite à l'élève. Il y a aussi peu de chances pour que deux élèves voisins aient à traiter le même exercice, ce qui limite l'influence du groupe.

2 - Instruction préfixée

Certaines instructions du langage MAGISTER ont pour effet de vérifier la satisfaction d'un test. Ce sont : l'instruction de comparaison qui compare la dernière réponse fournie avec une liste de réponses-types, l'instruction de recherche qui vérifie qu'une chaîne donnée apparaît bien dans une réponse, l'instruction de lecture en temps limité qui attend dans un délai donné un message de l'élève. Selon les cas, le test sera ou ne sera pas satisfait : oui , l'étudiant a répondu en moins de 20 secondes ; non il n'a pas répondu en moins de 20 secondes. Un préfixe, placé devant une instruction fait dépendre son exécution de la vérification de ce test.

Les deux préfixes utilisés sont :

— (oui, le test est satisfait)
¬ (non, le test n'est pas satisfait)

Exemples :

```
    LIRE EN 20 SECONDES ;
¬ ECRIRE "Plus vite !" ;
```

si l'élève ne répond pas dans le délai de 20 secondes il verra s'imprimer au terminal le message : Plus vite !. S'il est "dans les temps", l'instruction d'écriture sera sautée.

Remarques :

a) la vérification d'un test par un préfixe ne supprime pas la permanence de ce test.

b) la permanence d'un test est maintenue jusqu'à l'apparition d'une instruction présentant une alternative, comme celles que nous avons vues plus haut (lecture en temps limité, comparaison, recherche).

3 - Instructions conditionnelles

L'exécution d'une instruction peut dépendre de la vérification d'une condition plus élaborée qu'un simple test. On a alors affaire à une instruction conditionnelle dont le format est :

SI <expression booléenne> ALORS <instruction inconditionnelle>

ou encore

SI <expr. bool> ALORS <instr. incond.> SINON <instr. inconditionnelle>

Une expression booléenne est l'analogue de celle qu'on rencontre en ALGOL 60, à ceci près qu'il n'y a pas de variables booléennes explicites et qu'on se ramène toujours à la vérification d'une relation entre deux expressions arithmétiques.

Remarque :

Une intruction conditionnelle peut très bien être préfixée. Si le test vérifié par le préfixe est satisfait, on examine la condition suivante introduite par le SI. S'il n'est pas satisfait, l'instruction dans son ensemble n'est pas exécutée, et on passe à l'instruction suivante.

4 - Instruction d'affectation

Les quantités traîtées dans le langage sont toutes de type chaîne. Ce sont soit des variables repérées par leur nom, soit des constantes. Dans les deux cas leur "valeur" est une succession de caractères de longueur quelconque, éventuellement nulle. Les constantes sont écrites en encadrant le texte de guillemets.

Exemples :

a) "Ceci est une constante"

b) "En voici une autre"

c) ""

Le troisième exemple montre une chaîne de longueur nulle. Les constantes peuvent être numériques. Dans ce cas on peut se passer des guillemets : "12" et 12 représentent la même constante. Les opérations arithmétiques usuelles peuvent s'appliquer à ces quantités numériques. Ainsi, le résultat de l'évaluation de

12 + 7

est une chaîne dont la valeur est 19. Les opérations numériques se font en décimal.

Les opérandes arithmétiques peuvent évidemment être des identificateurs de variables. L'évaluation de l'expression a - b conduit à une chaîne constituée des chiffres composant le nombre résultat de la soustraction. Si a (ou b) n'est pas numérique l'opération n'est pas effectuée et un message est transmis au terminal. L'opérateur de concaténation, &, a une priorité inférieure à celle des opérateurs arithmétiques. Ainsi, l'expression
10 & 5 + 7 a pour valeur 1012 alors que (10 & 5) + 7 vaut 112.

L'instruction d'affectation permet de donner à une variable repérée par son nom, une valeur qui est soit une chaîne de caractères soit le résultat de l'évaluation d'une expression soit une combinaison des deux. Le symbole de base correspondant est := .

Exemples :

```
a := "votre réponse est incorrecte" ;
a := b + c / (a - b) ;
a := "la somme de "& m &" et de "& n &" vaut "m + n ;
```

De même que les étiquettes, les opérandes de l'affectation peuvent être groupées en liste aléatoire. La valeur qui est affectée est alors prise au hasard parmi les valeurs des différentes expressions.

Exemples :

a) texte := "Bravo !" | "Très bien" ;

b) n := - 1 | 0 | 1 | 2 | 3 | 4 ;

Ce dernier exemple peut être utile lorsqu'on veut faire varier les données d'un exercice à présenter plusieurs fois.

Remarque :

Il n'y a pas dans MAGISTER de déclarations explicites. La première occurrence d'une variable constitue en fait sa déclaration.

5 - Instruction d'écriture

Elle possède les mêmes opérandes que l'instruction d'affectation. Seule l'exécution diffère. Au lieu d'attribuer une valeur à une variable, l'instruction fera s'imprimer cette valeur au terminal.

Exemples :

a) ECRIRE "vous vous êtes trompé "& n &" fois aujourd'hui" ;
Si n vaut 3, le texte :
vous vous êtes trompé 3 fois aujourd'hui s'imprimera au terminal.

b) ECRIRE "Plus vite !" | "vous traînez." | "vous êtes lent." ;
l'un ou l'autre de ces 3 messages s'imprimera au terminal, le choix étant toujours fondé sur un critère aléatoire lors de l'exécution.

c) ECRIRE "le résultat est" & a * b + c * d ;
le résultat de l'évaluation de a * b + c * d sera imprimé après le texte :

le résultat est .

6 - Instruction de lecture

Elle sert à recevoir la réponse de l'élève lorsqu'il la donne au terminal. L'instruction a plusieurs formes possibles. La plus simple est :
LIRE ;

Le texte fourni par l'élève est alors retenu dans une zone spécifique qu'on désignera ici par l'appelation : "dernière réponse". On peut avoir accès à cette dernière réponse par les instructions de comparaison et de recherche.

On peut indiquer dans cette instruction de lecture un ou deux opérandes :
LIRE a ;

La réponse fournie par l'élève constitue la nouvelle valeur de la variable a.
LIRE EN 30 SECONDES ;

Un délai maximum de 30 secondes est imparti à l'élève pour fournir sa réponse. Une fois les 30 secondes écoulées le clavier est bloqué. Cette limitation facultative du temps de réponse donne lieu à la satisfaction ou à la non-satisfaction d'un test qu'on peut vérifier ensuite au moyen d'une instruction préfixée.

Ces deux types d'opérandes peuvent être combinés :
LIRE réponse EN 1 MINUTE ;

7 - Instruction de comparaison

Elle sert à comparer les valeurs de deux expressions. La première expression peut être soit la dernière réponse fournie par l'élève, et le modèle de l'instruction est alors :
COMPARER AVEC a + b ;

soit une expression donnée explicitement :
COMPARER a & "X Y Z" AVEC C ;

Dans ces deux formats, la seconde expression peut être remplacée par une liste d'expressions.

Exemple :

COMPARER AVEC 17 | a + b ;

Dans ce cas, on compare la valeur de la dernière réponse fournie avec 17 ; Si ce n'est pas 17, on la compare alors au résultat de l'évaluation de a + b. Pour tester le résultat de la comparaison, on se sert d'une instruction préfixée.

Exemple :

COMPARER AVEC "oui" ;
- ALLER A correct ;

8 - Instruction de recherche

C'est l'instruction la plus complexe du langage. Son rôle essentiel est de chercher dans une chaîne l'occurrence d'une sous-chaîne. La chaîne qui constitue le champ d'action de l'instruction peut être soit la dernière réponse fournie par l'élève et alors le format est :

CHERCHER <modèle de recherche> :

soit une chaîne donnée explicitement par son nom :

DANS a CHERCHER <modèle de recherche> ;

Dans les deux cas le déroulement de l'instruction est le même. Le modèle de recherche peut contenir des quantités variées. Le cas le plus simple est :

```
CHERCHER "A B C" ;
```

La dernière réponse fournie est parcourue de gauche à droite pour voir si la chaîne A B C y figure. Si elle y est, le test est satisfait. Si elle ne s'y trouve pas, la recherche n'est donc pas fructueuse et la vérification du test s'effectue par une instruction préfixée.

Exemple :

```
CHERCHER "oui" ;
— ECRIRE "non" ;
```

A la place d'une chaîne donnée explicitement, on peut voir apparaître un nom de variable.
CHERCHER a ;
c'est évidemment la valeur de a que l'on cherche.

Plusieurs quantités peuvent figurer dans un modèle de recherche, séparées par un opérateur de concaténation

```
DANS a CHERCHER b & "kilogrammes" & c ;
```

L'opérateur & est utilisé pour une concaténation stricte, c'est-à-dire une juxtaposition effective de deux opérandes. Un autre symbole, la virgule, indique une "concaténation élastique" entre deux opérandes.

Exemple :

```
CHERCHER "A B C" , "X Y Z" ;
```

On cherche les chaînes de texte A B C et X Y Z séparées par un contexte de longueur quelconque, éventuellement nulle. Avec ce modèle, les réponses

A B C X Y Z

A B C L M N O P X Y Z

A B C < % $ + X Y Z

seront acceptées et la recherche sera jugée fructueuse. Le contexte correspondant est la chaîne vide dans le premier cas, la chaîne L M N O P dans le second, et la chaîne < % = $ + dans le troisième.

La virgule joue donc dans un modèle de recherche le rôle d'opérateur de "proximité" sans imposer une concaténation stricte. Une autre façon d'expliquer son rôle est de dire que la virgule représente un contexte de longueur quelconque entre deux morceaux de la chaîne. C'est dans ce cas, un contexte auquel on ne s'intéresse pas et qui ne sera plus utilisé par la suite.

Si le professeur désire donner un nom à un contexte il peut le faire en indiquant ce nom entre apostrophes dans le modèle de recherche.

Exemple :

CHERCHER "A B C" & 'a' & "X Y Z" ;

Dans ce cas, si les chaînes A B C et X Y Z figurent dans cet ordre dans la dernière réponse fournie, l'instruction CHERCHER effectue un morcellement de cette réponse et affecte à la variable a la valeur de la chaîne de texte comprise entre A B C et X Y Z. Il se peut que cette chaîne soit vide si A B C et X Y Z sont juxtaposés, cela n'est pas gênant.

Un contexte peut être utilisé plusieurs fois dans un modèle.

Exemple :

```
DANS a CHERCHER 'x' & 17 & x ;
```

On cherche ici le nombre 17. Si on le trouve, on appellera x la chaîne qui précède 17 dans a. Puis on cherche cette nouvelle valeur de chaîne qui vient d'être affectée à x.

Si a vaut 2172, alors la recherche est fructueuse et d'autre part la variable x a pour valeur 2. Si a vaut 3172, alors la recherche n'est pas fructueuse et l'affectation à x n'est pas faite. L'instruction de recherche effectue donc un morcellement de la chaîne examinée et une ou plusieurs affectations à des contextes, en cas de succès.

Exemple :

```
DANS a CHERCHER 'b' & " " & 'a' ;
```

Ici on cherche dans a un blanc. Le premier rencontré de gauche à droite fait l'affaire. Ce qui est à gauche de ce blanc est un contexte appelé b. ce qui est à droite est également un contexte à qui on donne le nom a de la chaîne initiale. Cette instruction aura donc pour effet de raccourcir la chaîne a de tous les caractères précédant le premier blanc. C'est un moyen commode pour isoler le premier mot d'une chaîne. Ici, ce premier mot est appelé b et le reste de la chaîne après le blanc est la nouvelle valeur de a.

Ainsi, si a est la chaîne :
"un deux trois quatre"
après l'exécution de l'instruction ci-dessus,
b vaudra "un"
et
a vaudra "deux trois quatre".

9 - Contraintes sur les contextes

On peut indiquer un certain nombre de contraintes aux contextes figurant dans les modèles de recherche. Par exemple on peut indiquer une longueur : 'a , 3' est un contexte ayant 3 caractères.

Exemple :

Dans l'instruction

CHERCHER 'a , 3' & 12 ;

on cherche 12 immédiatement précédé de 3 caractères. Si on trouve ce qu'on cherche, alors on appellera a cette chaîne de 3 caractères.

On peut indiquer comme longueur dans un contexte, la longueur d'une variable.

Exemple :

CHERCHER "x + y" & 'a , ! b' ;

On cherche ici la chaîne x + y immédiatement suivie d'un contexte de même longueur que b. L'opérateur ! placé devant un nom de variable indique la longueur de cette variable en nombre de caractères. Si la recherche est satisfaite la valeur du contexte de même longueur que b est affectée à la variable a.

Il peut arriver que seule la longueur d'un contexte soit importante, et qu'on se désintéresse de son contenu. Dans ce cas, il est possible d'indiquer seulement la longueur du contexte, sans préciser le nom de ce contexte.

Exemple :

CHERCHER 12 & '4' ;

On cherche le nombre 12 immédiatement suivi de 4 caractères, sans se préoccuper de savoir quels sont ces caractères.

Enfin, un autre type de contrainte est d'imposer au contexte d'être numérique, c'est-à-dire uniquement composé de chiffres, ou au contraire alphabétique.

Exemples :

CHERCHER '< n >' ;

On cherche un nombre dans la réponse; le premier rencontré, de gauche à droite fait l'affaire et est affecté à n.

CHERCHER '(a , ! b)' & "X Y Z" ;

On a ici un contexte alphabétique a de même longueur que b.

Sous-chaînes

Pour toute chaîne a, on peut parler des sous-chaînes de a composées de la façon suivante :

a(3, 7) .

Il s'agit d'une chaîne formée des 3°, 4°, 5°, 6° et 7° caractères de a.

10 - Fonctionnement de l'instruction CHERCHER à l'interprétation

(Voir organigramme page 42)

L'instruction CHERCHER possède comme on l'a vu deux opérandes : l'un est l'argument de la recherche, c'est-à-dire la chaîne dans laquelle on cherche quelque chose, l'autre est le modèle, c'est-à-dire une description de ce que l'on cherche. Lors de l'interprétation de l'instruction, on effectue une espèce de recouvrement de la chaîne argument par le modèle.

a) construction du modèle

Cette construction se fait à l'interprétation puisque les quantités figurant dans le modèle sont soit des constantes, soit des variables auxquelles on vient d'affecter une valeur. Le modèle peut contenir deux sortes de champs bien distincts : d'une part les quantités fixes, d'autre part les contextes. Ainsi pour l'instruction

DANS x CHERCHER 'a,4' & "A B C" , b ;

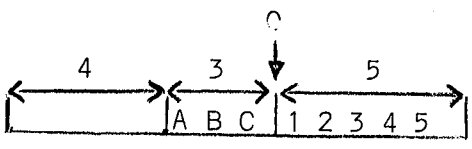
les quantités fixes sont A B C et la valeur de b ;

les contextes sont 'a, 4' et la virgule précédant b.

les champs fixes ont toujours une longueur bien déterminée

(3 pour A B C , !b pour b).

La longueur des contextes est soit imposée (4 pour a), soit quelconque et éventuellement nulle (pour la virgule). Le modèle sera composé de plusieurs morceaux ou champs ayant ces longueurs respectives. Pour un contexte dont la longueur n'est pas imposée, le champ aura une longueur nulle. Dans l'exemple ci-dessus, on aura donc pour modèle :



si b a pour valeur 1 2 3 4 5.

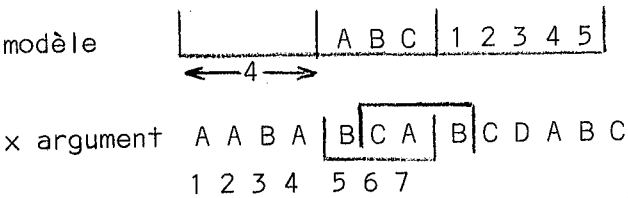
Ainsi le modèle a pour longueur la somme des longueurs des parties fixes et des longueurs imposées aux contextes, s'il y en a. A ce modèle est associé un descripteur qui indique quels sont les divers champs.

b) test de la longueur

Une fois le modèle fabriqué on compare sa longueur à celle de la chaîne argument. Si elle lui est supérieure c'est que la recherche ne peut pas être fructueuse. En effet il ne peut pas y avoir d'inclusion du modèle dans quelque chose qui lui est plus petit.

c) recherche des champs fixes

Si le test de la longueur est satisfait, on commence à rechercher le premier champ fixe. Pour cela, on effectue une comparaison de ce champ fixe de modèle avec une sous-chaîne de même longueur dans la chaîne argument. Le début de cette sous-chaîne est déterminé par la longueur du contexte précédant le champ fixe s'il y en a un. Ainsi dans l'exemple ci-dessus, la comparaison de A B C se fera avec la sous-chaîne composée des 5°, 6° et 7° caractères de x, puisque la longueur imposée au contexte a est 4.



Si les deux opérandes ne sont pas identiques, on continue la comparaison avec la sous-chaîne d'à côté dans l'argument et ainsi de suite.

Si aucune comparaison n'est satisfaite dans ce balayage de l'argument, alors la recherche est infructueuse.

d) affectation des contextes

Si au contraire, la comparaison est satisfaite, alors on affecte à un identificateur de manoeuvre (disons a') les n caractères qui précèdent la sous-chaîne de l'argument égale au champ fixe.

Si le contexte a une longueur imposée, alors n est égal à cette longueur. Ici, n vaut 4 et a' vaut B A B C.

Si le contexte n'a pas de longueur imposée, alors n est égal au nombre total de caractères qui précèdent la sous-chaîne.

Par exemple l'instruction :

DANS y CHERCHER 'c' & 72 ;

appliquée à la chaîne y : "12 x 6 = 72"
affectera à c la valeur "12 x 6 = " .

Remarque

Tant que la recherche complète n'est pas terminée il n'y a pas d'affectation de contextes à des identificateurs du programme. L'affectation ne se produit qu'à la fin et seulement si la recherche est fructueuse. C'est la raison pour laquelle on se sert d'identificateurs de manoeuvre tels que a' .

e) vérification des contraintes de contexte

Une fois le contexte affecté à un identificateur de manoeuvre, il faut vérifier que les contraintes, s'il y en a, sont bien respectées. (contexte alphabétique ou numérique). Si ces contraintes ne sont pas respectées, alors on retourne exécuter la séquence de recherche de champs fixes décrite en (c). Si au contraire, les contraintes sont respectées ou

s'il n'y a pas de contrainte, on continue la recherche avec le second champ fixe du modèle, et le reste de l'argument. Pour cela on reprend le paragraphe (c).

f) fin de la recherche

Plusieurs causes peuvent rendre la recherche infructueuse. Par exemple :

- un champ fixe du modèle n'a pas été trouvé.
- un contexte n'a pas la longueur indiquée dans le modèle.
- un contexte n'est pas numérique.
- etc ... .

Dans ce cas, aucune affectation de contexte n'est effectuée et les identificateurs indiqués dans l'instruction n'ont pas leur valeur modifiée.

Au contraire, si toutes les conditions ont été satisfaites et lorsque le modèle de recherche a été entièrement retrouvé dans l'argument, alors on affecte aux identificateurs mentionnés dans les contextes les valeurs des identificateurs de manoeuvre.

Remarques

1- Lorsqu'une recherche est infructueuse et si plusieurs conditions sont imposées dans le modèle, on ne sait pas laquelle de ces conditions n'a pas été satisfaite. On peut imaginer d'associer à toute intruction de recherche une liste de valeurs booléennes correspondant à la satisfaction de toutes les conditions d'un modèle, mais cette solution alourdirait le mécanisme de la reconnaissance d'une chaîne et ferait de l'instruction CHERCHER, déjà suffisamment complexe, un outil peu commode à manier pour les utilisateurs du langage qui ne seraient pas informaticiens.

2- Il se peut que dans un modèle, il n'y ait pas de partie fixe.

Exemple :

CHERCHER '<a>' ;

On cherche ici une valeur numérique qui, si on la trouve sera affectée à l'identificateur a. Dans ce cas, l'exécution de l'instruction commence en (e) :

3- Il se peut également qu'il n'y ait pas de contextes dans un modèle.

Exemple :

CHERCHER a & b ;

Les valeurs concaténées de a et de b constituent le seul champ fixe du modèle. Dans ce cas on n'effectue pas les phases (e) et (f).

11 - Commentaires

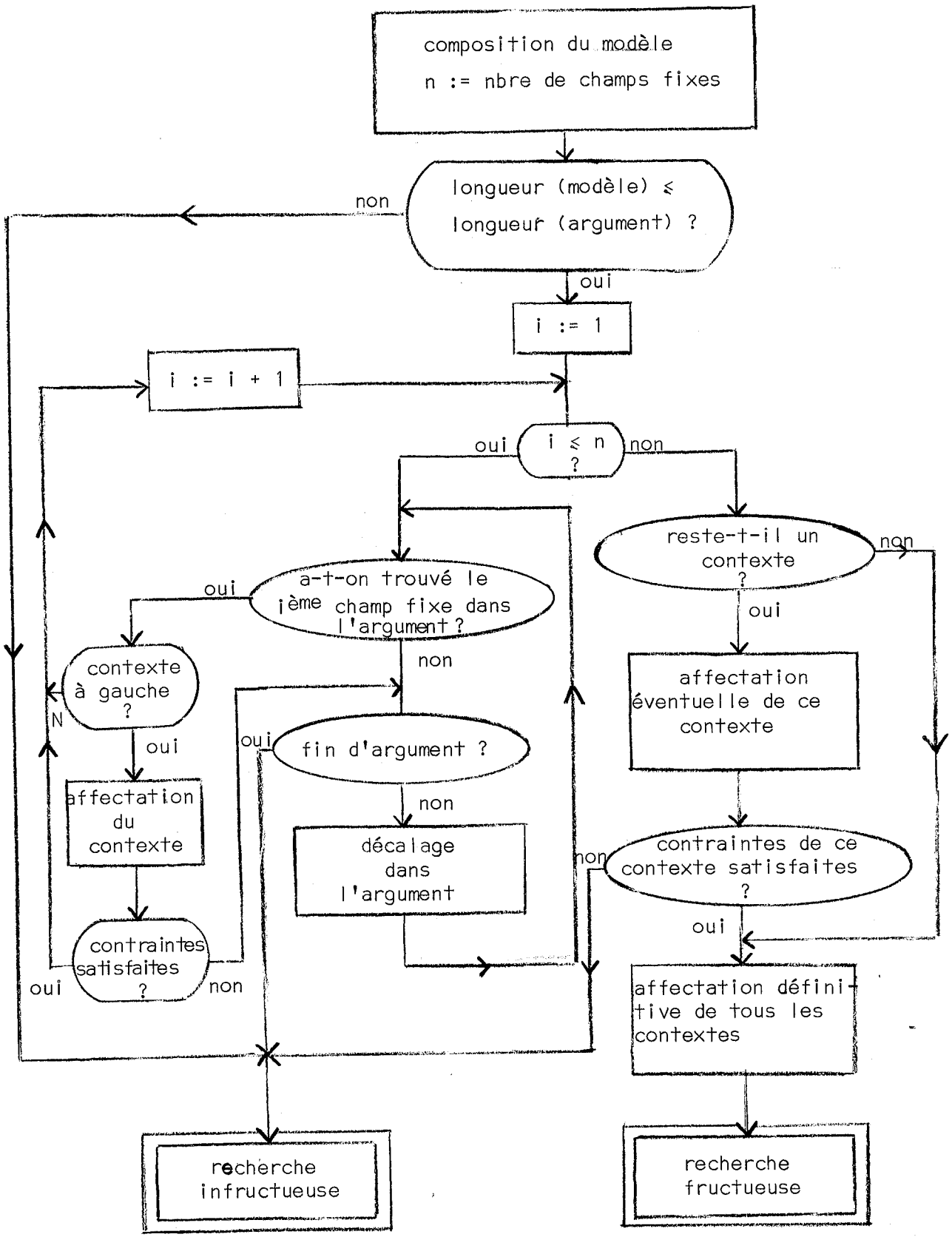
L'utilisateur peut écrire des commentaires n'importe où dans son programme source, en les encadrant de deux $.

12 - Carte syntaxique

On trouvera en annexe une grammaire mise sous forme normale de BACKUS décrivant la syntaxe de MAGISTER. Il faut, à cette occasion, se souvenir qu'un chapitre n'est pas autre chose que le programme source décrit par cette grammaire.

Un cours est normalement composé de plusieurs chapitres. C'est donc un ensemble de programmes qui n'ont en général pas de lien entre eux, si ce n'est sur le plan pédagogique.

FONCTIONNEMENT DE L'INSTRUCTION DE RECHERCHE



C - EXEMPLES DE COURS

On trouvera dans ce paragraphe trois exemples simples d'utilisation de MAGISTER. Ce sont des séquences brèves, qui n'ont pas la prétention de l'efficacité pédagogique et n'ont d'autre but que de montrer certains aspects du langage.

Les disciplines abordées sont :

- les mathématiques avec la notion de vecteur et de produit scalaire.
- l'informatique, avec l'étude d'une instruction d'une machine.
- la grammaire française avec un paragraphe sur le sujet et le verbe.

1 - Mathématiques

```
ECRIRE "Nous définirons ici un VECTEUR comme une suite ordonnée
de 2 ou plusieurs nombres.
Cela signifie que, pour que 2 vecteurs soient identiques, ils
doivent contenir les mêmes nombres, écrits dans le même ordre.
Ainsi
     (2 ; 4)
n'est pas identique à
     (4 ; 2) .
Remarquez que nous écrivons la suite des nombres entre parenthèses
et que les nombres sont séparés par des points-virgules.
Ainsi, (8 ; 2 ; 3) est un vecteur.
Pensez-vous que (8 ; -2 ; 3) soit un vecteur ?" ;

LIRE ;
CHERCHER "ne sais pas" | "peut-être" | "?" ;
```

— ALLER A e2 ;

neg : COMPARER AVEC "non" ;
— ALLER A e1 ;
CHERCHER "oui" ;
¬ ECRIRE "Soyez plus catégorique dans votre réponse." ;
¬ ALLER A neg ;
ECRIRE "Très bien.
(8 ; -2 ; 3) est certainement un vecteur. Le second terme de cette suite, -2 est un nombre négatif, mais c'est parfaitement légitime. En effet, tout nombre (ou symbole représentant un nombre) peut être un terme de la suite";

reprise : ECRIRE"
Ainsi, (X ; Y ; 4 ; -1 ; 0) est un vecteur." ;
ALLER A suite ;

e1 : ECRIRE "C'est le -2 qui vous ennuie ? n'importe quel nombre peut être utilisé comme terme d'une suite, même les nombres négatifs. Un symbole peut également être utilisé pour représenter un nombre." ;
ALLER A reprise ;

e2 : ECRIRE "C'est pourtant bien simple ! Donner une autre réponse." ;
LIRE ;
ALLER A neg ;

suite : ECRIRE "Les nombres qui définissent un vecteur sont appelés <u>composantes</u>. Le vecteur (2 ; 3) a 2 composantes ; (3 ; 9 ; -4 ; 6 ; -1) a 5 composantes et -4 est la troisième composante.

```
        combien le vecteur (0 ; 1 ; 2 ; 3) a-t-il de composantes ?" ;
        LIRE ;
        CHERCHER 4 | "quatre" ;
        ¬ ECRIRE "Il y a 4 nombres dans cette suite; il y a donc 4 compo-
        santes dans ce vecteur." ;

scalaire : ECRIRE "Bien. Nous allons définir maintenant une opération sur
        les vecteurs ayant le même nombre de composantes. On l'appelle
        produit scalaire parce que le résultat en est un scalaire, c'est-
        à-dire un nombre.
        Si A est le vecteur (a1 ; a2) et B le vecteur (b1 ; b2), le pro-
        duit scalaire de A et B (qu'on note A.B) est :
             A.B = a1 x b1 + a2 x b2
        A.B se lit ''A scalaire B''.
        quel est le produit scalaire de A = (2 ; 5) et B = (7 ; 1) ? " ;

iter : LIRE ;
        CHERCHER '<n>' ; ¬ ECRIRE "Donnez une réponse numérique." ;
        ¬ ALLER A iter ;
        Si n ¬= 19 ALORS ACTIVER scal 1 ;
        ECRIRE "Exact.
        (2 ; 5) . (7 ; 1) = (2 x 7) + (5 x 1) = 14 + 5 = 19
        Les composantes négatives ne posent pas de problème particulier.
        Par exemple :
        (-4 ; 3) . (6 ; 2) = (-4 x 6) + (3 x 2) = -24 + 6 = -18.
        Le produit scalaire peut se faire facilement avec des vecteurs qui
        ont plus de deux composantes. Par exemple :
        (4 ; 2 ; 3) . (3 ; 5 ; -1) = 4 x 3 + 2 x 5 + 3 x (-1) = 12 + 10 -3 = 19
        ALLER A scal2 ;

scal1 : ECRIRE "Votre réponse est inexacte.
        Pour obtenir le produit scalaire des deux vecteurs (4 ; 8) et (3 ; 2),
```

```
        il faut faire le produit des premières composantes de chaque vecteur.
        4 x 3 = 12,
        et le produit des secondes composantes :
        8 x 2 = 16,
        puis faire la somme
        12 + 16 = 28.
        le nombre 28 est le produit scalaire de (4 ; 8) et (3 ; 2).
        Quel est donc le produit scalaire de (2 ; 5) et (7 ; 1) ?" ;
        ALLER A iter.


scal2 : ECRIRE "Nous allons faire maintenant une série d'exercices sur le
          produit scalaire.
          Donnez simplement la valeur numérique du résultat." ;
          i := 1 ;


scal3 : a := -3 | -2 | -1 | 1 | 2 | 3 ;
        b :=  3 | 4 | 5 | 6 ;
        c := -2 | 3 | -5 | 6 ;
        m :=  4 | -2 | 1 | 0 ;
        n :=  6 | -3 | 2 | 1 ;
        p :=  0 | -6 | 4 | 2 | -3 ;
          ECRIRE "("& a &";"& b &";"& c &").("& m &";"&n &";"& p &")=";
          LIRE produit ; résultat := a * m + b * n + c * p ;
          Si produit <> résultat ALORS ECRIRE "non. c'est"
          & résultat & "." ;
          i := i + 1 ;
          Si i <= 10 ALORS ALLER A scal3 ;
```

2 - Informatique

```
ECRIRE "Nous allons étudier aujourd'hui l'instruction 'Branch on Count'.
Elle possède 2 formats : RR et RX qui correspondent respectivement
aux écritures :
     BCTR  R1,R2
et   BCT   R1,D2(X2,B2)  .

On utilise cette instruction pour contrôler l'exécution d'une boucle
dans un programme. Son déroulement est le suivant :
   1 - Le contenu du premier opérande (le registre R1) est diminué de 1.
   2 - On teste la valeur du résultat.
   3 - Si ce résultat est zéro, aucun transfert n'a lieu et on exécu-
       tera la prochaine instruction en séquence.
   4 - Si le résultat n'est pas nul, on se transfère à l'adresse
       spécifiée par le second opérande, sauf si ce second opérande
       est le registre R0.

Quand vous serez prêt, pressez la touche Retour-Chariot." ;
LIRE a ;
SI !a -= 0 , ECRIRE " Un Retour-Chariot suffisait ! " ;
code := "BCT" ;
ACTIVER rxrr ;
ECRIRE "Quelle quantité soustrait-on au contenu du premier opérande ?";
LIRE ;
CHERCHER 1 | "un" | "Un" | "UN" ;
¬ ECRIRE "Vous avez la mémoire courte ; c'est 1 ." ;
— ECRIRE "Oui" ;
ECRIRE "Supposons que le registre 7 contienne la valeur +1. Lors de
l'exécution de
     BCT   7,TOTO
 y aura-t-il transfert ?" ;
```

réponse : LIRE EN 10 SECONDES ;

```
        ¬ ACTIVER pressons ;
          CHERCHER "oui" ;


faux : — ECRIRE "Mais non ! On fait d'abord la soustraction, puis on examine
          le résultat. Ici, on obtient zéro et il n'y a donc pas de transfert.";
          ALLERA suite ;
          CHERCHER "non" ;


juste : — ECRIRE "en effet" ;
          — ALLERA suite ;
          CHERCHER "n' '| "ne" , "pas" ;
          — ALLERA juste ;
          CHERCHER ( "en", "aura" ) | ( "aura" , "transfert" ) ;
          — ALLERA faux ;
          ECRIRE "Votre réponse n'est pas claire ; donnez-en une autre ! " ;
          ALLERA réponse ;


suite : ECRIRE " Dans l'exemple suivant, le registre 7 contient au départ
                la valeur +5 .
        BOUCLE ___   _______
               ___   _______
               ___   _______
               ___   _______
               BCT   7,BOUCLE
        SUITE  ___   _______

          Combien de fois cette boucle sera-t-elle exécutée avant que l'on
          passe à SUITE ? " ;
          LIRE ;
          CHERCHER 5 | "cinq" ;
```

```
        — ECRIRE "oui" ;
        — ALLERA fin ;
        ECRIRE "Non ; la boucle sera exécutée 5 fois" ;


fin : ECRIRE "En conclusion, on doit retenir que la soustraction se fait
        avant le test et que le transfert a lieu si le résultat est non
        nul . " ;


pressons : ECRIRE "Vous êtes lent" | "Vous traînez" | "Vous devriez déjà avoir
        répondu" | "Répondez plus rapidement" ;
        LIRE .

rxrr : ECRIRE code & " est le code de l'instruction pour le format RX.
        Quel est le code pour le format RR ?" ;
        LIRE ;
        CHERCHER code & "R" ;
        — ECRIRE "oui" ;
        — ALLERA retour ;
        ECRIRE "Non; il suffit d'ajouter un R au code en RX pour obtenir
        le code en RR. Ici c'est donc : "& code & "R" ;


retour :  .
```

3 - Grammaire française

```
        ECRIRE "Dans la phrase suivante, indiquez le verbe :
                '' les enfants partent '' " ;


e1 : LIRE ;
```

```
        COMPARER AVEC "partent" ;
        — ALLER A e2 ;
        COMPARER AVEC "enfants" | "les enfants" ;
        — ECRIRE "Non. vous avez indiqué le sujet. Recommencez" ;
        — ALLER A e1 ;

horstexte : ECRIRE "Uilisez dans votre réponse les mots qui figurent dans
        cette phrase." .
        ECRIRE "Quel en est le verbe ?" ;
        ALLER A e1 ;


e2 : ECRIRE "oui. Et dans la phrase
             ''Les enfants sont partis'' ,
        Quel est le verbe ?" ;


e3 : LIRE ;
        COMPARER AVEC "sont" ;
        — ECRIRE "Votre réponse est partiellement exacte. En fait le verbe
        est ici composé des deux mots : sont partis." ;
        — ALLER A e4 ;
        COMPARER AVEC "sont partis" ;
        ¬ ACTIVER horstexte ;
        ¬ ALLER A e3 ;
        ECRIRE "Très bien!" ;


e4 : ECRIRE "Voyons maintenant autre chose.
        Dans les phrases interrogatives suivantes :
        ''Quand partez-vous ?''
        ''Viens-tu ?''
        ''Arriveront-ils à temps ?''
        les sujets sont-ils placés avant ou après le verbe ?" ;


e5 : LIRE ;
```

```
        COMPARER AVEC "après" ;
        — ALLER A e6 ;
        ECRIRE "Faites un peu attention à ce que vous faites.
        Répondez à nouveau" ;
        ALLER A e5 ;


e6 : ECRIRE "En effet. Ce que vous avez fait s'appelle une inversion du
        sujet.
        Par exemple, on dit habituellement :
        ''Les vedettes se balancent dans le port''.
        On peut dire aussi, en inversant le sujet :
        ''Dans le port se balancent les vedettes''. " ;
        a := "une voile blanche" | "un bateau" ;
        b := "paraît" | "se montre" ;
        c := "à l'horizon" | "près du port" ;
        ECRIRE "Faites la même opération avec la phrase suivante :
        "& a & b & c ;


e7 : LIRE ;
        CHERCHER c & b & a ;
        — ALLER A e8 ;
        CHERCHER c & a & b ;
        — ECRIRE "Vous avez inversé le complément de lieu "& c &"
        mais pas le sujet. Donnez une autre forme à cette phrase" ;
        — ALLER A e7 ; ECRIRE" non. Essayez encore une fois" ; ALLER A e7 ;


e8 : ECRIRE "Excellent !" ;
```

CHAPITRE III

C H A P I T R E III

COMPILATION DU LANGAGE MAGISTER

A - ASPECT CONVERSATIONNEL

1 - Généralités

Dans une compilation classique, le programme à compiler constitue un tout non morcelé pour les diverses phases qui se succèdent. La phase d'édition fournit à partir de l'ensemble du programme source une chaîne codée, qui sert ensuite d'entrée à la phase de transformation et de génération. Cette phase génère un programme objet qui est soit du code machine, soit du code interprété. Généralement, lorsqu'une phase détecte une erreur qui est de son ressort, elle poursuit son travail jusqu'au bout mais ne donne pas ensuite le contrôle à la phase suivante. Pour que la phase suivante soit appelée, il faut que la "matière" sur laquelle travaille la phase en cours soit sans faille. Un certain nombre de vérifications sont effectuées à la fin de chaque phase pour s'assurer que ce que l'on transmet est homogène.

Par exemple, les déclarations d'identificateurs, lorsqu'il y en a font l'objet d'un examen à la fin de la phase d'édition pour détecter les manques éventuels.

S'il y a une erreur, il est inutile d'appeler la phase suivante qui produirait alors quelque chose d'incorrect.

Le programmeur est simplement invité à corriger son erreur et à remettre tout le programme source en entrée pour une compilation de l'ensemble de ce programme.

Dans une compilation conversationnelle, en revanche, on s'efforce d'établir une interaction entre le programmeur et le programme en cours de compilation. Les erreurs commises par le programmeur lui sont aussitôt signalées et l'information où se situe l'erreur est refusée. Le programmeur corrige immédiatement l'information erronée et le traitement se poursuit de cette façon.

Il est souhaitable d'étendre cette interaction à une modification volontaire du programme par le programmeur. Si celui-ci désire modifier, ajouter, remplacer ou supprimer une ou plusieur instructions dans son programme, il doit pouvoir le faire sans entraîner une recompilation de l'ensemble du programme, mais en n'effectuant des changements que pour les instructions intéressées. Ceci peut être obtenu par une technique de compilation dite "incrémentielle" dans laquelle on se sert d'indicateurs explicites pour définir l'enchaînement des instructions. L'"incrément" est en général constitué d'une instruction ou d'une portion d'instruction. Dans ce cas on s'efforce de faire le plus grand nombre possible de transformations à partir des bribes de programme qui sont transmises par le terminal. Ainsi lorsque le programmeur écrit une instruction sur sa console, on va appliquer à cette instruction le traitement des diverses phases qui conduira soit à son rejet, en cas d'erreur, soit à son acceptation et à une génération éventuelle d'une partie du programme objet.

En cas de modification volontaire d'une instruction, la liste d'indicateurs explicites d'enchaînement des incréments objets est remise à jour pour refléter le changement intervenu dans le programme objet. (on trouvera en {5} un exposé très complet des différentes méthodes de compilation).

2 - Application au langage MAGISTER

Il s'agit donc ici de compiler pas à pas les instructions introduites par le professeur et éventuellement d'effectuer les modifications qu'il réclame.

Deux remarques s'imposent à ce niveau :

a) les modifications porteront souvent sur des points de détail et il est donc tout à fait indiqué d'avoir une compilation incrémentielle.

b) lorsqu'on écrit un compilateur conversationnel d'ALGOL ou de PL/I, surtout destiné à l'apprentissage d'un langage de programmation, on ne s'intéresse pas tellement aux performances du programme résultant. Ceci revient à reporter à la phase d'exécution ou d'interprétation un certain nombre de vérifications syntaxiques ou sémantiques, ce qui ralentit les performances.

Ici, au contraire, il s'agit d'accroître le plus possible l'efficacité de la phase d'interprétation, qui gère dans notre cas la séance d'apprentissage avec les élèves. Il faut donc que la compilation proprement dite (analyse syntaxique et synthèse sémantique) fasse toutes les vérifications souhaitables, même si c'est au détriment du délai de réponse du système vers le professeur. La priorité est donnée à l'interprète plutôt qu'au compilateur, à l'élève plutôt qu'u maître. Néanmoins, celui-ci dispose d'un langage de commandes assez souple pour introduire son cours en machine.

Une série de commandes et de requêtes lui permettent d'effectuer les opérations de création ou de modification désirées.

B - LE LANGAGE DE COMMANDE

Lorsque l'utilisateur a chargé dans sa mémoire virtuelle le système CMS, il appelle le compilateur par la commande :

cours

suivie du nom du cours et du numéro de chapitre.

Exemple :

cours algèbre 11

Le blanc est un séparateur pour les divers éléments de la commande, comme pour toutes les commandes et requêtes examinées ici. L'effet de cette commande cours est de charger dans la mémoire virtuelle les différents modules du compilateur et de donner le contrôle au module de départ.

Deux cas peuvent se présenter :

1) le chapitre indiqué n'existe pas encore.

Le professeur désire donc introduire un nouveau chapitre en machine. Le terminal est alors placé dans un environnement de génération de programme.

Le professeur est invité à écrire les instructions du chapitre les unes après les autres. Les lignes sont numérotées sur la gauche par le compilateur. En cas d'erreur syntaxique ou sémantique, celle-ci est immédiatement signalée par un commentaire approprié et le même numéro de ligne est alors imprimé au terminal. L'ancienne ligne erronée n'est évidemment pas conservée.

Remarques

a) Une ligne peut contenir une instruction, mais pas plus.
Si le professeur écrit plusieurs instructions par ligne, seule la première instruction est retenue pour cette ligne. Le compilateur réécrit alors les autres instructions sur les lignes suivantes, en faisant progresser le numéro de ligne.

Exemple

```
0 0 0 7 (instruction 1) ; (instruction 2) ; (instruction 3)
0 0 0 8 (instruction 2) ;
0 0 0 9 (instruction 3) ;
```

Cette façon de procéder a été adoptée pour permettre la localisation par un numéro de ligne, d'une instruction en cas de modification.

b) Une instruction peut tenir sur plusieurs lignes. C'est par exemple le cas pour l'instruction ECRIRE lorsqu'il s'agit de présenter un long texte à l'élève. Une erreur dans une instruction peut se produire alors que une ou plusieurs lignes correctes constituant le début de l'instruction ont déjà été acceptées.

Seule la ligne erronée est refusée et les résultats partiels obtenus avec les lignes correctes précédentes restent valides. Si l'instruction est correcte, il lui correspond un enregistrement dans le programme objet. La génération du code objet s'effectue au fur et à mesure que les instructions sont écrites au terminal.

Pour sortir de l'environnement de génération, il suffit d'introduire une ligne vide au terminal, c'est-à-dire de presser deux fois de suite la touche Retour-Chariot. Il est nécessaire de terminer l'écriture de l'instruction en cours avant de quitter cet environnement.

Le terminal est alors placé dans un autre environnement, celui de modification. Le professeur peut, à ce moment-là, opérer des changements dans son chapitre. L'environnement de modification est décrit plus loin.

2) le chapitre indiqué existe déjà.

Le professeur désire donc travailler sur un chapitre déjà introduit en machine. Le terminal est alors placé dans un environnement d'attente de commandes. Quatre commandes peuvent alors être indiquées par le professeur.

a) la commande :

generer

met le terminal dans l'environnement de génération décrit plus haut. Les instructions écrites à ce moment-là seront placées à la fin du programme existant. C'est donc là le moyen de compléter un chapitre.

b) la commande

detruire

a pour effet d'effacer de la machine toute trace du chapitre indiqué dans la commande cours. La présence d'un chapitre est liée à l'existence de trois fichiers : le fichier source contenant les instructions du programme telles que l'utilisateur les a écrites après les corrections éventuelles, le fichier objet généré par le compilateur, et une table d'identificateurs et d'étiquettes qui permet de vérifier l'homogénéité du code objet. La commande detruire efface ces 3 fichiers du disque de l'utilisateur puis rend le contrôle à CMS.

Remarque

Dans le système CMS, il existe une commande analogue, c'est :

ERASE

qui efface un fichier du disque.

Nous ne l'avons pas utilisée pour la raison suivante :
les noms et les types des 3 fichiers décrits plus haut sont codés et ne peuvent pas s'imprimer au terminal, et ceci pour qu'on ne puisse pas modifier ces fichiers dans l'environnement de CMS. (par EDIT par exemple).
La seule possibilité de modifier ces 3 fichiers est donc de passer par le compilateur. Puisque le nom et le type de ces fichiers sont codés ils sont donc inaccessibles à partir de CMS et il fallait une commande spécifique pour les détruire.

c) la commande

texte

a pour objet de faire sortir sur l'imprimante les fichiers source et identificateurs du chapitre. C'est l'analogue de :

OFFLINE PRINT

de CMS. Là encore, à cause du codage des noms et types, il fallait une commande spécifique pour assurer cette impression.

d) la commande

modifier

place le terminal dans un environnement de modification. L'utilisateur peut alors écrire l'une des 12 requêtes indiquant le type de mise à jour à effectuer sur les fichiers. Cette mise à jour s'effectue dans la mémoire virtuelle de la machine où l'on trouve donc les 3 fichiers source, objet et identificateurs. Vu du côté de l'utilisateur cette mise à jour se passe de la façon suivante :

Un pointeur indique dans le fichier source la position de la ligne courante. C'est au niveau de cette ligne qu'ont lieu les modifications du fichier source qui sont ensuite répercutées sur le fichier objet et la table d'étiquettes et d'identificateurs.

Voici la liste des requêtes avec leurs fonctions respectives

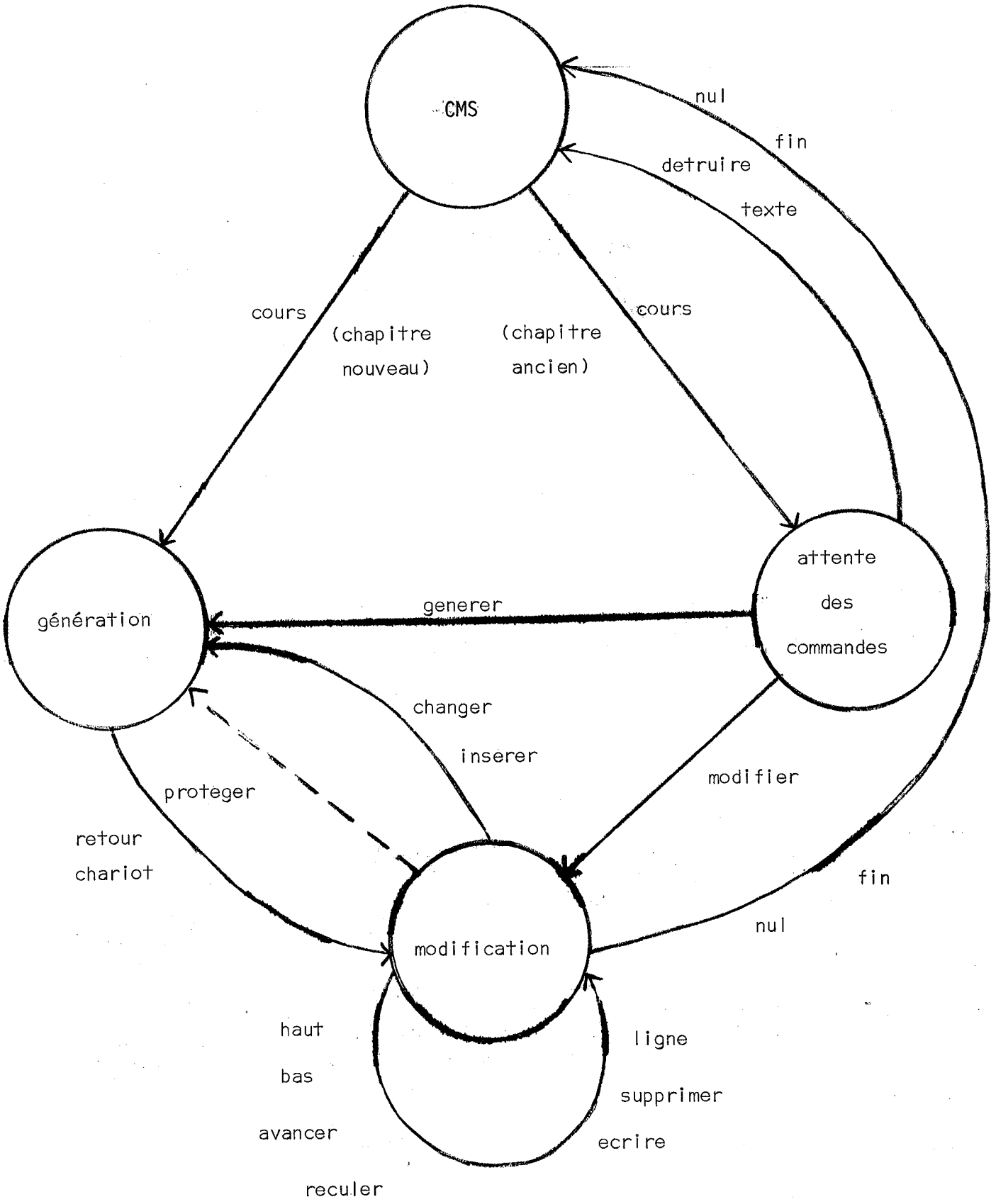
- haut
  place le pointeur au début du fichier avant la première ligne.
- bas
  place le pointeur à la fin du fichier en face de la dernière ligne qui est alors imprimée au terminal.
- avancer n où n est un entier (1 par défaut) déplace le pointeur de n lignes vers le bas du fichier puis imprime la ligne courante.
- reculer n
  fait remonter le pointeur de n lignes dans le fichier puis imprime la ligne courante.
- ligne n
  place le pointeur en face de la $n^{ième}$ ligne du fichier qui est alors imprimée.
- changer <ligne>
  remplace la ligne courante par la ligne figurant en paramètre. L'instruction concernée par cette ligne est alors recompilée.
- inserer
  place le terminal dans l'environnement de génération décrit plus haut. Les lignes écrites par l'utilisateur sont insérées après la ligne courante repérée par le pointeur. On revient à l'environnement de modification en pressant deux fois de suite la touche Retour-Chariot.
- supprimer n (n = 1 par défaut)
  détruit du fichier les n lignes à partir de la ligne courante. Cette requête peut entraîner des modifications dans la table des identificateurs et étiquettes si, par exemple, une déclaration d'étiquette figurait dans une ligne supprimée.
- protéger
  sauvegarde sur disque les fichiers source, objet et table, dans leur état actuel puis place le terminal dans l'environnement de génération.

- ecrire n (n = 1 par défaut)
  imprime au terminal les n lignes à partir de la ligne courante. La nouvelle ligne courante est la dernière imprimée.
- nul
  rend le contrôle immédiatement à CMS sans recopier sur disque les versions actuelles des 3 fichiers source, objet et table. La mise à jour est donc annulée.
- fin
  écrit sur disque les versions actuelles des 3 fichiers source, objet et table puis rend le contrôle à CMS.

e) Récapitulation sur les commandes et requêtes

Ces commandes et requêtes peuvent être écrites en clair ou en abrégé. La première lettre suffit à les différencier.

Voici un schéma décrivant les divers environnements et la manière de passer de l'un à l'autre.


CMS
nul
fin
detruire
texte
cours
(chapitre
nouveau)
(chapitre
ancien)
cours
génération
generer
attente
des
commandes
changer
inserer
modifier
proteger
retour
chariot
fin
nul
modification
haut
bas
avancer
reculer
ligne
supprimer
ecrire

C - DESCRIPTION DU COMPILATEUR

1 - Références

La méthode de compilation adoptée est celle décrite par M. GRIFFITHS dans {17}.

Brièvement, elle consiste à faire assurer la partie d'analyse syntaxique par un programme fabriqué automatiquement à partir d'une grammaire de type LR1 décrivant le langage. Cet analyseur syntaxique est composé d'une série de procédures définies à l'aide de macro-instructions. Nous avons utilisé quelques macro-instructions décrites par M. PELTIER dans {24}. Ce sont celles qui permettent la récursivité dans les appels de procédures : MODULE, ROUTINE, CHECK, CALL, ENTER, EXIT, RETURN.

Un module est un ensemble de routines. Les routines peuvent s'appeler les unes les autres par CALL (avec retour) ou par ENTER (transfert simple). La macro CHECK permet de vérifier la valeur d'un symbole de base attendu et appelle un sous-programme d'erreur dans le cas où on ne trouve pas le symbole qu'on attendait. EXIT et RETURN sont des retours vers la routine appelante.

La conjonction de l'analyseur syntaxique et des macro-instructions constitue un outil très agréable à utiliser et permet de concentrer son effort sur la génération du code objet. Cette génération se fait au moyen de fonctions sémantiques qui figurent dans la grammaire elle-même. L'analyseur syntaxique qui est délivré comporte alors des appels à des routines correspondant aux fonctions à réaliser. Ces routines doivent être écrites à la main et elles ont pour la plupart le rôle de générer du code.

2 - Analyse lexicographique

Elle est effectuée par la routine PREPROC qui joue donc le rôle d'un éditeur ou préprocesseur. Sa tâche essentielle est de fournir un symbole de base à la routine qui l'a appelé.

C'est PREPROC qui gère les commentaires, en les ignorant purement et simplement, les mots-clés reconnus par une technique qui s'apparente au "hash-coding", les chaînes littérales et les nombres, les identificateurs rangés dans une table comportant aussi leur longueur et le nombre de leurs occurrences dans le programme. PREPROC appelle la routine LIRE qui lui fournit 1 caractère et qui débloque le clavier quand le dernier caractère de la ligne a été transmis.

3 - Analyse syntaxique

Elle est réalisée par le module produit de la grammaire. Ce module vérifie la validité des instructions et appelle PREPROC à chaque symbole de base. En cas d'erreur, la ligne en cours est annulée, c'est-à-dire qu'on se replace dans l'état correspondant à la fin de la dernière ligne correcte. Ceci nécessite la sauvegarde de cet état (niveau de la table d'identificateurs et d'étiquettes, chaîne de reprise des appels des routines) à chaque fois qu'une ligne correcte est terminée. Si une erreur se produit dans la ligne suivante, il suffit de restaurer tous ces éléments et de lancer une nouvelle lecture au terminal.

4 - Synthèse sémantique

Elle est réalisée par une série de routines qui génèrent des éléments de code objet en fonction de ce que l'on trouve en entrée. Ainsi les expressions arithmétiques sont traitées par une routine qui effectue leur mise sous forme postfixée, en utilisant la classique pile d'opérateurs. L'interprète fonctionne évidemment avec une pile pour l'évaluation des expressions postfixées. Pour l'instruction CHERCHER, on génère les éléments nécessaires à l'interprète pour construire le modèle de la recherche. La génération des "listes aléatoires" d'expression comporte une sorte d'aiguillage vers les divers sous-programmes d'évaluation.

Exemple :

a + b * c | "résultat" | 172 ,sera représenté par

R ≡ retour de sous-programme

C H 8 ≡ chaîne de 8 caractères

N B 3 ≡ nombre de 3 chiffres.

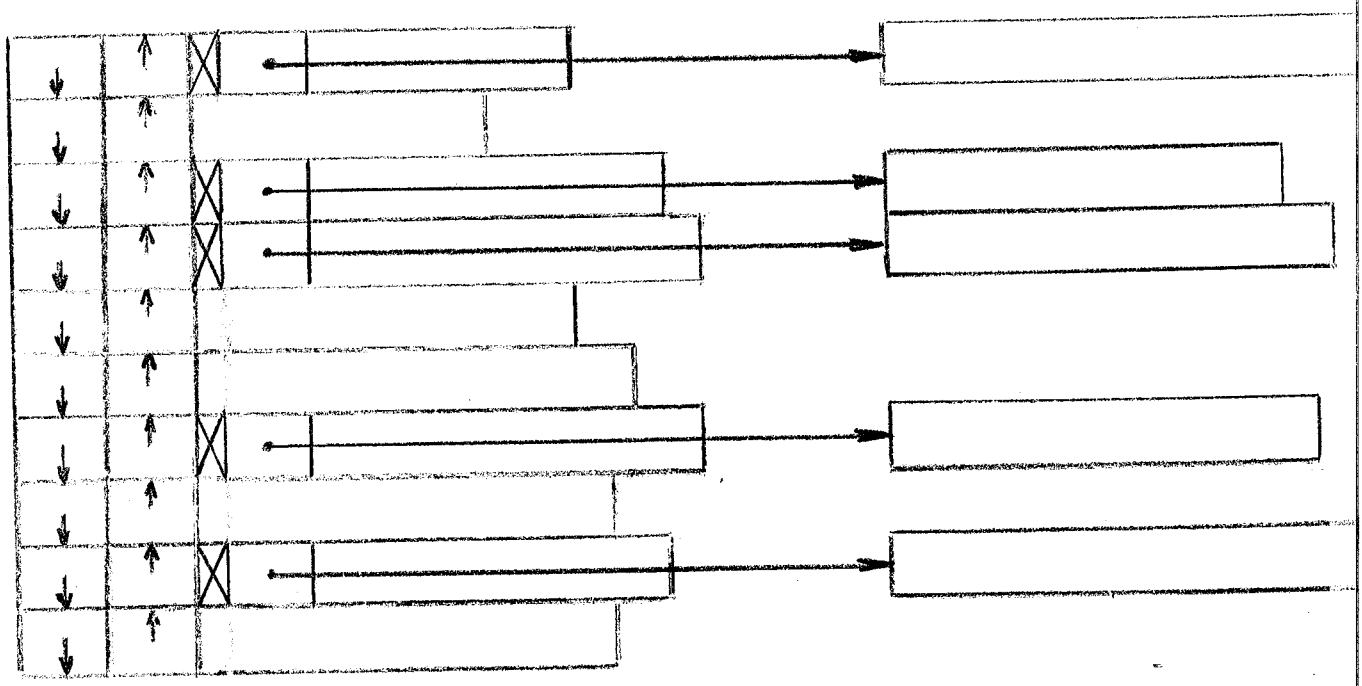
5 - Gestion de la mémoire et des fichiers

A la compilation, on conserve dans la mémoire virtuelle le fichier source, le fichier objet et la table d'identificateurs et d'étiquettes. Lorsqu'ils sont sur disque, les fichiers sont écrits ligne par ligne pour le source et instruction par instruction pour l'objet. En mémoire, on associe au fichier source une liste symétrique de pointeurs qui localisent la ligne précédente et la ligne suivante. Cette liste est symétrique pour pouvoir avancer ou reculer dans le fichier source à la recherche de la ligne indiquée en paramètre de la requête ligne. Par exemple, si le pointeur de ligne localise la ligne 35, et si l'utilisateur désire aller à la ligne 19, on utilise cette liste symétrique pour remonter dans le fichier source jusqu'à la ligne 19. Le même processus est utilisé avec les requêtes avancer et reculer.

Chaque ligne qui est un début d'instruction est marquée dans le fichier source. Cette marque est faite par le module d'analyse syntaxique au début de chaque instruction. A chaque ligne marquée correspond un enregistrement du fichier objet. La correspondance est faite en mémoire après la marque.

En cas de modification d'une ligne source, on remonte, par le système de liste, jusqu'à la ligne marquée qui commence l'instruction en cours. On recompile toute l'instruction en tenant compte de la modification apportée au fichier source. La liste du fichier source est modifiée en mémoire.

Un incrément nouveau est créé, toujours pointé par la marque dans la ligne. Ainsi, on n'a pas besoin de liste pour le programme objet. A la fin de la compilation, on se sert de la liste source pour réécrire sur disque les fichiers source et objet.



liste source lignes enregistrements objets

FICHIER source FICHIER OBJET

X marque de début d'instruction

Remarques

1) La table d'identificateurs et d'étiquettes indique le nombre d'occurrences de chaque élément. Ceci est fait pour la raison suivante :

Si l'utilisateur supprime toute apparition d'une étiquette y compris sa déclaration, alors il peut maintenant se servir du même symbole pour représenter un identificateur. Il faut donc pouvoir compter les occurrences des étiquettes, et des identificateurs. C'est également par cette table qu'on peut savoir en fin de compilation si toutes les étiquettes utilisées dans le programme ont bien été déclarées.

2) En cas de modifications, il n'y a pas de "ramasse-miettes" (garbage collection). D'une part, ce serait très coûteux, d'autre part, les modifications restent modestes dans le contexte de l'enseignement assisté et concernent des points de détail.

C O N C L U S I O N

CONCLUSION

Les informaticiens ont quelquefois tendance à considérer leur discipline comme une fin en soi, et non comme un moyen.

Ils ressemblent en cela à ces spécialistes qui ne veulent pas quitter leurs problèmes spécifiques pour s'intéresser aux applications possibles de leur technique.

C'est dans les domaines les moins bien maîtrisés que cet isolement est le plus néfaste.

L'enseignement assisté par ordinateur est un exemple parfait d'application où l'informaticien doit se demander avant toute chose quelle utilisation sera faite de l'outil qu'il réalise, qui s'en servira effectivement, et pour atteindre quel résultat.

Le plus sage est de comparer l'enseignement automatisé avec les autres intermédiaires qui lient le professeur et l'étudiant. Pour établir cette comparaison il est nécessaire d'avoir un système d'enseignement assisté, le plus général possible et de faire de nombreuses expérimentations.

C'est là l'objectif essentiel du système dont cette thèse décrit la première partie.

Le langage MAGISTER est volontairement simple, d'une part pour rendre son emploi commode aux non-informaticiens, d'autre part pour éviter une spécialisation dans une discipline donnée avec une stratégie pédagogique donnée.

En contre-partie, il impose au professeur de tout expliciter dans son cours, ce qui peut passer pour un inconvénient. En fait, il n'est pas mauvais de se pencher avec minutie sur les détails d'un cours, pour tenter d'en dégager des principes généraux pour sa rédaction.

La méthode de compilation présentée ici, repose sur l'utilisation conjointe de deux outils indépendants, l'un CP/CMS, très lié au matériel, l'autre, l'analyseur syntaxique, universel. Le passage à un autre type de matériel constituera un travail assez mince, limité à quelques modifications dans les opérations d'entrée et de sortie. Par exemple, pour un transfert sur un 360 standard, équipé d'un système partageant une ressource réentrante à plusieurs utilisateurs, le coût est de un ou deux mois. Pour utiliser un autre matériel, il faut d'une part réécrire le compilateur et d'autre part réécrire le système de diffusion. Cela représente un coût d'environ un an. Ces temps s'entendent pour une personne.

Le système proposé, par son caractère général permettra la comparaison de stratégies pédagogiques différentes appliquées à une même discipline ou à des disciplines différentes, et l'ordinateur sera bel et bien un nouvel outil pédagogique.

ANNEXE : SYNTAXE DE MAGISTER .

{chapitre} ::= {instruction} {ponctuation} ⊥
{instruction} {ponctuation} {chapitre}

{instruction} ::= {etiquette} : {instruction} ⊥ {instruction prefixee} ⊥
{instruction non prefixee}

{ponctuation} ::= ; ⊥ .

{instruction prefixee} ::= {prefixe} {instruction non prefixee}

{prefixe} ::= _ ⊥ ¬

{instruction non prefixee} ::= {instruction conditionnelle} ⊥
{instruction inconditionnelle}

{instruction inconditionnelle} ::= {instruction de lecture} ⊥
{instruction d'ecriture} ⊥
{instruction de transfert} ⊥
{instruction vide} ⊥
{instruction de recherche} ⊥
{instruction d'affectation} ⊥
{instruction de comparaison}

{instruction conditionnelle} ::= {condition} {instruction inconditionnelle}

{condition} ::= SI {expression booleenne} ALORS ⊥
SI {expression booleenne} ALORS {instruction inconditionnelle} SINON

{expression booleenne} ::= {facteur booleen} ⊥
{facteur booleen} OU {expression booleenne}

{facteur booleen} ::= {secondaire booleen} ⊥
{secondaire booleen} ET {facteur booleen}

{secondaire booleen} ::= {primaire booleen} ⊥ NON {primaire booleen}

{primaire booleen} ::= {relation} ⊥ ({expression booleenne})

{relation} ::= {expression arithmetique} {operateur de relation}
{expression arithmetique}

{operateur de relation} ::= = ⊥ < ⊥ > ⊥ ¬= ⊥ <= ⊥ >=

{instruction de lecture} ::= LIRE ⊥ LIRE {attribut de lecture}

{attribut de lecture} ::= {chaine} ⊥ {chaine} {temps limite} ⊥
{temps limite}

{temps limite} ::= EN {nombre sans signe} {unite de temps}

{nombre sans signe} ::= {chiffre} ⊥ {nombre sans signe} {chiffre}

{unite de temps} ::= MINUTE ⊥ MINUTES ⊥ SECONDE ⊥ SECONDES

{instruction de transfert} ::= {verbe de transfert} {liste de transfert}

{verbe de transfert} ::= ALLERA ⊥ ACTIVER

{liste de transfert} ::= {transfert} ⊥ {liste de transfert} | {transfert}

{transfert} ::= {etiquette} ⊥ {etiquette} {specification de chapitre}

{etiquette} ::= {identificateur}

{specification de chapitre} ::= DU CHAPITRE {nombre sans signe}

{instruction de comparaison} ::= COMPARER {attribut de comparaison}

{attribut de comparaison} ::= AVEC {liste d'expressions} ⊥
{expression} AVEC {liste d'expressions}

{instruction d'ecriture} ::= ECRIRE {liste d'expressions}

{instruction d'affectation} ::= {partie gauche} {liste d'expressions}

{partie gauche} ::= {chaine} :=

{liste d'expressions} ::= {expression} ⊥ {expression} | {liste d'expressions}

{expression} ::= {expression simple} ⊥
{expression} {operateur de concatenation} {expression simple}

{operateur de concatenation} ::= &

{expression simple} ::= {expression arithmetique} ⊥ {expression alphanumerique}

{expression arithmetique} ::= {terme} ⊥ {operateur additif} ⊥
{expression arithmetique} {operateur additif} {terme}

{operateur additif} ::= + ⊥ -

{terme} ::= {facteur} ⊥ {terme} {operateur multiplicatif} {facteur}

{operateur multiplicatif} ::= * ⊥ /

{facteur} ::= {primaire arithmetique} ⊥ {facteur} ** {primaire arithmetique}

{primaire arithmetique} ::= {longueur} ⊥ {variable} ⊥
{nombre sans signe} ⊥ ({expression})

{instruction de recherche} ::= CHERCHER {modele de recherche} ⊥
DANS {chaine} CHERCHER {modele de recherche}

{modele de recherche} ::= {element de recherche} ⊥
{modele de recherche} {operateur de concatenation} {element de recherche}

{element de recherche} ::= {contexte} ⊥ {liste alphanumerique}

{contexte} ::= , ⊥ ' {contexte alphanumerique} '

{contexte alphanumerique} ::= {contexte alphabetique} ⊥
{contexte numerique} ⊥ {contenu du contexte}

{contexte alphabetique} := ({contenu du contexte})

{contexte numerique} := < {contenu du contexte} >

{contenu du contexte} ::= {chaine} , {etendue} ⊥ {etendue} ⊥ {chaine}

{etendue} ::= {longueur} ⊥ {nombre sans signe}

{liste alphanumerique} ::= {element alphanumerique} ⊥
{liste alphanumerique} | {element alphanumerique}

{element alphanumerique} ::= {expression alphanumerique} ⊥
({modele de recherche})

{expression alphanumerique} ::= {variable} ⊥ {constante}

{variable} ::= {chaine} ⊥ {sous-chaine}

{chaine} ::= {identificateur}

{identificateur} ::= {minuscule} ⊥ {identificateur} {minuscule} ⊥
{identificateur} {chiffre}

{minuscule} ::= a ⊥ b ⊥ c ⊥ d ⊥ e ⊥ f ⊥ g ⊥ h ⊥ i ⊥ j ⊥ k ⊥ l ⊥ m ⊥ n ⊥
o ⊥ p ⊥ q ⊥ r ⊥ s ⊥ t ⊥ u ⊥ v ⊥ w ⊥ x ⊥ y ⊥ z

{chiffre} ::= 0 ⊥ 1 ⊥ 2 ⊥ 3 ⊥ 4 ⊥ 5 ⊥ 6 ⊥ 7 ⊥ 8 ⊥ 9

{sous-chaine} ::= {chaine} ({amplitude})

{longueur} ::= ! {variable}

{amplitude} ::= {borne} ⊥ {borne} : ⊥ : {borne} ⊥ {borne} : {borne}

{borne} ::= {chaine} ⊥ {nombre} ⊥ {longueur}

{nombre} ::= {nombre sans signe} ⊥ {signe} {nombre sans signe}

{signe} ::= {operateur additif}

{constante} ::= {texte} ⊥ {nombre}

{texte} ::= " {sequence de caracteres} "

{sequence de caracteres} ::= {caractere} ⊥ {sequence de caracteres} {caractere}

{caractere} ::= {chiffre} ⊥ {minuscule} ⊥ {majuscule} ⊥
= ⊥ < ⊥ ; ⊥ : ⊥ ' ⊥ > ⊥ * ⊥ (⊥) ⊥ _ ⊥ + ⊥ ! ⊥
- ⊥ & ⊥ < ⊥ # ⊥ , ⊥ . ⊥ ¬ ⊥ ? ⊥ / ⊥ |

{majuscule} ::= A ⊥ B ⊥ C ⊥ D ⊥ E ⊥ F ⊥ G ⊥ H ⊥ I ⊥ J ⊥ K ⊥ L ⊥ M ⊥ N ⊥
O ⊥ P ⊥ Q ⊥ R ⊥ S ⊥ T ⊥ U ⊥ V ⊥ W ⊥ X ⊥ Y ⊥ Z

REFERENCES BIBLIOGRAPHIQUES

REFERENCES BIBLIOGRAPHIQUES

{1} BARBEDIENNE A., DELIGNE M.J., VINCENT-CARREFOUR A., 1969 L'enseignement assisté par calculateur à Lannion. In : l'enseignement programmé, vol. 6, (Dunod), pp 33-44.

{2} BELLISSANT C., 1969, MAGISTER, un langage d'écriture de cours. In : l'enseignement programmé, vol. 7, (Dunod), pp 75-85.

{3} BELLISSANT C., 1970, Teaching and Learning languages. In : Proceedings of the IFIP World Conference on Computer Education.

{4} BESTOUGEFF H., FARGETTE J.P., LE CORRE Y., 1968, Une expérience de vérification des connaissances à l'aide d'un ordinateur. In : l'enseignement programmé, vol. 1, (Dunod), pp 23-27.

{5} BOLLIET L., 1967, Notation et processus de traduction des langages symboliques. Thèse d'Etat. Faculté des Sciences de Grenoble.

{6} CAUSSE B., 1968, Enseignement pratique de la programmation à l'aide d'un calculateur. Rapport technique, Faculté des Sciences de Toulouse. 45 pp.

{7} CAUSSE B., PLANCHON J.C., RAYNAUD R., 1968, Cours programmé et exercices d'ALGOL. Faculté des Sciences de Toulouse.

{8} CLOUZOT O., 1969, La méthode d'analyse sémantique d'un contenu. In : l'enseignement programmé, vol. 6 (Dunod), pp 21-32.

{9} COULSON J.E., 1962, Programmed Learning and Computer-based Instruction, (John Wiley and Sons), 291 pp.

{10} DELIGNE M.J., 1969, Langages d'écriture des cours en enseignement programmé. Rapport de D.E.A., C.N.E.T., 31 pp.

{11} DONIO J., 1969, Une utilisation du calculateur pour l'enseignement en médecine. Une application au domaine de l'hématologie. In : l'enseignement programmé, vol. 6, (Dunod), pp 41-50.

{12} FEURZEIG W., PAPERT S.A., 1968, Programming-languages as a conceptual framework for teaching mathematics. In : Tendances actuelles de la recherche en enseignement programmé, Sciences du comportement, vol. 8, (Dunod), pp 233-248.

{13} FORSYTHE G.E., 1968, Computer Science and Education. In : Proceedings of the IFIP Congress 68, vol. 2, pp 1025-1039.

{14} FRYE C.H., BENNIK F.D., FEINGOLD S.L., 1968, User's guide to PLANIT. System Development Corporation, Technical memorandum, 96 pp.

{15} GAINES B.R., 1968, Adaptively controlled instruction for a tracking skill. In : Tendances actuelles de la recherche en enseignement programmé. Sciences du comportement, vol. 8, (Dunod), pp 321-336.

{16} GASTINEL N., 1960, Correction automatique de tests à l'Université de Grenoble, In : Chiffres (Revue de l'association française de calcul), vol. 3, pp 237-245.

{17} GRIFFITHS M., 1969, Analyse déterministe et compilateurs. Thèse d'Etat. Faculté des Sciences de Grenoble, 81 pp.

{18} D'HAINAUT L., 1968, Une tentative de perfectionnement des cours programmés à réponse construite par adjonction d'une mémoire et rédaction à l'aide de séquences fonctionnelles. In : l'enseignement programmé, vol. 3, (Dunod), pp 39-49.

{19} HOUZIAUX M.O., 1965, Les fonctions didactiques de DOCEO, système adaptatif d'enseignement automatique. In : Actes du XII° Colloque de l'A.U.P.E.L.F..

{20} I.B.M., CP/CMS User's Guide. Ref. 320-2015, IBM Scientific Center. Cambridge, Massachussets.

{21} LANDA L.N., 1966, Diagnostic et enseignement programmé. In : Actes de la 4° Conférence Internationale sur l'enseignement programmé et les machines à enseigner. (Gesellschaft für Programmierte Instruktion. Klett und Oldenburg).

{22} MARTIN G.A., 1968, Analyse logique des réponses en enseignement automatisé par calculateur. In : Tendances actuelles de la recherche en enseignement programmé. Sciences du comportement, vol. 8, (Dunod), pp 296-302.

{23} PASK G., 1968, Adaptive machines. In : Tendances actuelles de la recherche en enseignement programmé. Sciences du comportement, vol. 8, (Dunod), pp 251-261.

{24} PELTIER M., 1969, The Macro-System. Etude n° FF2.0076. Centre Scientifique IBM de Grenoble, 19 pp.

{25} PERRIAULT J., 1968, Typologie, objectifs et conditions de développement des machines à enseigner en France. (Maison des Sciences de l'Homme.) 29 pp.

{26} PERRIAULT J., 1970, Inventaire des méthodes d'analyse des matières en vue de leur enseignement. Cours. (UER Didactique des Disciplines Scientifiques, Paris).

{27} POHL L., 1968, Signification et importance des algorithmes dans l'enseignement des langues vivantes. In : l'enseignement programmé, vol. 2, (Dunod), pp 49-57.

{28} ROMISZOWSKI A.J., 1968, L'utilisation de l'enseignement programmé dans l'industrie britannique. In : l'enseignement programmé, vol. 2, (Dunod), pp 13-27.

{29} SCHESTAKOW A.W., 1968, L'enseignement programmé et les machines à enseigner en U.R.S.S.. Sciences du comportement, vol. 5, (Dunod), 210 pp.

{30} SILVERN G.M. and SILVERN L.C., 1966, Programmed-instruction and Computer-assisted instruction. An overview. In : Proceedings of the IEEE, vol. 54, n° 12, décembre 1966, pp 1648-1655.

{31} SUPPES P., JERMAN M., BRIAN D., 1968, Computer-assisted Instruction : Stanford's 1965-66 arithmetic program, (Academic Press), 385 pp.

{32} SUPPES P., 1968, Computer Assisted Instruction : An overview of operations and problems. In : Proceedings of the IFIP Congress 68 vol. 2, pp 1103-1112.

{33} THOMPSON F.B., LOCKEMANN P.C., DOSTERT B., DEVERILL R.S., 1969, REL : A rapidly extensible language system. In : Proceedings of the 24th National Conference of the ACM. (ACM publication), pp 399-417.

{34} TONGE F.M., 1968, Design of a programming language and system for computer assisted learning. In : Proceedings of the IFIP Congress 68, vol. 2, pp 1349-1355.

{35} WEIZENBAUM J., 1968, ELIZA : A computer program for the study of natural language communication between man and machine. In : C.A.C.M., vol. 9, n° 1, janvier 1968, pp 36-45.

{36} ZINN K.L., 1968, Languages for programming conversational use of computers in instruction, Proceedings of the IFIP Congress 68, vol. 2, pp 1388-1394.

TABLE DES MATIERES

C H A P I T R E I

LE SYSTEME GENERAL

A - TERMINOLOGIE

On appellera ici "Système Général d'Enseignement assisté par ordinateur" ou "Système général" ou encore plus laconiquement "Système", tout ce qui est mis en oeuvre (matériel, méthodes, programmes) pour établir un apprentissage automatisé. Le matériel joue très souvent un rôle prépondérant dans l'élaboration d'un système et conditionne fortement le choix des méthodes utilisées et l'écriture des programmes.

Par "programme", il faut comprendre ici tout composant interne du système et non pas un cours programmé qui est soumis à l'élève. La "programmation" est donc naturellement la technique d'écriture des programmes ce qui est l'apanage des "programmeurs". On réservera l'expression "rédacteur de cours" ou plus brièvement "rédacteur" aux premiers utilisateurs d'un système, c'est-à-dire en fait aux enseignants.

Ces précisions de vocabulaire peuvent paraître superflues. En fait, les plus grandes confusions dans ce domaine tiennent aux divers sens donnés aux mots "programme", "programmation", "programmeur", voire "programmateur". Le mot "système" a lui aussi connu grand nombre de vicissitudes, et malheureusement, il n'y a pas là de palliatif satisfaisant.

B - REFERENCES

C'est autour de 1960 que sont apparus les premiers systèmes d'enseignement assisté par ordinateur. Le mot "système" est peut-être ambitieux pour définir les expériences de l'époque. L'ordinateur était utilisé à un