Contribution à l'étude des problèmes des terminaux graphiques, un système de programmation graphique conversationnelle

Olivier Lecarme

N° d'ordre

THESES

présentées à la

Faculté des Sciences de Grenoble

pour obtenir

LE GRADE DE DOCTEUR ES SCIENCES APPLIQUEES

par

Olivier LECARME

Licencié es Sciences, Docteur de 3ème cycle

Première thèse

CONTRIBUTION A L'ETUDE DES PROBLEMES D'UTILISATION DES TERMINAUX GRAPHIQUES

UN SYSTEME DE PROGRAMMATION GRAPHIQUE CONVERSATIONNELLE

Deuxième thèse

PROPOSITIONS DONNEES PAR LA FACULTE

Thèses soutenues le 28 Septembre 1970, devant la Commission d'Examen

M. J. KUNTZMANN Président

MM. L. BOLLIET
P. FEAUTRIER Examinateurs
N. GASTINEL

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

Je tiens à remercier :

Monsieur le professeur Jean Kuntzmann, Directeur du Service de Mathématiques Appliquées, qui a bien voulu me faire l'honneur de présider le jury de cette thèse.

Monsieur le professeur Noël Gastinel, Directeur du Laboratoire de Calcul et de l'Institut de Programmation, qui s'est vivement intéressé à mon travail et m'a prodigué ses conseils et ses critiques.

Monsieur le professeur Louis Bolliet, qui a suivi et conseillé tout mon travail en programmation depuis ma thèse de troisième cycle.

Monsieur P. Feautrier, qui a bien voulu faire partie du jury de cette thèse.

Tous mes collègues du laboratoire qui m'ont aidé dans la réalisation de ce travail de leurs remarques, de leurs conseils ou de leur contribution active, en particulier Messieurs Bellissant, Cleemann, Hans, Lucas et Siret.

Je n'ai à remercier aucun service de dactylographie, puisque j'ai moi-même procédé à la composition de ce texte, mais j'ai été aidé dans sa correction par mon épouse et Monsieur Lucas, et dans son impression par Messieurs Bellot et Cistac. Enfin, Monsieur Mounet et le service de tirage ont terminé la réalisation matérielle de cet ouvrage.

CONTRIBUTION A L'ETUDE

DES PROBLEMES D'UTILISATION

DES TERMINAUX GRAPHIQUES

UN SYSTEME DE

PROGRAMMATION GRAPHIQUE CONVERSATIONNELLE

A ma femme

TABLE DES MATIERES

Tables

TABLE DES FIGURES

0 - PREAMBULE

La présente thèse est l'aboutissement de six années de travaux, que j'ai réalisés à la fois comme chercheur et comme enseignant puisque cela a été mon rôle d'abord comme assistant puis comme maître-assistant à la Faculté des Sciences de Grenoble. Certains de ces travaux représentent des réalisations pratiques (6, 8, 10 et 11), d'autres ont conduit à l'écriture de cours polycopiés pour les étudiants de l'Institut de Programmation, de la maîtrise ou du D.E.A. (3, 4, 9, 10, et 17), d'autres encore représentent ce que l'on est plus convenu d'appeler de la recherche: thèse de troisième cycle (1), séminaires (2, 5 et 12), articles (14 et 15) et communications (7, 13 et 16).

L'ensemble de ces travaux peut être réparti entre trois grands centres d'intérêt: les langages de programmation, les systèmes d'exploitation et les traitements graphiques. Ces trois centres d'intérêt se retrouvent dans le sujet princiapl de cette thèse, ou un langage de programmation spécial est utilisé dans le cadre d'un système tout aussi spécial, à seule fin de permettre d'effectuer des traitements graphiques.

Les travaux que l'on peut classer dans le domaine des langages de programmation sont les suivants:

1 - Thèse de doctorat de spécialité (troisième cycle) [Le65-1][1] : Etude comparative des principaux langages de programmation. J'y ai acquis la connaissance d'une douzaine de langages différents, ce qui constitue une expérience sans prix.

2 - Séminaire de programmation sur la comparaison des "langages de listes" [Le64], en préliminaire à la thèse ci-dessus.

3 - Cours polycopié sur le langage IPL-5 [Le66-1], adapté de l'anglais, qui m'a fait connaître un certain nombre de techniques de base dans la manipulation de données non élémentaires.

4 - Cours polycopié sur le langage machine de l'ordinateur IBM 7044 [Le65-2], en collaboration avec M. Mermet; ce travail m'a aidé à dégager les outils fondamentaux de toute programmation.

5 - Séminaire de programmation sur le langage AED [Le69-1], qui m'a montré certaines options intéressantes dans la définition d'un

---

[1]Les références entre crochets renvoient à la bibliographie en Annexe 1.

langage de programmation, en particulier d'outils importants mais non élémentaires.

6 - Compilateur Algol sur l'ordinateur IBM 1130 [BeLe67-1, BeLe67-2], en collaboration avec M. Bellissant. Ce travail important m'a beaucoup appris, en particulier sur les problèmes de génération d'un programme interprété, et la méthode que j'y ai appliquée est à ma connaissance la seule mise en oeuvre effective de l'idée proposée par A.A. Grau [Gr61].

Dans le domaine des systèmes d'exploitation des ordinateurs, les travaux que j'ai réalisés sont les suivants:

7 - Communication au cinquième congrès de l'AFIRO sur l'écriture en Algol des systèmes [Le66-2]. Ce travail m'a montré que l'écriture en langage évolué d'un système d'exploitation était utile aussi bien pour faciliter la construction du système que pour exposer clairement son fonctionnement.

6bis - Compilateur Algol sur IBM 1130. Dans le domaine des systèmes, ce travail m'a fait toucher du doigt les problèmes qui se posent quand on veut ajouter quelque chose à un système existant, en particulier un sous-système nouveau et non prévu à l'origine.

8 - Interface du système COMIT II pour fonctionnement sous le système IBSYS de l'ordinateur IBM 7044 [Le67]. Cette réalisation pratique m'a montré l'importance des problèmes de compatibilité et l'utilité de la définition claire et précise des quelques noeuds qui constituent un interface.

9 - Cours sur les systèmes d'exploitation [Le70-1]. Ce cours, d'un volume très important à cause de la généralité du sujet, enseigné à l'Institut de Programmation en particulier, m'a demandé un très gros travail de documentation, qui a été très utile au travail qui constitue cette thèse, dans le domaine des systèmes conversationnels. D'autre part, il m'a montré qu'il était tout à fait praticable d'utiliser l'ordinateur pour la composition et le tirage d'un texte de volume important [IBM5, IBM9].

Enfin, les travaux qui se rapportent au domaine des traitements gtaphiques sonr les suivants:

10 - Définition du langage LAGROL [LeLu68], en collaboration avec M. Lucas. Ce travail m'a aidé à déterminer quels pouvaient être les outils élémentaires à fournir par un langage de programmation graphique. Il constitue d'autre part un deuxième exemple de langage interprété, et le système GENIAL qu'a réalisé à partir de là M. Lucas a montré son utilité.

11 - Réalisation du "support graphique" pour le système CMS [HaLe70], en collaboration avec M. Hans, et insertion des sous-programmes G.S.P. dans ce système. Ce travail assez délicat m'a familiarisé avec le fonctionnement interne du support graphique du

système O.S. et des sous-programmes G.S.P. C'etait d'autre part une condition sine qua non de la réalisation du système Eulalie.

12 - Séminaire de programmation sur une première présentation du langage Euphémie et du système Eulalie (qui n'avaient pas encore de noms à l'époque) [LeC168], en collaboration avec M. Cleemann.

13 - Communication au Colloque sur les systèmes conversationnels organisé à Grenoble [Le68], pour présenter le côté conversationnel du système Eulalie.

14 - Article dans la Revue française d'informatique et de recherche opérationnelle, sur les langages de programmation graphique [CLL68], en collaboration avec MM. Cleemann et Lucas.

15 - Article en anglais proposé à l'Annual review of automatic programming [Le69-2], sur le même sujet que le séminaire.

16 - Communication à la deuxième Journée de visualisation de l'AFCET [Le69-3], sur le niveau des langages et la puissance des outils en programmation graphique (à paraître dans la Revue française d'informatique et de recherche opérationnelle [Le70-2]).

17 - Manuel d'utilisation des sous-programmes G.S.P. [LeSa70], en collaboration avec M. Saillard.

A part les deux premiers et le dernier, tous les travaux cités au paragraphe précédent se rapportent au même sujet et montrent divers points du même projet, dans divers états d'avancement. La thèse de troisième cycle de M. Cleemann est également un bon moyen de faire le point de la situation à l'époque où elle a été présentée. Un aspect fondamental du présent travail est en effet qu'il s'agit d'un travail réalisé en collaboration:

- Le langage "Macro-Euphemie" a été défini et mis en oeuvre par M. Cleemann.
- Le langage Euphemie a été defini conjointement par M. Cleemann et moi-même, en utilisant beaucoup des résultats acquis dans la réalisation de "Macro-Euphémie".
- Le traducteur et le retraducteur ont été réalisés par M. Cleemann.
- Les modules de gestion de l'écran et d'interprétation ont été réalisés par MM. Guihard et Guillou.
- Enfin, d'autres personnes de l'Institut de Mathématiques Appliquées ou d'ailleurs sont intervenues plus ou moins par des idées et des conversations, en particulier M. Lucas, mais aussi MM. Cagnat, Siret et Hans, ainsi que M. Loutrel (de la C.I.I.) et M. Craig Johnson (du Centre scientifique IBM de Cambridge).

Le sujet de cette thèse est longuement indiqué dans son titre, et le plan que je vais suivre s'en dégage aisément: une brève introduction examine la situation actuelle des traitements graphiques par ordinateur. Le chapitre 2 propose dans un exposé de ce

qu'est l'utilisation des terminaux graphiques une terminologie assez complète. Le chapitre 3 examine le cas général des langages de programmation graphique, et conduit tout naturellement au chapitre 4 à l'exposé d'un langage particulier que je propose, le langage Euphémie ("la bien disante" en grec). Le chapitre 5 concerne alors l'utilisation de ces langages de programmation graphique de facon conversationnelle, et conduit au chapitre 6 à la description d'un système particulier adapté au langage précédent, le système Eulalie ("la bavarde" en grec). Enfin, après une brève conclusion les annexes fournissent une bibliographie assez étendue, un glossaire résumant la terminogie du chapitre 2, une description syntaxique du langage Euphémie en Snobol4, et une description du système Eulalie en langage Euphémie.

1.1 - ESPOIRS ET DECEPTIONS.

L'évolution au cours des années de l'idée que l'on s'est faite de l'utilisation des terminaux graphiques n'est pas sans rappeler ce qui s'est passé dans le domaine des systèmes en temps partagé: les premières réalisations expérimentales des années 60 à 64 conduisent à l'euphorie générale de 1965, où des projets toujours plus grandioses sont proposés à l'enthousiasme des foules. Suit une période de désenchantement quand on s'aperçoit que ce qui est tenu est bien loin de ce qui était promis. Actuellement, l'espoir semble renaître et le mouvement en avant reprendre, mais pas toujours dans les directions que l'on envisageait il y a cinq ans.

Dans le domaine des terminaux graphiques, l'évolution a été beaucoup plus lente, probablement à cause du plus petit nombre de gens intéressés, mais elle suit un schéma semblable, qui n'est pas encore parvenu à l'étape où le mouvement en avant a réellement repris. La période des premières réalisations expérimentales se situe vers les années soixante, et les systèmes les plus marquants sont celui de la General Motors et le système Sketchpad d'I.E. Sutherland au M.I.T. [Su63]. L'année de l'euphorie est 1963, comme peut le montrer en particulier la liste des communications présentées lors de la Spring Joint Computer Conference de l'AFIPS de cette année, avec en particulier des textes de S.A. Coons, T.E. Johnson [Jo63], D.T. Ross, R. Stotz et I.E. Sutherland [Su63]. La période du désenchantement et de la frustration commence peu après, et on peut dire qu'elle dure encore.

Il me semble bon d'analyser quels étaient les espoirs de l'année euphorique, et d'où viennent les déceptions qui ont suivi. Nous pourrons ainsi nous demander si l'on a eu tort de concevoir les espoirs, ou si l'on a failli dans les tentatives de réalisation.

1.2 - LA SITUATION EN 1963.

Le résultat le plus clair, sur un public pourtant compétent, de la publication des travaux réalisés à la General Motors et par I.E. Sutherland, a été de faire croire que les traitements graphiques ne demandaient aucun travail à l'utilisateur, et qu'il suffisait

d'arriver les mains dans les poches devant le terminal, sans trop savoir exactement ce que l'on voulait faire; le "cerveau électronique", aussi compréhensif et intuitif que rapide et infaillible, se chargerait lui-même de la définition exacte du problème, et évidemment de sa résolution. La notion même de programme n'avait aucune raison d'être, a fortiori celle de langage de programmation, et tout se passerait dans un dialogue entre l'être humain et sa machine: au moyen de son "crayon lumineux" (qui en fait n'écrit pas et ne produit pas de lumière), l'homme dessinerait sur l'écran une approximation de ce qu'il voudrait, puis il n'aurait plus qu'à dire à la machine: "- Arrange-moi ça!" pour que le dessin soit parfait; il pourrait dire aussi: "- Montre-moi ça de l'autre côté!", ou "- Agrandis-moi ce détail!", et la machine effectuerait miraculeusement tout ce qu'il demanderait; comment parlerait-il à sa machine? En frappant sur une machine à écrire, ou même en parlant à haute voix, ou mieux encore, qui sait, en se contentant de penser!

Le côté "science-fiction" de ce schéma volontairement grossi n'était pas le plus grave, il s'en faut de beaucoup, car il n'impliquait pas une méconnaissance totale des problèmes des traitements graphiques, mais une extrapolation un peu exagérée des possibilités des systèmes qui venaient d'être réalisés en 1963. En fait, beaucoup de gens ont fait pendant longtemps et font encore de parfaits contre-sens sur la nature et l'utilité des traitements graphiques. Les constructeurs de matériel sont eux-mêmes souvent responsables de cet état de fait, en particulier quand ils "font l'article" à d'éventuels clients en proposant des exemples d'utilisation des terminaux graphiques qui sont de parfaits modèles de ce qu'il ne faut pas faire, soit qu'il s'agisse de simples démarquages de problèmes résolus par d'autres moyens (y compris sans ordinateur du tout), soit qu'il s'agisse de démonstrations coûteuses et spectaculaires mais sans la moindre utilité.

1.3 - LES RAISONS DU DESENCHANTEMENT.

Une premiere raison de déception particulièrement évidente apparaît à l'acheteur éventuel d'un terminal graphique dès que le constructeur du matériel en arrive aux détails sordides: un terminal graphique est actuellement un outil extrêmement coûteux, ce qui interdit carrément son utilisation aux laboratoires à moyens restreints, et ne permet pas à ceux qui sont plus riches de s'en procurer plusieurs, sauf dans le cas des utilisateurs à moyens particulierement fastueux, tels que les militaires par exemple.

Une deuxième raison moins évidente mais beaucoup plus grave parceque plus fondamentale, apparaît à l'acheteur pressenti quand, après avoir posé la question: "- Que peut-on faire avec votre appareil?" et avoir obtenu du commerçant la réponse: "- Tout!", il demande comment on peut le faire. La réponse qu'il obtiendra sera bien entendu que "c'est très simple", mais il s'apercevra bien vite que, pour réaliser la moindre des applications qu'il a pu envisager, il lui faudra assumer un effort de programmation tout à fait disproportionné. Il constatera ainsi l'absence totale de langage de programmation graphique spécialisé, et la disproportion tout aussi totale qui sévit entre les outils fournis par les terminaux et les langages qui permettent de mettre en oeuvre ces outils.

Si un utilisateur riche et inconscient passe outre, malgré ces deux premières raisons de prudence, et connecte un terminal graphique à son ordinateur, c'est très souvent pour des raisons de mode et de prestige qui n'ont pas grand'chose à voir avec la recherche scientifique et l'emploi rationnel d'un ordinateur (ou du moins elles ne le devraient pas). Après une première période d'engouement où le terminal fonctionne sans arrêt sur des programmes de démonstration fournis par le constructeur, arrivent alors les nouvelles déceptions qui font qu'au bout de quelque temps on ne sait plus quoi faire du terminal. Est-ce le matériel qui est mauvais? Non, et de toutes façons on peut faire du travail honnête avec du matériel médiocre; le fond du problème est que l'on n'a pas vu quelle était l'utilité fondamentale du terminal graphique, et que l'on a voulu conserver avec ce nouvel outil des habitudes de travail et de pensée qui ne lui sont pas adaptées.

Si en effet on ne cherche dans un terminal graphique qu'un moyen de présenter les résultats d'un programme sous forme graphique, il vaut bien mieux utiliser une table traçante, qui permet de faire des dessins beaucoup plus grands, précis et complexes, ou un traceur sur microfilms, qui permet le remplissage des surfaces et les demi-teintes, et produit comme la table traçante une information non fugitive. Si l'on veut utiliser le terminal comme un moyen de conversation alphanumérique plus commode dans certains cas qu'une machine à écrire, il vaut bien mieux opter pour un terminal alphanumérique spécialisé. Tous les appareils que je viens de citer ne coûtent en effet qu'une fraction souvent faible du prix du terminal graphique, et leur utilisation est généralement beaucoup plus simple. En caricaturant un peu, je pourrais ajouter que si l'on veut utiliser le terminal pour faire des dessins, il vaut bien mieux prendre un crayon, une gomme et une rame de papier, dont on peut se servir chez soi et pour pas cher.

1.4 - QUE FAIRE AVEC UN TERMINAL GRAPHIQUE?

Un terminal graphique n'est-il donc qu'un jouet pour P.D.G. ou pour millionaire? Sûrement pas, à condition que l'on en connaisse bien toutes les limites et toutes les possibilités. Ceci serait d'ailleurs intégralement vrai s'agissant de ce jouet encore plus coûteux qu'est l'ordinateur. Commençons donc par savoir ce que l'on ne peut pas faire avec un terminal graphique, ou plus exactement quels sont ses défauts et ses manques.

Les qualités graphiques de l'image produite sont médiocres; l'écran est de dimensions assez faibles, en général trente centimètres sur trente, parfois encore moins; le faisceau lumineux produit un point de dimensions non négligeables (entre un et cinq dixièmes de millimètre), aussi est-il illusoire d'attendre une excellente précision de tracé; la définition des points de l'écran tient compte de ce fait, et les lignes auront généralement la forme d'escaliers; il faut un certain temps pour afficher quoi que ce soit, et toute image doit être régénérée à une fréquence au moins égale à la persistance rétinienne, aussi ne peut-on afficher d'images tres complexes; pour la même raison, on ne peut pas afficher de "belles" courbes, car celles-ci doivent être approchées au moyen de petits segments de droites en nombre forcément restreint; le dessin est purement linéaire, excluant le remplissage des surfaces et les demi-teintes. En bref, même avec un excellent matériel, on ne peut espérer obtenir que des schémas filiformes et géométriques, même si l'information conservée dans l'ordinateur est beaucoup plus précise.

L'image produite est fugitive, c'est-à-dire que sa conservation sous une forme ou une autre est sous la responsabilité de l'ordinateur. Le mode de génération de l'image est imposé par la technologie du matériel, et n'a aucun rapport simple avec les propriétés géométriques ou topologiques des objets affichés, c'est-à-dire que le programme qui doit produire un dessin ne peut pas "penser" en termes géométriques, ou bien alors doit effectuer une transformation depuis une représentation qui traduit ces propriétés jusque dans la représentation adaptée à la technologie du terminal. Ces deux raisons impliquent que la programmation d'un terminal graphique dans le langage de l'ordinateur auquel il est connecté est lourde, difficile et délicate.

Les moyens de communication que l'on ajoute au terminal sont à première vue très séduisants, mais difficiles à utiliser: quand on introduit de l'information alphanumérique, on s'attend à la voir apparaître en même temps sur l'écran, mais cela suppose des moyens

technologiques complexes et une programmation non naturelle; le moyen de désignation qu'est le pointeur optique produit une information qu'il est difficile de lier à la logique de l'image à cause des problèmes de génération cités plus haut; comme moyen d'introduction de coordonnées, il est catastrophique, et il a fait perdre à beaucoup de gens un temps précieux qui aurait pu être mieux utilisé; les moyens spécifiques d'introduction de coordonnées, tels que la tablette Rand, le crayon capacitif ou le "manche à balai", sont très difficiles à utiliser comme moyens de désignation.

Mais le plus gros défaut peut-être des terminaux graphiques est le nombre immense de solutions que l'on peut adopter dans toutes les étapes de la conception d'un système de visualisation graphique: définition des outils élémentaires d'affichage, choix du mode d'entretien et du mode de balayage de l'image, adjonction de moyens de communication plus ou moins farfelus, mode de connexion à l'ordinateur, etc. En fait, les constructeurs de matériel luttent entre eux à qui mettra le plus de perfectionnements sur le terminal qu'il fabrique, mais ils semblent se préoccuper beaucoup moins de ce à quoi pourront servir tous ces "gadgets" coûteux et spectaculaires (ce qui est la marque d'une technologie encore trop jeune). Ceci a pour résultat qu'il n'existe pas deux terminaux qui se ressemblent, et qu'il est par conséquent à peu près impossible de présenter des solutions générales aux problèmes; en particulier, la définition de moyens d'utilisation universels ou simplement généraux, langages de programmation ou autres, est presque aussi utopique que celle d'une monnaie mondiale.

Ceci posé, le terminal graphique a deux qualités fondamentales et capitales; aucune de ces deux qualités ne lui est propre, mais son originalité est qu'il les rassemble dans un même outil. La première qualité est qu'il s'agit d'un moyen de produire une information graphique, c'est-a-dire qu'il utilise une des facultés fondamentales de l'être humain, qui est la compréhension globale et intuitive desinformations visuelles; ce point est commun aux traceurs de courbes, tables traçantes et traceurs sur microfilm, et chacun s'accorde à reconnaître qu'un simple dessin vaut infiniment mieux qu'un tableau de résultats numériques ou qu'un long discours. La deuxième qualité est qu'il s'agit d'un moyen conversationnel, c'est-à-dire qu'il utilise une autre des facultés fondamentales de l'être humain, qui est la possibilité de réaction immédiate à une information par prise d'une décision; ce point est commun aux terminaux alphanumeriques ou aux divers types de machines à écrire, et l'emploi conversationnel des ordinateurs est une des plus grandes transformations qui se sont produites dans l'informatique.

La conjonction de ces deux qualités fondamentales dans un même outil permet à l'homme de prendre des décisions au vu d'une image, ce qui est une de ses activités les plus naturelles et les plus productives (voir par exemple la conduite d'un véhicule ou les activités d'une ménagère, ou la plupart des actions de la vie courante). Ce point est d'une telle importance qu'il permet de passer outre les défauts cités précédemment: tant pis si l'image n'est pas très précise, on n'a pas besoin de faire des mesures ou de la regarder à la loupe; tant pis si elle est schématique, ou plutôt tant mieux, car elle n'indiquera que ce qui est important; peu importe si les outils de communication sont imparfaits, le tout est qu'ils existent. En fait, il est absolument certain que les terminaux graphiques répondent à un besoin fondamental, et que leur avenir ne dépend que de la suppression ou au moins de la réduction de leurs trois défauts les plus importants: coût élevé, manque de compatibilité et pénurie criante d'outils programmés d'utilisation.

Le travail que recouvre cette thèse est un essai limité de réduction du troisième de ces défauts, par la définition d'un moyen d'utilisation des terminaux graphiques qui réduise les difficultés de programmation en maintenant intactes les possibilités graphiques et conversationnelles. Ce travail ne peut influer sur les deux autres défauts, en particulier sur le deuxième, et c'est pourquoi il ne peut pas être présenté comme une solution générale, mais comme une simple tentative timide pour tendre vers une telle solution: cette prudence n'est pas exagérée quand on voit avec quelle rapidité évoluent toutes les techniques.

2 - LES CONCEPTS FONDAMENTAUX DES TRAITEMENTS GRAPHIQUES

Ce chapitre va servir à donner un certain nombre de définitions et établir certaines classifications. Il s'agit d'aboutir à une terminologie cohérente et utilisable, afin d'être sûr de savoir exactement de quoi l'on parle, ce qui n'est que rarement le cas quand on utilise des termes vagues et imprécis, mal traduits de l'américain ou même pas traduits du tout.

Il me faut donc définir les outils technologiques dont on dispose pour les traitements graphiques, les outils de programmation, et enfin les outils de dialogue, qui ne sont d'ailleurs pas spécifiquement graphiques. Une classification des langages que l'on utilise dans ces traitements servira d'introduction à la suite de ce travail.

2.1 - LES OUTILS TECHNOLOGIQUES.

Les outils technologiques sont ceux qui font partie intégrante du matériel livré par le constructeur: ils comprennent des outils de génération d'images et des outils de dialogue, ces derniers seront étudiés au paragraphe 2.3.

Parmi tous les dispositifs de production d'information graphique que l'on peut connecter à un ordinateur, la console de visualisation[2] est le dispositif qui utilise un écran cathodique comme moyen d'affichage. Ceci établit déjà la distinction avec les traceurs de courbes et tables traçantes, purement mécaniques, et avec les traceurs sur microfilms. Les consoles de visualisation elles-mêmes peuvent se répartir en deux groupes: un terminal alphanumérique ne peut afficher que des lettres, chiffres et autres symboles pris dans un certain alphabet, à l'exclusion de quoi que ce soit d'autre; un terminal graphique peut afficher au moins des points en tout emplacement de son écran (moyennant une certaine discrétisation), ce qui lui permet de faire apparaître des lignes quelconques, y compris celles qui représentent des caractères.

---

[2]Un glossaire figurant en annexe 2 regroupe les définitions de tous les termes spécialisés utilisés.

Je me m'intéresserai par la suite qu'aux terminaux graphiques, et plus précisement à ceux qui sont d'un type courant, c'est-à-dire qui affichent une information monochrome, ne sont pas capables de remplir des surfaces, et permettent de fabriquer des images à partir de trois tracés élémentaires: points, vecteurs et caractères.

Un des problèmes majeurs des terminaux graphiques classiques est que l'image affichée sur l'écran est fugitive; il faut donc l'entretenir, c'est-a-dire la régénérer à une certaine fréquence. Une façon intéressante de le faire est de confier ce travail de régénération au terminal lui-même, pour éviter de monopoliser pour cela un ordinateur puissant; dans ce cas, les ordres de génération sont placés dans une mémoire d'entretien associée au terminal et dans laquelle celui-ci les trouve l'un après l'autre, de même que l'unité centrale d'un ordinateur trouve dans sa propre mémoire les instructions du programme à exécuter. Ce programme d'affichage exécutable par le terminal constitue la liste d'affichage. Dans le cas d'un petit ordinateur, on accepte de se passer de mémoire d'entretien, mais la machine ne peut quasiment rien faire d'autre.

Pour produire une image, il faut donc construire dans la mémoire de l'ordinateur la liste d'affichage voulue, puis l'envoyer dans la mémoire d'entretien par une opération qui pour l'ordinateur est semblable à une écriture sur disque ou tambour, et enfin transmettre au terminal l'ordre de commencer la régénération à partir d'une certaine adresse dans la mémoire d'entretien. Au cas où le terminal n'a pas de mémoire d'entretien, l'ordinateur lui envoie un à un les ordres qui constituent la liste d'affichage, puis recommence au début quand il a terminé; étant donnée la vitesse à laquelle il doit le faire, il n'a pratiquement jamais le temps de faire autre chose, même si le terminal lui signale par une interruption le moment où il peut envoyer le prochain ordre à exécuter.

2.2 - LES OUTILS DE PROGRAMMATION.

2.21 - CLASSIFICATION DES OUTILS.

Les outils nécessaires à la programmation graphique peuvent être classés en trois grandes catégories.

Pour construire les objets graphiques qui composent l'image à afficher, on a besoin d'outils de génération, qui servent en fait à constituer la liste d'affichage; si l'on considère ce travail d'assez haut pour se désinteresser de la façon dont il est effectivement réalisé, on peut dire qu'il s'agit d'outils de

définition, de même que les déclarations des langages à haut niveau servent à définir les objets sur lesquels on va travailler, en leur donnant un nom et des caractéristiques.

Une fois la liste d'affichage composée, on la transmet au terminal, et on pourra ensuite la modifier directement dans la mémoire d'entretien, arrêter la régénération, la relancer à telle ou telle adresse, bref effectuer sur l'ordinateur les opérations qui correspondent aux opérations d'entrée et sortie des dispositifs périphériques classiques; on a besoin pour cela d'outils de transmission, qui servent donc à commander le terminal par l'envoi d'informations; si l'on considère ce travail depuis la même altitude que précédemment, on peut parler d'outils d'utilisation des objets définis.

Enfin, la conversation entre l'ordinateur et le terminal n'est pas à sens unique, elle comprend le transfert d'informations du terminal vers l'ordinateur, par le moyen d'interruptions liées à des dispositifs précis et d'opérations de lecture lancées par l'ordinateur; ceci correspond à des outils de communication, qui sont la caractéristique fondamentale des consoles de visualisation parmi les périphériques qui produisent une information graphique.

2.22 - LES OUTILS DE GENERATION.

Les outils de génération peuvent être placés dans des catégories bien hiérarchisées. En commençant par le bas, on trouve l'élément, qui est l'atome technologique; c'est ce qui peut être généré de façon élémentaire par le terminal, c'est-à-dire dans le cas le plus courant un point, un vecteur ou un caractère alphanumérique; certains terminaux comptent l'arc de cercle comme élément, mais ils sont rares et très coûteux.

Au niveau immédiatement supérieur, on trouve l'objet, qui est l'atome programmé; c'est ce qui peut être généré en une seule instruction, c'est-à-dire un ensemble d'éléments (éventuellement un seul) de même nature: point ou suite de points, vecteur ou suite de vecteurs, chaîne de caractères. Il est intéressant, à partir du moment où l'objet est un outil de programmation, de lui donner des propriétés supplémentaires; c'est ainsi que l'on crée des objets de types nouveaux, par exemple la position, qui est un point éteint et sert à placer le faisceau en un endroit précis, la ligne, qui est un vecteur dont l'origine est implicite (une suite de lignes qui font partie du même objet est forcément formée de vecteurs consécutifs), et le segment, qui est un vecteur dont on donne l'origine et

l'extrémité (une suite de segments qui font partie du même objet peut être formée de vecteurs non consécutifs); d'autre part, il est très utile de pouvoir identifier un objet au moyen d'un nom ou d'un numéro, afin de pouvoir s'y référer par la suite pour le modifier, le déplacer ou le faire disparaître.

Le niveau supérieur est celui de la figure, qui regroupe des objets (éventuellement un seul) en un ensemble qui possède des propriétés globales. Parmi ces propriétés on peut citer: - le nom de la figure, qui sert à la désigner pour lui faire subir toutes les manipulations nécessaires; - le système de coordonnées adopté, à definir par des moyens plus ou moins proches de ceux de la géométrie analytique; - la portion d'écran affectée, qui permet l'effacement de ce qui dépasse ce contour; - les divers modes possibles pour la génération des objets, l'envoi de coordonnées, les détections au pointeur optique ou l'introduction de données alphanumériques.

Par exemple, la figure SCHEMA (Figure 4.1) est formée des objets TRIANGLE (construit avec des vecteurs), NUAGE (construit avec des points) et TEXTE (construit avec des caractères). Il peut exister simultanément la figure CENTRE, formée de l'unique objet ORIGINE, construit avec un unique point. Si ces deux figures sont affichées sur l'écran, elles produisent une image, qui est ce qui apparaît sur l'écran à un instant donné.

On constate quelquefois la nécessité d'un regroupement d'objets à l'intérieur d'une figure, pour leur faire subir certaines manipulations simples telles que l'effacement ou le déplacement, sans pour autant associer à cet ensemble toutes les propriétés d'une figure; on utilise alors une séquence, qui comme son nom l'indique regroupe des objets générés consécutivement.

2.23 - LES STRUCTURES.

Une figure représente généralement une information graphique qui a certaines propriétés géométriques, topologiques, et même simplement logiques. Soit une figure représentant un triangle rectangle isocèle ABC de sommet A; ses propriétés géométriques sont l'égalité des segments AB et AC, et la valeur de 90° de l'angle $\widehat{BAC}$; ses propriétés topologiques sont qu'il s'agit d'un triangle, c'est-à-dire par exemple que AB et AC sont concourants; enfin, on peut lui associer des propriétés logiques non graphiques, telles qu'une masse ou une charge électrique en chacun des sommets.

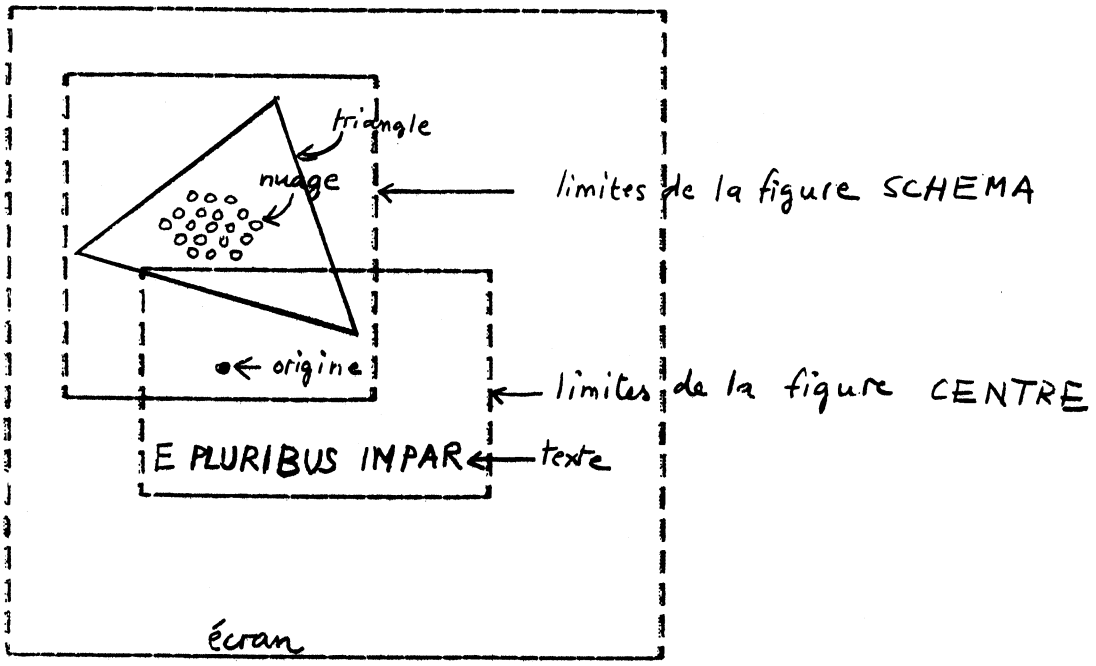


Figure 2.1 - Exemple d'image.

La hiérarchisation des outils adoptée plus haut peut servir à exprimer certaines propriétés très simples, par exemple le fait que les objets TRIANGLE, NUAGE et TEXTE ont entre eux un lien logique quelconque qui explique leur regroupement en une figure. En revanche, ce système est impuissant à exprimer les propriétés géométriques et topologiques, non plus que toute propriété logique un tant soit peu élaborée. On ressent donc très fréquemment la nécessité d'un outil d'un niveau encore supérieur, qui ne sera plus calqué sur le mode de génération des objets mais au contraire sur l'idée que se fait de ces objets leur utilisateur; cet outil d'expression des propriétés est une structure, terme vague qui peut recouvrir à peu près n'importe quoi.

Une structure est un ensemble d'éléments, qui peuvent être ou n'être pas de la même nature, mais dont le tout n'est pas égal à la somme des parties. Un ensemble [a,b,c,d] n'est pas une structure; il en devient un aussitôt qu'on lui fixe une loi, par exemple une loi de groupe. En d'autres termes, une structure est un tout qui, en tant que tout seulement, a des propriétés nouvelles. C'est à ce

titre que l'on peut dire qu'une cellule animale est plus structurée qu'un cristal: il existe plus de relations fonctionnelles entre ses éléments, et donc le tout a davantage de propriétés. On n'obtient pas une structure par addition d'éléments: il faut encore les combiner, c'est-à-dire les mettre en relation.

La liste d'affichage elle-même peut déjà être considérée comme une représentation structurée des informations, puisque en fait elle les renferme, mais sous une forme à peu près impossible à interpréter. Il faut donc disposer d'un outil plus commode, et il est bien certain qu'on est encore loin d'avoir trouvé une représentation structurée qui satisfasse tout le monde. On utilise les tables de corrélation [IBM?], les matrices en coordonnées homogènes [Ro64], les listes simples, les listes symétriques, les structures en arbres, les structures en anneaux [Su63], les "plex" [Ro63], les mémoires associatives simulées contenant des triplets [Sy68], etc., et l'imagination des chercheurs s'est donné libre cours sans avoir encore conduit à la solution idéale.

Un des problèmes les plus graves qui se pose en effet dans la définition et la conception d'une structure est que, en cherchant trop à faciliter l'expression de propriétés particulièrement complexes et globales, on s'éloigne tellement de la façon dont sont effectivement générés les objets que le passage à la liste d'affichage devient d'une lourdeur et d'une complexité rédhibitoires, sans parler du passage en sens inverse qui a de fortes chances d'être encore plus difficile.

2.3 - LES OUTILS DE DIALOGUE.

2.31 - OUTILS TECHNOLOGIQUES.

Les outils de dialogue peuvent être classés d'après la nature de l'information qu'ils permettent d'envoyer du terminal à l'ordinateur.

L'information la plus classique est l'information alphanumérique, qui permet de composer des messages formés de mots et de phrases; on la transmet au moyen d'un clavier de touches semblable à celui d'une machine à écrire, c'est le clavier alphanumérique. Cet outil, malgré sa simplicité apparente, pose un problème délicat: le texte frappé doit apparaître quelque part, et il est logique que ce soit sur l'écran; or, ce qui apparaît sur l'écran est ce qui a été prévu par le programmeur dans la liste d'affichage qu'il a construite; il faut donc que l'introduction de texte depuis le clavier alphanuméri-

que modifie la liste d'affichage pour modifier l'information affichée. Tout ceci impose plusieurs moyens technologiques spéciaux: l'ordre de génération de texte doit déjà exister, et les caractères frappés doivent se placer derrière, à la place d'un texte, blanc ou non, prevu par l'utilisateur; un curseur visualisé sur l'écran indique alors la position où viendra le prochain caractère; pour éviter les fausses manoeuvres, il faut un blocage technologique qui empêche de modifier une information non alphanumérique, et aussi qui empêche de modifier un texte si le programmeur ne veut pas qu'il le soit (caractères protégés); pendant le processus d'introduction du texte, l'ordinateur n'intervient pas, aussi faut-il un moyen de le réveiller pour lui signaler que le texte est complet; il faut alors que l'ordinateur puisse lire l'information envoyée, c'est-à-dire en général lire le contenu de la memoire d'entretien. L'ensemble de ce travail nécessite un nombre assez important d'opérations programmées, et le dialogue par ce moyen n'est pas aussi elémentaire qu'on pourrait le penser; surtout, il est impossible d'envoyer un message si le programme en train de tourner sur l'ordinateur ne l'a pas prévu, c'est-à-dire qu'il est impossible d'attirer l'attention d'un programme qui n'écoute pas.

Une sorte d'information peut-etre plus intéressante parceque plus spécifiquement graphique est l'envoi de coordonnées. Il existe un certain nombre de dispositifs prévus pour cet usage, qui se différencient par la technique adoptée, la précision de l'information fournie et le coût du matériel. Certains permettent de pointer un outil aigu directement sur l'écran (tablette Sylvania [TeSa68]), alors que d'autres nécessitent une surface opaque qu'il faut obligatoirement placer ailleurs (tablette Rand [DaE164]); ce deuxième système, ainsi que celui qui n'envoie pas de coordonnées mais un vecteur d'orientation et de longueur variable (boule roulante, manche à balai, "souris"), nécessitent un travail de programmation non négligeable pour que l'utilisateur puisse savoir à quel endroit precis de l'écran correspond le point qu'il montre. Un premier défaut de ces dispositifs ne leur est pas inhérent mais provient plutôt de l'usage qu'on en fait: il est difficile de relier l'information qu'ils fournissent aux objets qui apparaissent sur l'écran, et il s'agit une fois de plus d'un problème de représentation structurée. Un deuxième défaut peut-être plus grave encore est leur disparité: les constructeurs rivalisent d'ingéniosité dans ce domaine, mais la programmation est bien loin de suivre.

La sorte d'information la plus intéressante sans doute est l'information logique que l'on donne en montrant un objet affiché. Le seul moyen qui existe, et qui donne d'ailleurs toute

satisfaction[3] , est le pointeur optique, simple cellule photo-électrique capable d'envoyer un signal quand elle détecte de la lumière. Le traitement de ce signal occasionne ensuite quelques difficultés, quand on veut savoir quel objet de l'image affichée a été désigné; l'arrêt automatique de la régénération permet de savoir l'adresse de l'ordre que l'on était en train d'exécuter, et d'examiner le contenu des registres de coordonnées du terminal, en n'oubliant pas que si l'élément pointé était un vecteur, on y trouve les coordonnées de l'extrémité et non pas celles du point désigné[4] . La technologie s'arrête généralement là, et c'est la programmation qui doit alors découvrir le lien entre ces informations et la représentation structurée des objets graphiques, de façon à déterminer ce que l'on a montré. Même les représentations les plus simples, telles que les tables de corrélation, suffisent en général à trouver de quel objet il s'agit, et de quelle figure fait partie cet objet, mais on arrive alors à un problème presque philosophique. Si je montre par exemple le côté inférieur du triangle de la figure SCHEMA, est-ce que j'entends désigner ce vecteur précis (élément), ou le triangle entier (objet), ou le schéma triangle-nuage-texte (figure), ou meme l'ensemble de ce qui apparaît sur l'écran (image)? Tant que l'ordinateur ne saura pas lire dans la pensée de l'utilisateur, la réponse à cette question ne pourra être qu'explicite, c'est-à-dire que l'on doit avant de désigner quelque chose savoir pourquoi on le fait et le dire à l'ordinateur par un moyen quelconque.

Ceci nous conduit aux derniers outils de dialogue, que l'on pourrait classer dans les outils divers. Ils servent simplement à éviter la frappe et l'envoi d'un message explicite quand celui-ci ne doit renfermer aucune information variable, et qu'il se réduit donc au nom d'une commande. On remplace alors le nom par un simple numéro pour sa transmission à l'ordinateur, la traduction numéro -> nom de commande étant établie par programme au moyen d'une simple table, et la traduction commande -> numéro étant faite par l'écriture du nom de la commande à côté du dispositif qui envoie l'information numérotée. Le premier de ces outils est purement technologique, c'est le clavier de fonctions: il s'agit d'un simple clavier d'un certain nombre de touches muettes, à côté desquelles on peut écrire ce que l'on désire; quand on appuie sur l'une des

---

[3]Il a cependant l'inconvénient de ne pouvoir servir qu'a cet usage, c'est pourquoi on utilise quelquefois à sa place un système à tablette muni d'un comparateur.
[4]Les dispositifs à comparateur ne présentent pas ce dernier inconvénient.

touches, on envoie ainsi à l'ordinateur le numéro correspondant; quoique le nom de ce dispositif indique qu'il est souvent utilisé pour indiquer des fonctions à exécuter, ceci n'a rien de fondamental, et on peut très bien par exemple associer des touches aux diverses réponses possibles à une question. L'avantage primordial de ce dispositif est sa simplicité de conception et d'utilisation, qui le rend pratique et peu coûteux. Un deuxième outil permet de faire le même travail et un peu plus, mais il est alors entièrement programme, c'est la technique du menu: les noms des commandes apparaissent sur l'écran comme sur la carte d'un restaurateur, et on désigne le nom choisi au moyen du pointeur optique; cette technique réclame un travail de programmation relativement important, quoi qu'elle soit fonctionnellement identique à la precédente; l'avantage qu'elle fournit est que la liste des commandes n'est pas statique, c'est-à-dire qu'elle peut évoluer en fonction du temps pour n'indiquer que les commandes valides à un certain moment, et qu'elle peut être de longueur indéfinie, une des commandes servant alors à demander l'affichage de la page suivante du menu si l'on ne trouve pas son bonheur dans la page courante.

2.32 - UTILISATION DES OUTILS DE DIALOGUE.

L'utilisation des outils de dialogue dont je viens de parler se caractérise par deux aspects contradictoires: technologiquement, le dialogue se fait par l'intermédiaire d'interruptions, c'est-à-dire d'évènements non synchronisés avec le déroulement du programme; logiquement, le dialogue se fait par questions et réponses dans un sens ou dans l'autre, c'est-à-dire que le programme ne peut comprendre un message que s'il s'attendait à le recevoir. En fait, le problème vient d'un point capital que l'on oublie trop souvent: l'ordinateur est beaucoup plus stupide que l'être humain, ou plus exactement le programme ne renferme que la quantité d'intelligence que le programmeur y a mise, et rien de plus; par conséquent, à toute question qui n'entre pas très précisement dans le cadre étroit de ce qui a été prévu dans le programme, celui-ci ne peut que répondre: "- Je ne comprends pas".

Nous avons donc à concilier l'aspect technologique asynchrone du dialogue avec son aspect logique synchrone. Ce sera le rôle d'un certain nombre de méthodes de traitement des interruptions, qui reviennent toutes à en différer la prise en compte jusqu'au moment où le programme est capable de le faire. Un autre point important est que les sources d'interruptions sont nombreuses, et qu'il est rarissime qu'a tout moment elles aient toutes une signification pour le programme: il faut donc pouvoir ignorer complètement certaines

de ces sources, et cela de façon évidemment dynamique, puisque la nature des messages reconnaissables dépend d'un contexte évolutif.

Le traitement des interruptions technologiques est donc fait par un sous-programme sous-jacent, qui n'influe sur le déroulement du programme normal que par le jeu d'indicateurs qu'il modifie et que l'on peut consulter. On peut à tout moment indiquer à cet esclave chargé des basses besognes quelles sont les sources d'interruptions que l'on désire traiter et celles que l'on veut ignorer. Le programme est alors libre de ses mouvements, il peut quand il le désire consulter les indicateurs de son choix pour voir si une demande y est en attente, examiner les informations associées à une interruption qui s'est produite précédemment, attendre qu'une ou plusieurs interruptions précises se produisent.

L'outil programmé qui permet tous ces traitements est le <u>niveau</u> (d'interruptions). Il s'agit en quelque sorte d'un ensemble de files d'attente, chaque file étant associée à une source d'interruptions précise. Certaines de ces files peuvent être fermées vers leur entrée, elles correspondent alors à des sources d'interruptions que l'on veut ignorer; les autres sont ouvertes, chaque interruption nouvelle est ajoutée a l'entrée. On consulte le niveau en examinant la sortie des files et en en extrayant les informatons trouvées. Le nom même du niveau provient de ce que l'on peut en traiter plusieurs à la fois, en les empilant les uns au-dessus des autres: les nouvelles interruptions ajoutent des informations dans les files d'attente du niveau placé au sommet de la pile, mais on peut examiner tous les autres, retirer des niveaux du sommet de la pile ou du milieu, intervertir leur ordre, etc.[5]

Cet outil reste cependant assez élémentaire, en ce sens que l'on continue de parler de sources d'interruptions, et que l'on ne sait pas par exemple considérer l'ensemble des évènements liés aux outils de dialogue comme des symboles qui peuvent servir à composer des messages. Des études sont faites en ce sens [Pa68], mais on ne sait pas très bien où l'on va car l'expérience manque en ce domaine, et l'on reste esclave de la manière acquise depuis des siècles d'écrire de l'information au moyen de chaînes séquentielles de symboles de même nature: s'il est impratique dans la conversation courante d'écrire certains mots ou de dessiner certaines choses difficiles à décrire, il est impossible dans un texte écrit de remplacer la mauvaise description d'un son ou d'une odeur par le son ou l'odeur

---

[5]Le rôle et l'utilisation du niveau seront mieux précisés dans l'exemple particulier du langage Euphémie (voir le paragraphe 4.17).

eux-mêmes; c'est pourtant à peu près ce à quoi il faudrait aboutir dans le dialogue avec l'ordinateur par l'intermédiaire d'un terminal graphique.

2.4 - CLASSIFICATION DES LANGAGES.

Le travail que l'on effectue sur un terminal graphique connecté à un ordinateur nécessite la connaissance et l'utilisation d'un certain nombre de langages bien différents.

Considérons d'abord un utilisateur qui construit un programme conversationnel[6] graphique; son travail peut être décomposé en deux phases successives. La première phase est l'écriture du programme qui devra être exécuté par l'ordinateur; ce programme comprendra des instructions classiques logiques ou arithmétiques, mais aussi des instructions plus spécifiquement graphiques, pour la définition ou la génération des objets à manipuler, leur transmission vers le terminal et leur utilisation, et des instructions de communication pour lire et interpréter les demandes et réponses envoyées par l'opérateur du terminal au moyen des outils technologiques de dialogue. Pour composer ce programme, l'utilisateur se sert d'un langage de programmation, qui peut être le langage d'assemblage de l'ordinateur, mais qui peut aussi être d'un niveau plus élevé; ce peut être par exemple Algol, Fortran ou PL/I, avec des sous-programmes effectuant les traitements graphiques, ou les mêmes langages avec de nouvelles instructions spécialisées, ou, ce qui est probablement le mieux, un nouveau langage spécialisé dans les traitements graphiques.

Une fois le programme correctement composé, compilé et chargé dans l'ordinateur, il commence à s'exécuter, et doit alors pouvoir dialoguer avec l'utilisateur placé devant son terminal; ce dialogue va permettre d'envoyer des données pour modifier certains paramètres de l'image affichée, ou des commandes pour modifier l'image, l'effacer, la conserver, la dupliquer, la déplacer, pour passer à la phase suivante du travail, pour calculer certains résultats à partir des images réalisées, pour terminer tout le travail. Cette étape nécessite un langage de communication qui est d'une nature assez spéciale puisque il utilise tous les outils technologiques de dialogue fournis par le terminal; la forme même de ce langage dépend uniquement de la façon dont a été conçu le programme, de ce à quoi

---

[6] La nature et l'utilité du travail conversationnel seront précisées au chapitre 5.

il s'attend et de ce qu'il est capable de reconnaître. Une autre remarque importante à faire est que l'individu qui utilise ce langage de communication n'est pas forcément celui qui a écrit le programme; en fait, le programme avec lequel on dialogue peut constituer un interlocuteur dont on n'a absolument pas besoin de connaître les principes de fonctionnement, et dont il suffit de savoir le mode d'emploi.

Si nous considérons maintenant le travail du programmeur quand il compose, essaie et met au point son programme, nous constatons que les deux langages précédents ne sont pas suffisants. Il va falloir forcément travailler dans le cadre d'un certain système d'exploitation, qui fournit les compilateurs des langages utilisés, les outils programmés élémentaires des opérations d'entrée et sortie, et les outils d'analyse et mise au point des programmes. Ce système d'exploitation reconnaît un certain langage de commande, avec lequel on lui indique ce que l'on désire faire. Etant donné que le terminal est un dispositif éminemment conversationnel, il semble normal de pouvoir composer et mettre au point les programmes qui l'utilisent de façon également conversationnelle; il faut donc disposer d'un système qui permette à l'utilisateur de s'asseoir devant son terminal sans autre outil que lui-même, et de n'en repartir qu'après avoir composé et mis au point son programme ou au moins une partie.

La distinction entre ces trois types de langages me semble d'autant plus importante qu'elle est rarement bien claire. Il y a en particulier très souvent confusion entre le langage de commande et le langage de communication, ou entre celui-ci et le langage de programmation. La caractéristique la plus évidente d'un langage de programmation, dans le contexte qui nous occupe, est qu'il n'est jamais conversationnel (même s'il permet de rédiger des programmes qui le sont): si l'écriture d'un programme est chose difficile, c'est justement parce que, ne pouvant faire appel à l'être humain lors de chaque décision à prendre, on est obligé de prévoir toutes les possibilités. Une des instructions fondamentales des langages de programmation qui ne peuvent exister dans les langages conversationnels est l'instruction "aller à". Les deux autres langages que j'ai distingués sont tous deux conversationnels, mais il est facile de les reconnaître: quand on parle au système, on utilise le langage de commande; quand on parle à son programme, on utilise son langage de communication.

3.1 - NIVEAU DES LANGAGES ET PUISSANCE DES OUTILS.

3.11 - PRINCIPES D'ORGANISATION.

La distinction que je viens de faire entre les trois sortes de langages que l'on utilise dans les traitements graphiques est souvent peu nette, à tel point que si l'on regarde d'un peu près les travaux effectués dans ce domaine, on s'aperçoit qu'ils sont presque tous à cheval sur deux ou trois categories, et fournissent des langages plus ou moins bâtards que l'on ne sait comment baptiser. Cette situation me semble mauvaise, et je ne vais m'intéresser dans ce chapitre qu'au coté "programmation" des langages.

Même dans ce domaine réduit, on retrouve le gigantesque désordre bien connu dans l'ensemble des langages de programmation, et qui a été exprimé par l'image suggestive d'une Tour de Babel sur la couverture d'un numéro de 1962 des Communications de l'A.C.M., et plus récemment sur celle d'un livre de J. Sammet consacré entièrement à ce sujet. Ce dernier ouvrage [Sa69] tente de mettre un peu d'ordre au moyen de certaines classifications, et l'on constate que, si les langages se prêtent relativement bien à un découpage en catégories par rapport aux domaines d'application (calculs numériques, gestion, commande des machines-outils, traitements symboliques, simulation, etc.), ce travail est à peu près impossible lorsque l'on tente d'introduire, dans ces catégories ou dans l'ensemble général, des hiérarchies par rapport à certains critères.

C'est ainsi qu'après avoir personnellement essayé, dans ma thèse de troisième cycle [Le65-1], de classer les langages par rapport aux trois critères du niveau, de la richesse et des restrictions, j'étais arrivé à trois courbes qui auraient dû normalement être la projection sur trois plans orthogonaux de la même courbe de l'espace, et où malheureusement deux de ces courbes étaient rectilignes, mais non pas la troisième... Cette difficulté du classement hiérarchisé conduit à l'alternative suivante: si un ensemble ne peut être classé suivant une certaine relation d'ordre, il faut incriminersoit l'ensemble, soit la relation (je rejette la troisième voie qui consisterait à incriminer à la fois l'ensemble et la relation, et qui supposerait que la personne qui fait l'essai de classement s'est trompée d'objet à étudier). Etant donné qu'un

classement hiérarchisé est foncièrement logique, et que l'ensemble des langages de programmation est une construction humaine et non pas un système naturel et pré-existant, c'est cet ensemble qu'il faut incriminer, et en proposant un remède à cette situation fondamentalement mauvaise, au moyen de principes d'organisation.

Je ne m'intéresserai ici, dans le domaine de la programmation graphique, qu'à deux critères de classification. Le premier est celui du niveau du langage: pour préciser ce terme vague, je dirai qu'un langage est de plus haut niveau qu'un autre s'il permet d'exprimer les mêmes notions d'une façon plus simple, plus naturelle et plus rigoureuse; ainsi Fortran est-il de niveau plus élevé que la plupart des langages d'assemblage, Algol et PL/I sont de niveau à peu près semblables, avec une légère préférence pour le premier. Le deuxième critère de classification est celui de la puissance des outils fournis par les langages: par exemple, l'outil "boucle pour" d'Algol est plus puissant que l'outil correspondant de Fortran, car il permet de faire plus de travail pour le même effort de programmation; un sous-programe de lecture ou d'écriture qui gère lui-même les mémoires-tampons, les problèmes de blocage et de déblocage des enregistrements et ceux de simultanéité, est plus puissant qu'un sous-programme qui ne le fait pas.

Le premier principe d'organisation que je propose est le suivant:

> Le niveau d'un langage et la puissance des outils qu'il fournit sont ou du moins doivent être en étroite relation.

On ne doit pas définir d'outils trop puissants relativement au niveau du langage, ni de langage de trop haut niveau relativement aux outils qu'il fournit. Cela ne signifie évidemment pas qu'un langage de haut niveau ne doive fournir que des outils puissants, et omettre les opérations d'addition et de soustraction; en revanche, cela signifie par exemple qu'un outil de lecture de données du style de l'instruction de lecture dirigée par les données du langage PL/I (instruction "GET DATA") est aussi inutile et néfaste dans un langage d'assemblage qu'il fait cruellement défaut dans Algol 60.

Le deuxième principe d'organisation est le suivant:

> Dans la définition des moyens qui doivent permettre l'utilisation d'un dispositif donné (et en particulier d'un terminal graphique), on doit commencer par définir les outils les plus élémentaires, puis les exprimer au moyen d'un langage également élémentaire, avant de passer aux outils puis au langage du niveau immédiatement supérieur.

Les nouveaux outils sont plus puissants que les précédents, le nouveau langage est d'un niveau plus élevé que le précédent, même

s'il l'inclut. On continue ainsi suivant un chemin en zig-zag, des outils au langage et du langage aux outils, autant qu'il est nécessaire. Chaque étage d'outils et de langage s'appuie sur le précédent, comme dans une maison bien assise, et il faut passer par le rez-de-chaussée et les deux premiers étages pour arriver au troisième et au grenier. Toute combinaison langage-outils qui ne trouve pas sa place dans ce schéma est alors considérée comme un hybride mal conçu, un mutant boîteux qui ne devrait pas exister.

Je vais à présent développer ce schéma dans un exemple complet mais hypothétique, en distinguant quatre etages de définition des outils et du langage, qui correspondront à quatre puissances des outils, c'est-à-dire successivement aux éléments, aux objets, aux figures et aux structures.

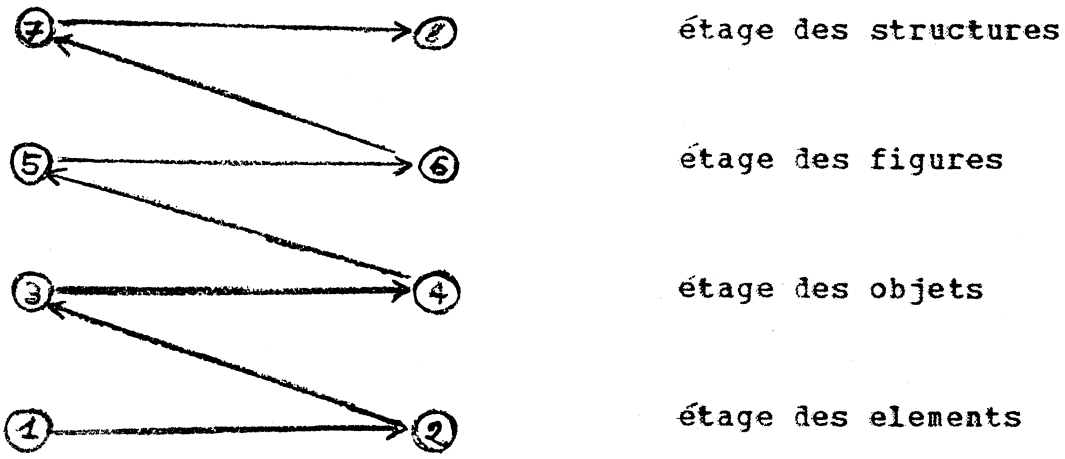


Figure 3.1 - Les quatre étages de langages.

3.12 - ETAGE DES ELEMENTS.

Je désire utiliser un terminal graphique à écran cathodique, qui dispose des caractéristiques suivantes: balayage cavalier, mémoire d'entretien, générateur de vecteurs, générateur de caractères, possibilité de coordonnées absolues et relatives. La première chose à faire est de définir plus précisément quels doivent être les

outils élémentaires que fournira ce terminal; par mesure de simplification, j'omettrai les problèmes liés à l'introduction de texte dans la mémoire d'entretien au moyen du clavier de machine à écrire.

Langage	Outil	Catégorie
AFPA	affichage de points en mode absolu	
AFPR	affichage de points en mode relatif	
AFVA	affichage de vecteurs en mode absolu	
AFVR	affichage de vecteurs em mode relatif	
AFCP	affichage de caractères de petite taille	
AFCM	affichage de caractères de taille moyenne	
AFCG	affichage de caractères de grande taille	génération
TR adresse	transfert	
TRCA adresse	transfert avec conservation d'adresse	
GI	génération d'interruption	
EM adresse, longueur	écriture dans la mémoire d'entretien	
LR adresse	lancement de la régénération	
AR	arrêt de la régénération	
LXY	lecture des registres de coordonnées	transmission
LI	lecture des indicateurs	
LM adresse, longueur	lecture de la mémoire d'entretien	
IPO adresse	interruption du pointeur optique	
ICF numéro	interruption du clavier de fonctions	
ICA	interruption du clavier alphanumérique	communication
IME adresse	interruption en mémoire d'entretien	

Tableau I - Etage des éléments (langage technologique).

En ce qui concerne la génération, la mémoire d'entretien doit contenir une liste d'affichage qui est en quelque sorte un programme exécutable par le terminal lui-même, formé d'ordres et de données: ordres d'affichage de points, de vecteurs et de caractères (en trois tailles différentes), ordres de transfert inconditionnel, de transfert avec conservation d'adresse (pour permettre l'appel de sous-programmes), de génération d'interruption; coordonnées absolues et relatives, avec possibilité d'indiquer l'extinction du faisceau

lumineux; caractères alphanumériques, y compris un caractère de passage à la ligne et un caractère ineffectif.

Les problèmes de transmission sont dévolus à un canal et à une unité de contrôle, qui doivent être capables d'effectuer les opérations suivantes: écriture dans la mémoire d'entretien de N mots à partir de l'adresse A, lancement de la régénération à partir de l'adresse A, lecture des registres de coordonnées en x et y, lecture des indicateurs nécessaires, lecture dans la mémoire d'entretien de N mots à partir de l'adresse A.

Enfin, les problèmes de communication sont résolus au moyen d'un certain nombre d'interruptions que le terminal peut envoyer à l'ordinateur. Les causes d'interruptions sont évidemment fonction des dispositifs prévus: ce sont donc le pointeur optique, qui envoie une interruption quand il détecte de la lumière, le clavier de fonctions, qui fait de même quand on enfonce une de ses touches, l'ordre de génération d'interruption de la mémoire d'entretien, qui fonctionne quand le cycle de régénération passe dessus, et la touche du clavier alphanumerique qui sert à indiquer la fin d'une ligne.

Ces trois ensembles d'outils me permettent de définir un langage, de niveau technologique, qui n'est encore qu'un moyen d'exprimer clairement les outils, et n'est pas plus un langage de programmation que ne l'est le langage propre à un ordinateur. Il ne contient donc qu'un certain nombre de codes mnémoniques, exposés dans le tableau I.

3.13 - ETAGE DES OBJETS.

Il faut à présent définir les outils nécessaires pour une programmation réelle du terminal. Les principales adjonctions par rapport à l'étage précédent sont les suivantes: en génération, de nouveaux types d'objets (position, ligne et segment), et des sous-programmes de fabrication des ordres d'affichage d'objets non simples (arcs de cercles ou d'ellipses, grilles cartésiennes ou polaires, linéaires ou logarithmiques, etc.); en transmission, des ordres de lecture et d'écriture faisant le travail voulu par le jeu de divers paramètres, et ne nécessitant pas la connaissance du fonctionnement du canal ou de l'unité de contrôle; en communication, l'utilisation de masques d'interruptions, et des procédés de traitement fondés sur la consultation périodique d'indicateurs, l'attente d'évènements, ou la définition de sous-programmes asynchrones.

Langage	Outil	Catégorie
tous les codes mnémoniques de génération avec un G en tête: GAFPA GTR DEBUT GGIGAFAC p,q,.. GAFGR P,Q,..	construction en mémoire centrale des ordres de génération génération des ordres d'affichage d'un arc de cercle défini par p, q, ... génération des ordres d'affichage d'une grille	génération
GECR ZONE, ADRESSE GLIR ZONE, ADRESSE GLIR ZONE,'XY' GLIR ZONE,'I' GECR 'LR', ADRESSE GECR 'AR'	appel des sous-programmes gérant eux-mêmes les problèmes de synchronisation et de traitement des erreurs	transmission
GMASQ ICO,ICA GTRAI ICF,SP GATEN IME	ne pas traiter les interruptions PO, CA appeler SP en cas d'interruption CF attendre l'interruption ME	communication

Tableau II - Etage des objets (langage d'assemblage).

Le langage de programmation doit s'insérer dans le langage d'assemblage existant sur l'ordinateur (par exemple par le biais de macros-instructions), et en utiliser toutes les possibilités, en particulier l'adressage symbolique et les compteurs d'adresses pour le problème de la généraion des ordres de la liste d'affichage, et les macros-instructions pour l'écriture des ordres de transmission ou de communication. On peut résumer ceci dans le tableau II.

3.14 - ETAGE DES FIGURES.

L'étage suivant doit fournir des outils plus commodes, et qui ne nécessitent plus la connaissance détaillée du fonctionnement du terminal graphique; il faut par exemple pouvoir dire que l'on fait disparaître de l'écran (que l'on omet) un objet ou un ensemble d'objets (figure ou séquence), plutôt que de devoir écrire en mémoire d'entretien un ordre de transfert qui saute les ordres de génération concernés. Il est utile de pouvoir utiliser un système de coordonnées plus pratiques et plus significatives que celles qui sont définies par le terminal lui-même (par exemple [0,0] en bas à gauche et [1023,1023] en haut à droite), de pouvoir donner des noms aux objets et de les manipuler au moyen de ces noms.

Langage	Outil	Catégorie
figure f; segment S,t; S = 0,0,4,10;	déclarations de variables graphiques génération d'un objet	génération
afficher f; omettre S; inclure POINT; annuler Fig2;	traitement des figures et des objets de façon symbolique, sans plus avoir besoin de longueurs, adresses, etc.	transmission
traiter t1,t2,po; attendre t4,TUT; i=état t2,po;	manipulation des interruptions de façon symbolique, sans sous-programmes ni traitements asynchrones	communication

Tableau III - Etage des figures (langage algorithmique).

Les nouveaux outils seront donc principalement, pour la génération l'introduction des notions de figure et de séquence avec tout ce qui s'y rattache; pour la transmission le remplacement des notions de lecture et d'écriture par celles d'affichage, inclusion, omission et annulation d'objets, de séquences ou de figures; pour la communication, l'introduction de moyens simples d'autoriser certaines interruptions, de voir si elles se sont produites et de savoir dans quelles conditions, au moyen de l'outil nouveau qu'est le niveau d'interruptions.

Le langage de programmation qui comporte ces outils doit se trouver d'un niveau semblable à Fortran ou Algol; si c'est l'un de ces langages, cela signifie ou bien qu'on a ajouté au langage hôte de nouveaux types de variables et de nouvelles instructions, ou bien que tout est fait par le moyen de procédures et sous-programmes. La première solution est difficile et la greffe que l'on fait risque de donner naissance à un monstre, et la deuxième solution conduit à un langage très pénible d'aspect et d'emploi, comme on le verra par la suite (paragraphe 4.21). Il est donc presque obligatoire de construire un nouveau langage spécalisé dans les traitements graphiques, qui complète les outils que l'on vient de voir par l'utilisation étendue de la notion de variable graphique et des notations symboliques. Le tableau III résume la situation, sur laquelle je reviendrai longuement au chapitre 4.

3.15 - ETAGE DES STRUCTURES.

Le dernier étage de la pyramide est beaucoup plus difficile à préciser que les précédents, étant donné qu'à l'heure actuelle il n'existe pas à ma connaissance d'ensemble vraiment cohérent d'outils de ce niveau, ni à plus forte raison de langage de programmation associé. En fait, si tout le monde s'entend pour dire "qu'il faut des structures" pour faire des traitements graphiques, peu de personnes sont capables d'expliciter ce qu'elles entendent par là, et en tout cas personne n'est d'accord sur ce qu'il faut exactement. Il apparaît que l'on peut peut-être s'entendre sur ce que l'on veut faire, et encore pas toujours, mais absolument pas sur la façon de le faire. Il est donc probable qu'il vaut mieux, dans la définition des outils de ce dernier étage, ne pas fournir à l'utilisateur des "structures" toutes faites, mais seulement les outils qui permettent de les définir et de les utiliser. De même qu'un langage de programmation tel qu'Algol ou Fortran ne fournit pas d'instruction d'inversion de matrices, mais seulement les instructions qui permettent de le faire, il ne faut pas donner comme outils des algorithmes, mais il faut connaître ces algorithmes pour donner les moyens de les réaliser.

Pour préciser un peu ceci, il est improbable qu'il soit utile de fournir à l'utilisateur dans un langage de programmation graphique un outil qui permette la rotation d'un angle quelconque autour d'un axe quelconque d'une figure à trois dimensions; cependant, il faudra lui donner les moyens de définir des figures à trois dimensions et de les afficher sur un écran qui n'en a que deux. De même, il faudra que l'utilisateur puisse exprimer facilement des contraintes géométriques ou topologiques telles qu'égalité ou colinéarité, ou des propriétés des objets telles que charge électrique ou résistance à la flexion, mais il faut lui laisser le soin de s'occuper de ces contraintes ou de ces propriétés pour les ignorer ou les prendre en considération suivant ses besoins.

Les trois catégories d'outils que j'ai distinguées précédemment (paragraphe 2.21) restent valables, mais il vaut mieux parler maintenant de définition et d'utilisation des objets, figures et structures, plutôt que de génération et de transmission. Les outils de définition permettent de construire les objets graphiques par l'intermédiaire de représentations structurées; les outils d'utilisation permettent d'une part les opérations d'affichage, inclusion, ommission et annulation (qui sont en fait des opérations de traduction de la représentation structurée en ordres de génération de la liste d'affichage), d'autre part les opérations qui modifient

les propriétés des objets graphiques (déplacements ou déformations en respectant certaines contraintes).

Le langage de programmation associé est forcément un langage très spécialisé et de très haut niveau s'il doit rester utilisable et efficace: en effet, le placage d'outils puissants sur un langage existant et qui n'a pas été prévu pour cela ne peut conduire qu'à des résultats désastreux. Il est impossible de définir maintenant à quoi doit ressembler un tel langage, ni même de dire s'il doit pouvoir être accessible aux programmeurs normaux, mais il est certain que sa structure et sa syntaxe doivent être aussi simples que rigoureuses si l'on veut que l'utilisateur puisse comprendre quelque chose à ce qu'il fait. Malgré les études en cours dans ce domaine, le sujet est pour le moment trop futuriste pour que je puisse poursuivre sans risquer de faire de la science-fiction.

3.2 - ETUDE DES LANGAGES EXISTANTS.

Les langages de programmation graphique qui existent actuellement peuvent être répartis grossièrement en trois catégories, indépendamment des niveaux auxquels ils se placent. Les langages de programmation simple s'utilisent comme les langages de programmation habituels, et sont destinés à tous ceux qui veulent s'en servir. Les langages de construction de systèmes graphiques ne sont pas d'usage universel, mais servent simplement à quelques programmeurs, pour les aider dans la fabrication d'un système de dessin dont l'utilisateur n'aura pas à connaître le mode de fonctionnement; ces langages sont les seuls actuellement qui fournissent des outils non élémentaires de représentation structurée. Enfin, les langages purement graphiques vont encore plus loin en ce sens que ce sont beaucoup plus des langages de commande que des langages de programmation; la représentation structurée fait alors partie du système qui se trouve sous le langage, et l'utilisateur n'a pas même à savoir qu'elle existe.

3.21 - LES LANGAGES DE PROGRAMMATION SIMPLE.

3.211 - LE LANGAGE LAGROL: [LeLu68, Lu68-1,Lu68-2] se place à l'étage des objets; il présente la particularité de se placer très nettement au-dessus du langage technologique correspondant, sur le plan des outils qu'il fournit. En effet, le terminal graphique auquel il est associé est très rudimentaire puisque il n'est capable d'afficher que des points et ne dispose pas d'une mémoire d'entretien. L'étage des objets comprend donc la définition de nouveaux

outils: sous forme d'objets, à savoir les vecteurs, les chaînes de caractères, les déplacements et les arcs de cercle; sous forme de données, à savoir les coordonnées absolues et relatives; sous forme d'instructions, à savoir le branchement simple, l'appel de sous-programme graphique, la modification des bases des coordonnées relatives; sous forme d'options, à savoir huit tailles de caractères, huit densités d'affichage des lignes, huit intensités d'affichage.

Catégorie	Outils	Observations
Génération	Objets: Points Vecteurs Déplacements Textes Arcs de cercle	Coordonnées absolues, ou relatives à deux registres, pour tous les objets sauf les déplacements, qui sont purement relatifs.
Transmission	Lancement de l'affichage par appel de l'interprète Arrêt Appel de sous-programme en langage de la machine.	Les modifications de la liste d'affichage se font en langage de la machine, sans qu'il y ait d'outil précis associé
Communication	Mode avec ou sans interruptions Traitement ou non du pointeur optique Sous-programmes asynchrones	Il n'y a pas de clavier de fonctions. La liaison entre la frappe d'un caractère et son apparition sur l'écran ou sur le télétype doit être entièrement programmée.

Tableau IV - Le langage Lagrol.

Le langage de programmation associé, qui est le langage Lagrol, est un langage interprété dont la notation symbolique est directement acceptable par l'assembleur de l'ordinateur D.E.C. PDP-8 qui gouverne le terminal. Ce principe d'interprétation rend très facile la simulation d'outils qui n'existent pas sur le terminal réel, et montre très clairement le passage de l'étage des élements à l'étage des objets.

En revanche, les outils de transmission apparaissent moins clairement, à cause de l'absence de mémoire d'entretien, et la

génération n'est que rarement faite par le programme lui-même, étant donné le peu de possibilités que l'on a dans le langage de la machine. La génération est donc faite par le programmeur, de façon statique, au moment ou il écrit son programme; la transmission se fait par appel de l'interprète du langage Lagrol, lequel peut sur certaines instructions spéciales rendre le contrôle au programme, ce qui permet toutes les modifications possibles à la liste d'affichage, de façon directe.

Enfin, les outils de communication sont d'une utilisation délicate, principalement à cause de la simplicité spartiate de l'ordinateur. Leurs possibilités se limitent en gros à deux modes, l'un où les interruptions sont ignorées et où l'on doit continuellement examiner des indicateurs, l'autre où les interruptions sont permises et provoquent l'appel suivant le cas d'un sous-programme parmi trois: pointeur optique, caractère frappé sur le clavier alphanumérique, caractère imprimé (le clavier alphanumérique est celui d'un télétype connecté à l'ordinateur et sans rapport avec le terminal graphique).

3.212 - LE LANGAGE GRAPHIQUE GENERAL PROPOSE PAR H E KULSRUD: [Ku 68] est défini dans le cadre d'un système compilateur de compilateur qui doit permettre de produire des langages graphiques pour de nombreuses configurations différentes d'ordinateur et de terminal graphique associés. En fait, la caractéristique majeure de ce langage est qu'il n'est pas spécifiquement prévu pour un terminal graphique utilisé de facon conversationnelle, et qu'en fait il ne contient aucun outil de communication. En revanche, il fournit des outils que l'on ne s'attend guère à voir dans un langage de programmation, c'est-à-dire des outils d'analyse des images du style de "Ces deux segments se coupent-ils?" ou bien "Cet objet fait-il partie de cette figure?". L'auteur ne paraît pas en réalité très fixé sur ce à quoi doit servir son langage, et le fait qu'il en envisage une version non écrite est significatif: ce langage est à la fois de programmation et de communication, ou plutôt il hésite entre les deux.

Le côté purement graphique du langage de H. E. Kulsrud fait nettement partie de l'étage des objets: on dispose de points, de vecteurs et d'arcs de cercles, définis de façon absolue par rapport à un système de coordonnées que l'on peut redéfinir entre deux générations. Un autre objet est plus complexe et plus inhabituel, c'est le graphe défini par deux fonctions données sous la forme: "x=f(y); y=g(x)", l'une des deux fonctions pouvant (devant ?) être omise. Les outils de transmission comprennent une instruction d'affichage qui s'appelle curieusment "imprimer", des instructions

Catégorie	Outils	Observations
Génération	Objets: Points Lignes (vecteurs), Arcs (de cercle) Courbes Echelle Appels de procédures	Seul le premier d'une suite de points ou de lignes peut être nommé. Les procédures ont des paramètres.
Transmission	Affichage Conservation de l'image Récupération de l'image Copie de sous-éléments Rotation de sous-éléments Effacement de sous-éléments	Un sous-élément est analogue à une séquence
Communication	Néant	

Tableau V - Le langage de Kulsrud.

de définition, conservation et changement d'image visiblement faites pour un traceur sur microfilm, et une instruction d'effacement. Enfin, il est possible de définir des procédures graphiques qui constituent en quelque sorte de nouveaux objets, mais il n'y a de toutes façons pas de hiérarchisation ni même de structuration possible des éléments de l'image affichée.

Le langage associé à ces outils est d'une forme assez hybride, qui hésite entre un langage d'assemblage et un langage comme Fortran ou plus précisement Mad. Certaines instructions s'écrivent sous une forme qui rappelle les langages évolués, alors que d'autres sont extrêmement pauvres, et l'ensemble du langage reste d'une nature hésitante et mal définie: il lui manque une syntaxe logique, de véritables déclarations et la notion de figure pour être au troisième étage de la pyramide des langages, alors que sa syntaxe est trop compliquée, et que certaines instructions sont sans intérêt s'il faut le mettre au deuxième étage.

3.213 - LE LANGAGE LAPREC: (HaBo68, Ha69, Ha70) est beaucoup plus facile à classer que le précédent: c'est un langage de programmation simple de l'étage des figures, conçu pour l'utilisation d'un terminal graphique classique, qui ne dispose cependant pas d'une mémoire d'entretien. Du point de vue des outils de génération fournis, la particularité de Laprec est que les figures peuvent se contenir les unes les autres, introduisant ainsi une hiérarchisation continue. Les outils de transmission sont constitués de l'unique instruction d'affichage, les figures devant être recalculées si l'on désire les modifier; cependant, ces modifications peuvent se faire très facilement quand il s'agit de transformations analytiques quelconques, c'est-à-dire aussi bien de déplacements que de déformations. Les outils de communication n'utilisent pas explicitement le niveau d'interruptions, mais un système par tables d'étiquettes et instructions d'attente qui rappelle le systeme GPAK [IBM1], à cette différence près qu'il n'y a pas en cas d'interruption appel d'un sous-programme, mais simple transfert à une étiquette indiquée à l'avance dans une instruction spéciale (ce transfert ne peut se faire qu'après exécution de l'exécution d'attente, et non pas au moment où l'interruption se produit, car on ne prévoit pas de traitements asynchrones).

Le langage de programmation associé à ces outils est une extension du langage Fortran, qui utilise pour certaines instructions purement graphiques le formalisme de l'instruction d'affectation. Il s'agit d'une solution assez hybride, choisie pour ne pas choquer les utilisateurs habitués depuis toujours à travailler dans le même langage. Elle a le grave inconvénient de conduire à un langage syntaxiquement boîteux et difficilement extensible au-delà d'une certaine limite; d'autre part, Fortran n'est peut-être pas le type de langage qu'il faut choisir comme outil général à notre époque.

Les principales extensions faites à Fortran pour conduire à Laprec sont: l'adjonction d'un nouveau type de variable, la figure (il n'y a pas d'objet explicitement déclaré, on confond un objet avec la figure qui ne contient que lui); l'extension de la signification de l'instruction d'affectation, où le signe "+" sert à indiquer l'union d'objets ou de figures pour composer une nouvelle figure; quelques instructions supplémentaires: affichage, effacement, attente et spécification du lieu où l'on doit se transférer en cas d'interruption d'un certain type. Le reste du travail, en particulier la génération effective des objets, se fait par l'appel de fonctions ou de procédures.

des objets qui permet les déformations et les déplacements. Les outils de communication sont constitués par le niveau d'interruptions, que l'on peut modifier, consulter et effacer, et par quelques instructions destinées à l'utilisation du clavier alphanumérique.

Categorie	Outils	Observations
Génération	Objets: points position ligne segment texte Séquences Figures Instruction universelle de génération	Les objets peuvent être nommés, numérotés, ou non désignés. Un objet fait toujours partie d'une figure. Les coordonnées peuvent être absolues ou relatives.
Transmission	Affichage Omission (extinction) Inclusion (rallumage) Annulation (effacement) Instruction universelle de mise à jour	Omission, inclusion et annulation portent sur les objets, séquences et figures. La mise à jour d'un objet permet des déformations et des déplacements sans recalcul des figures.
Communication	Déclaration de niveau Traiter des interruptions Ignorer des interruptions Consulter le niveau Lire la mémoire d'entretien Manipuler le curseur	La consultation de niveau se fait avec ou sans attente. Une détection au pointeur optique fournit le nom, le numéro et les coordonnées de l'objet montré, ainsi que le nom de la figure et de la séquence dont il fait partie.

Tableau VII - Le langage Euphémie.

Le langage associé à ces outils est un langage nouveau et spécialisé, qui comprend des déclarations et des instructions. Comme on le verra par la suite, ce langage est prévu pour être compilé de façon conversationnelle et incrémentielle, ce qui lui donne quelques particularités inhabituelles dans un langage de ce

niveau: en particulier, les déclarations sont entièrement statiques, c'est-à-dire qu'il n'y a pas de structure de bloc, et toute instruction syntaxiquement correcte le reste quelles que soient les instructions qui l'entourent, y compris les instructions d'itération et de condition. Les instructions d'usage graphique sont nombreuses et reconnues par des mots-clés, mais les autres instructions sont actuellement beaucoup plus pauvres, ce qui est une carence grave. L'aspect général du langage est intermédiaire entre Algol et PL/I.

3.22 - LES LANGAGES DE CONSTRUCTION DE SYSTEMES GRAPHIQUES.

3.221 - Langages à haut niveau.

Il existe plusieurs langages de programmation graphique à haut niveau qui servent à construire des systèmes graphiques plutôt que des programmes normaux. A cause de la puissance des outils qu'ils fournissent, ces langages ne constituent qu'une partie de systèmes complexes et généraux dont le but est la définition de systèmes particuliers; on peut aussi considérer qu'il s'agit de moyens de définition par l'utilisateur de son propre langage, sous une forme adaptée à son problème précis (ce que les américains appellent parfois "problem-oriented languages, par opposition aux classiques "procedure-oriented languages).

Le langage GPL/I [Jo68, Sy68] est une extension du langage PL/I, qui définit de nouvelles instructions tout en conservant la syntaxe du langage et en en utilisant au maximum les possibilités. C'est ainsi qu'il se sert de la syntaxe de la déclaration des structures et de leur utilisation, qu'il effectue la génération des objets au moyen de l'instruction d'affectation avec appel de fonctions génératrices, ou qu'il utilise le bloc "ON CONDITION" pour indiquer le traitement des interruptions au moyen de sous-programmes (ou fragments de programmes) asynchrones. La deuxième particularité fondamentale de GPL/I est qu'il fournit deux systèmes de représentation structurée internes et bien définis; le premier système sert à la manipulation des données purement graphiques ou logiques, au moyen d'une mémoire associative simulée qui permet la gestion de triplets du type "type(objet)=valeur"; le deuxième système sert à la manipulation de la liste d'affichage, au moyen d'une structure en anneau ou en plex d'un type classique [Su63, Ro..]. Le langage fournit de plus les outils de passage d'une structure à l'autre, et contient des moyens très puissants mais très specialisés et impossibles à modifier: il ne peut donc pas de façon pratique être utilisé par un programmeur normal, et n'est d'ailleurs pas prévu pour cela.

Le système de la Bell Telephone porte plusieurs noms suivant les références: Graphic-1 ou 2 [Ni65, ChPi65], Bellgraph [Ko68], Grafcom, GRIN-2, et j'en oublie. Il comprend une configuration technologique assez spéciale, qui fait intervenir plusieurs périphériques graphiques de nature différente, un langage de très bas niveau syntaxique qui fournit des outils de très haut niveau, et une structure de données originale dans son utilisation sinon dans sa réalisation; cette structure conduit en effet à une représentation en arbre où les branches peuvent être multiples et portent de l'information au même titre que les noeuds et les feuilles. Le système est fait pour éviter à l'utilisateur d'écrire un véritable programme, et lui fournit donc un ensemble d'outils puissants mais non modifiables.

Le système PLAN-PGS [ChDo68, IBM10, IBM11] ne contient même pas de véritable langage de programmation, mais il permet la définition par le programmeur d'une sorte de langage de commande ainsi que d'outils spécialisés définis au moyen de programmes écrits par exemple en Fortran. Ce système étant destiné à la résolution de problèmes généraux au moyen d'un terminal graphique, et non pas à la résolution de problemes spécifiquement graphiques, il ne fournit pas de structure de données utilisable de l'extérieur.

Le langage GULP [Pa68], bien que présenté comme un compilateur de compilateur, est en fait d'un pricipe assez semblable à PLAN-PGS. Comme ce dernier, il permet à l'utilisateur de se définir un pseudo-langage de commande dans le cadre d'une syntaxe fixée à l'avance. Cependant, ce langage est spécifiquement orienté vers les traitements graphiques; d'autre part, sa principale originalité est l'extension de la notion de caractère: le langage que se définit l'utilisateur est formé de chaînes de "caractères", ceux-ci comprenant aussi bien un caractère au sens habituel qu'une chaîne de caractères, un mouvement du manche à balai, une détection au pointeur optique ou l'appui sur une touche de fonctions. Il s'agit en fait de réduire toutes les actions de l'utilisateur à une séquence linéaire d'évènements élémentaires qui constituent les composants du langage utilisé. Le système GULP lui-même utilise de façon intensive des structures de données complexes, mais n'en permet pas l'accès à l'utilisateur.

3.222 - Langages à bas niveau.

Pour la construction par un programmeur spécialiste d'un système de traitements graphiques plus ou moins général, un langage tel que GPL/I cité plus haut risque de se placer à un niveau trop élevé pour permettre facilement les opérations fines et complexes que nécessite

la programmation des systèmes. Malgré ce que l'on aurait pu croire un moment, il semble en effet que la réalisation effective (et non pas la description ou l'expérimentation) de programmes de ce genre réclame encore, et peut-être pour longtemps, l'emploi du langage d'assemblage de l'ordinateur concerné. Il est donc intéressant de pouvoir intégrer dans ce cadre un véritable langage de programmation graphique qui fournisse des outils d'un niveau suffisant; cela va à l'encontre des principes que j'ai formulés précédemment (paragraphe 3.1), mais il s'agit d'un cas très particulier, et d'un langage qui n'est pas destiné à l'usage courant.

Le seul exemple que je donnerai dans ce domaine est le langage Macro-Euphémie déjà décrit par ailleurs [Cl69], et sur lequel je reviendrai (paragraphe 4.22). Il s'agit d'une extension du langage d'assemblage de l'ordinateur 360, réalisée au moyen de macros-instructions assez complexes; les outils utilisables par l'intermédiaire de ces nouvelles instructions sont fournis par l'ensemble de sous-programmes G.S.P. [IBM2, Ru68), et sont au niveau des figures. Le langage dont on dispose avec Macro-Euphémie peut en fait être considéré comme un ensemble d'instructions d'un niveau relativement élevé, dans lequel on peut insérer quand c'est nécessaire des instructions du langage d'assemblage. En raison de ce à quoi devait servir ce langage, on n'y a pas intégré d'instructions non graphiques, qui auraient permis de le rendre plus homogène. Cet outil a en tout cas prouvé son utilité, puisque c'est lui qui est utilisé pour toutes les parties graphiques du système Eulalie. On trouvera en annexe 4 des exemples d'utilisation de ce langage dans certaines parties caractéristiques du superviseur du système.

3.23 - LES LANGAGES PUREMENT GRAPHIQUES.

L'archétype des langages de cette catégorie est fourni par le système Sketchpad [Su63], qui a pris dans l'esprit des utilisateurs de terminaux graphiques une place tout à fait exagérée par rapport aux objectifs qu'il visait, pour la simple raison qu'il a été le premier exemple d'utilisation complète et organisée d'un écran cathodique couplé avec un ordinateur. Les objectifs de Sketchpad sont en effet assez limités, même si l'on a pu le qualifier d'outil général de résolution de problèmes, puisqu'il s'agit en fait d'un système de dessin aidé par ordinateur: on peut construire des schémas sur l'écran du terminal, les conserver, les modifier, les mélanger, les déformer, bref les manipuler à l'infini et de toutes les façons possibles, mais c'est à peu près tout ce que l'on peut en faire.

La première particularité intéressante de Sketchpad est qu'il ne renferme pas à proprement parler de langage de programmation, ni même de véritable langage de commande: l'utilisateur dialogue avec le système en utilisant un jeu très riche de boutons, touches, leviers, poussoirs, manettes ou autres, et surtout le pointeur optique; certains de ces dispositifs jouent le rôle d'un clavier de fonctions, d'autres permettent d'envoyer à l'ordinateur un simple nombre, enfin le pointeur optique sert à trois fonctions: introduire des coordonnées, désigner des objets et effectuer des tracés. C'est l'ensemble de ces manipulations possibles qui constitue en quelque sorte un langage, en ce sens que, par exemple, la signification d'un déplacement du pointeur optique ou d'une désignation d'objet dépend de la dernière touche de fonction actionnée ou de l'objet désigné précédemment.

La deuxième possibilité intéressante de Sketchpad est la possibilité qu'il offre d'appliquer des contraintes aux objets graphiques, et de respecter ces contraintes, de facon pondérée, quand on déforme une figure en en déplaçant les éléments. Cette possibilité s'appuie sur une structure de représentation des données en forme d'anneau, au demeurant assez lourde et redondante, qui n'a pas à être connue de l'utilisateur.

Un deuxième système intéressant dans cette catégorie, qui présente de notables différences avec Sketchpad, est le système GENIAL [Lu68-2, Lu68-3, Lu70]. Il s'agit là aussi d'un outil de fabrication de dessins au moyen de l'ordinateur, mais qui fonctionne sur un matériel beaucoup plus rudimentaire que le précédent.

La première particularité de GENIAL est qu'il reconnaît un langage de commande d'une forme beaucoup plus habituelle que celui de Sketchpad, puisque les commandes sont frappées sur un terminal de type machine à écrire, et ressemblent assez à un langage de programmation. A part l'absence de branchement, boucles et instructions conditionnelles, ce qui distingue ce langage de commande d'un langage de programmation est d'une part que le système frappe certains éléments des commandes à la place de l'utilisateur, d'autre part que certains autres éléments sont introduits par l'utilisateur au moyen du pointeur optique (désignation ou coordonnées).

La deuxième particularité de ce système est que la principale des opérations que l'on peut effectuer sur les objets qu figures fabriquées est de les <u>animer</u>, c'est-à-dire de leur appliquer en temps réel certains mouvements pré-définis et éventuellement composés de mouvements élémentaires. Des images animées peuvent se

Catégorie	Outils	Observations
Génération	Objets: Point Position Vecteur Texte Courbe définie par points Courbe définie par fonction Affectation = génération	Nommer les objets est impossible. On ne peut qu'ajouter des objets ou des figures aux figures existantes, et plus difficilement en enlever
Transmission	Affichage après évaluation Effacement Transformation analytique définie par sous-programme	On réutilise l'instruction d'affectation-génération pour faire subir des modifications aux figures.
Communication	Sources d'interruptions: détection par pointeur optique sur une figure; touche de fonction; fin de régénération spécification de lieu de transfert Instruction d'attente	Le clavier alphanumerique ne peut être utilisé que comme un lecteur de cartes Il manque l'instruction qui permette d'ignorer une interruption

Tableau VI - Le langage Laprec.

3.214 - LE LANGAGE EUPHEMIE: [LeC168, C169] sera décrit plus longuement par la suite puisqu'il constitue l'un des points majeurs de ce travail. Je voudrais cependant indiquer dans le cadre de ce chapitre quelques-unes de ses caractéristiques principales. Il s'agit tout d'abord d'un langage de programmation simple de l'étage des figures, destiné à l'utilisation d'un terminal graphique avec mémoire d'entretien. Les outils de génération comprennent les déclarations des objets, séquences et figures, et une instruction de génération universelle qui sert pour tous les types d'objets; l'appartenance d'un objet à une figure est statique s'il est nommé c'est-à-dire déclaré. Les outils de transmission comprennent l'affichage, l'extinction, le rallumage et l'effacement des figures, des séquences et des objets, et en plus une instruction de mise à jour

succéder par séquence pour constituer finalement un dessin animé complexe.

4 - LE LANGAGE EUPHEMIE.

Ce chapitre décrit complètement le langage Euphémie, dont j'ai déjà commencé à parler au paragraphe 3.214, mais il le fait sans référence à l'environnement dans lequel on peut utiliser ce langage, c'est-à-dire le système Eulalie (celui-ci sera décrit au chapitre 6). La première partie de ce chapitre est un manuel de programmation destiné à l'utilisateur éventuel, tandis que la deuxième expose les principes qui ont guidé la conception du langage; les méthodes adoptées pour l'implantation d'Euphémie seront elles aussi décrites dans le chapitre 6.

4.1 - MANUEL DE PROGRAMMATION DU LANGAGE EUPHEMIE.

4.11 - PRESENTATION, SYMBOLES ET CONVENTIONS.

Le langage Euphémie est un langage de programmation d'un aspect classique et d'un niveau semblable à Algol ou PL/I, mais orienté spécialement vers l'utilisation d'un terminal graphique. On peut donc classer en deux catégories les outils qu'il fournit: les outils graphiques servent à l'utilisation du terminal, les outils non graphiques servent aux calculs arithmétiques classiques, à quelques manipulations de chaînes de caractères, et aux opérations de branchement (allerà en Algol, GOTO en PL/I), boucles (pour en Algol, DO en PL/I) et décisions (si ... alors ... sinon en Algol, IF ... THEN ... ELSE en PL/I). Toutes les variables utilisées doivent être déclarées, et les déclarations pourront à nouveau être réparties entre déclarations graphiques et déclarations non graphiques.

Un programme est formé d'une suite de déclarations puis d'instructions, qui sont des séquences de symboles terminées par un point-virgule. Les symboles sont formés au moyen des caractères suivants:
- 26 lettres ABCDEFGHIJKLMNOPQRSTUVWXYZ
- 10 chiffres 0123456789
- 26 caractères spéciaux :=<;'>*() _+!-F&#,.¬?/¦"∂∉ et l'espace

Les symboles, constituants des déclarations et instructions, sont des identificateurs, des symboles de base ou des constantes.

Les identificateurs servent à nommer les variables, graphiques ou non, utilisées dans le programme; un identificateur est une suite de lettres ou de chiffres, commençant par une lettre et formée de un à huit caracterès; par exemple:

A DEBUT X2345678 ALPHABET AZOR X4W3Z7P9 B E1

sont des identificateurs, alors que:

A'B ALPHABETIQUE 5ERTY O.LECARME

n'en sont pas.

Les symboles de base peuvent être répartis en deux catégories; les symboles de base simples sont formés d'un ou plusieurs caractères speciaux, et sont au nombre de 26, représentés ici séparés par des virgules (la virgule en constitue un elle-même):

+, -, *, **, /, <, <=, =, >=, >, ¬=, ||, (,), :, :=, ;, <==, <=*, <#=, <##, <==>, <=#>, <*=>, <##>

Ils servent le plus souvent d'opérateurs. Les symboles de base composés sont formés d'une chaîne de lettres placée entre deux apostrophes; ils sont très nombreux et en principe mnémoniques; un même symbole de base composé a une forme anglaise et une forme française, et peut être abrégé jusqu'à une longueur limite qui dépend du symbole, ceci afin d'éviter les ambiguïtés; ainsi:

'CARACTERES' 'CARACTERE' 'CARA' 'CHARACTER' 'CHAR'

constituent syntaxiquement le même symbole, qui dans la suite de la description sera toujours noté caractères. Les symboles de base composés seront en effet notés dans leur forme française la plus longue, en minuscules soulignées; on trouvera au paragraphe 4.19 un tableau complet des symboles de base composés avec leurs formes française et anglaise et leur longueur minimale dans leurs deux formes.

Les derniers des symboles sont les constantes, qui peuvent être de trois types. Une constante entière est formée d'une suite de chiffres précédée éventuellement d'un signe, elle représente un nombre entier qui doit être compris entre 2^{-31} et $2^{31}-1$. Une constante réelle est formée de trois parties dont deux quelconques peuvent être absentes: la partie entière est une suite de chiffres, précédée éventuellement d'un signe, la partie décimale est une suite de chiffres précédée d'un point, le facteur de cadrage est une constante entière comprise entre -78 et +75 et précédée d'un point d'exclamation; le total peut être précédé d'un signe, et représente un nombre réel en virgule flottante; ainsi:

3.1416 .31416!1 +314.16!-2 31416!-4

représentent le même nombre;

10 +10 1!1 +1!+01 .0001!5 1.0000!+1

représentent le même nombre (il faut d'ailleurs remarquer dans ce

cas que les quatre premières écritures sont celles de nombres entiers).

Enfin, une constante de type chaîne est formée d'une suite de caractères quelconques comprise entre deux guillemets (ne pas confondre le guillemet " qui délimite les chaînes et l'apostrophe ' qui délimite les symboles de base composés); si la suite de caractères doit contenir un guillemet, on le représente au moyen de deux guillemets immédiatement consécutifs; l'intérieur d'une chaîne est le seul cas où l'espace ou blanc est pris en considération; partout ailleurs, il est sans signification; les exemples suivants montrent quelques chaînes et leur longueur en nombre de caractères:

"0123456789" est de longueur 10
"JUSQU'AU BOUT" est de longueur 13
"LE ""PETIT"" BOUT" est de longueur 15
""""""" est de longueur 2
"" est de longueur 0

La suite de ce manuel de programmation ne donne pas de description syntaxique complète du langage Euphémie. Celle-ci peut être trouvée, dans une notation proche de celle qui va être utilisée, dans l'ouvrage de référence [C169]; d'autre part, on trouvera également cette description en annexe 3 de ce travail, sous la forme d'un programme en langage SNOBOL4 qui sert précisement d'analyseur syntaxique du langage Euphémie. Au cours de ce qui suit, on ne trouvera donc de description syntaxique que dans le cas où cela peut ajouter des informations au texte et aider à sa compréhension. Le symbolisme adopté est très simple: les unités syntaxiques sont nommées avec des mots écrits en minuscules et non soulignés, éventuellement groupés par des tirets; les parenthèses servent à mettre en facteur un groupe d'unités syntaxiques relativement à trois opérations possibles; ces opérations sont le choix entre deux unités, représenté par l'opérateur binaire ¦, l'omission possible d'une unité, représentée par l'opérateur unitaire ?, et la répétition possible d'une unité, représentée par l'opérateur unitaire &. Ainsi,

A ¦ B ¦ C correspond à A, B ou C
A ¦ (B ?C) correspond à A, B ou BC
(A ¦ &B) C correspond à AC, BC, BBC, BBBC, ...

Les descriptions syntaxiques indiquent toujours un certain ordre pour les éléments des instructions; en fait, quand un élément représentant une option est désigné par un mot-clé, sa place est sans signification. Ainsi, la description suivante:

écrire E.A. en E.A., E.A. ?(nombre E.A.) ?(entier ¦ réel ¦ chaîne);

peut correspondre aux écritures suivantes:

```
écrire N en X, Y nombre 10 réel;
écrire N réel nombre 10 en X, Y;
écrire N nombre 10 réel en X, Y;
écrire 10 entier en 20, 2;
écrire A*X+B en Z, T(I)+K chaîne nombre SIN(X);
```

4.12 - DECLARATIONS NON GRAPHIQUES.

Les variables non graphiques d'un programme en Euphémie sont les variables entières ou réelles, les tableaux entiers ou réels et les chaînes. Toutes ces variables sauf les tableaux peuvent recevoir une valeur initiale au moment de leur déclaration. Les tableaux n'ont qu'une dimension, et leur borne inférieure est toujours 1. Les chaînes ont une longueur déclarée, qui est en fait leur longueur maximale.

Les déclarations des entiers ont la forme:

```
  entier identificateur ?(:= entier) ?&(, identificateur ?(:=
entier)) ;
```

ce qui correspond par exemple à:

```
entier X, I:=1, Y;
entier J;
```

Les déclarations des réels ont la même forme:

```
  réel identificateur ?(:= nombre) ?&(, identificateur ?(:=
       nombre));
```

ce qui correspond à:

```
réel AA:=12.5, BB, CC:=-3, PI:=314.159265351-2;
réel Z;
```

Pour les tableaux, on donne le nombre d'éléments entre parenthèses après le dernier identificateur d'une liste de noms de tableaux de même longueur, cette liste étant elle-même précédée du type des tableaux qui la composent:

```
tableau élément-de-tableau ?&(, élément-de-tableau);
```

avec pour élément-de-tableau:

```
(réel | entier) identificateur ?&(, identificateur) [ entier-
     sans-signe ]
```

(on souligne les parenthèses qui constituent des symboles du langage décrit); voici donc des exemples de déclarations de tableaux:

```
tableau réel X1, Y1(24), entier T1, T2, T3(10), entier U(100);
tableau entier TABLEAU(10);
```

Enfin, la déclaration des chaînes précise leur longueur en nombre de caractères, et éventuellement leur valeur initiale, la longueur pouvant se rapporter à une liste de chaînes:

chaîne élément-de-chaîne ?&(, élément-de-chaîne);

avec pour élément-de-chaîne:

entier-sans-signe caractères identificateur ?(:= chaîne) ?&(, identificateur ?(:= chaîne))

Voici des exemples de déclarations de chaînes:

```
chaîne 10 caractères A:="BLABLA", B, C:="0123456789", 26 caractères D, E, ALPHABET:="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
chaîne 1 caractere CHAINE;
```

Une remarque importante à faire est que ces déclarations ne peuvent pas utiliser les valeurs de variables, ni comme valeur initiale, ni comme dimension de tableau, ni comme longueur de chaîne; les écritures suivantes sont erronées:

```
entier I:=J;
tableau réel T(N);
chaîne K caractères TRUC;
```

Il existe un dernier type d'entité non graphique identifiable, c'est l'étiquette; une étiquette se déclare elle-même par sa seule apparition, et sert à nommer une instruction pour une référence ultérieure au moyen de l'instruction allerà. Une étiquette est donc un identificateur qui peut précéder n'importe quelle instruction, en étant séparé d'elle par le caractère "deux points". Une instruction ne peut avoir qu'une étiquette, une déclaration ne peut pas en avoir du tout.

Instruction non étiquetée:

```
Y := A / X + B;
```

Instruction étiquetée:

```
RETOUR:  Y := A / X + B;
```

4.13 - INSTRUCTIONS NON GRAPHIQUES.

4.131 - Expressions.

Une expression arithmétique d'Euphémie se forme très exactement suivant les mêmes principes que l'expression arithmétique simple d'Algol 60. On dispose donc de cinq opérateurs arithmétiques groupés en trois priorités:

** (exponientation)
* et / (multiplication et division)
+ et - (addition et soustraction)

Les parenthèses servent à modifier l'ordre des priorités quand on le

désire. Une expression arithmétique peut mélanger des variables et des constantes, entières ou réelles, les conversions nécessaires étant faites s'il y a lieu, toujours dans le sens d'un entier vers un réel. Une référence à un tableau est une variable; elle se forme en plaçant derrière le nom du tableau et entre parenthèses la valeur de l'index, qui est à son tour une expression arithmétique. Il n'y a pas d'expression arithmétique conditionnelle au sens d'Algol 60. Les seules fonctions utilisables sont les fonctions standards, définies par le système; leurs identificateurs n'ont pas à être déclarés. Il s'agit de sinus (SIN), cosinus (COS), racine carrée (RAC2), exponentielle (EXP), arctangente (ATAN) et valeur absolue (ABS).

Si une variable graphique (figure, objet, séquence ou arrêt; voir le paragraphe 4.14) apparaît dans une expression, elle est considérée comme un entier. Le seul intérêt de cette utilisation est de permettre l'affectation du nom d'une variable à une autre (voir un exemple au paragraphe 4.176).

Une expression de chaîne peut utiliser l'opérateur binaire de concaténation, noté ||, et l'opérateur composite sélection ... depuis ... longueur ..., qui est plus prioraire que le précédent mais moins que les opérateurs arithmétiques; ce dernier opérateur sert à extraire d'une chaîne (indiquée après sélection) une sous-chaîne dont on donne la position de début (après depuis) et la longueur (après longueur); si la position de début n'est pas indiquée, c'est le premier caractère; si la longueur n'est pas indiquée, on prend toute la fin de la chaîne. Par exemple, avec les déclarations du paragraphe précédent:

ALPHABET || C

a pour valeur

"ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"

sélection C depuis 2 longueur 9 a pour valeur "123456789", de même que sélection C depuis 2

sélection ALPHABET longueur 4 || "T" a pour valeur "ABCDT"

Après sélection peut figurer une expression de chaîne, après longueur et depuis peuvent figurer des expressions arithmétiques.

4.132 - Instruction d'affectation.

Une instruction d'affectation sert à donner une valeur à une variable; elle peut être arithmétique ou de chaîne, et on peut affecter une expression de chaîne à une variable arithmétique ou une expression arithmétique à une chaîne. On peut affecter une expression à une sous-chaîne délimitée par l'opérateur composite sélection. Il n'y a pas d'instruction d'affectation multiple.

L'opérateur d'affectation est le symbole ":=". Voici des exemples d'instructions d'affectation utilisant les déclarations précédentes:

```
I:=X+Y;
BB := 0;
AA:= AA ** I + PI * X1(Y) / 180 + T3(I+2);
E1:  J:= AA;
DEBUT:  A:= sélection C depuis I longueur J/4 || CHAINE;
E := B;
D := "VIVE LE LANGAGE EUPHEMIE !";
sélection D depuis 9 longueur 16 := "SYSTEME EULALIE";
E2:  X1(T3(I+2)):=-6;
```

L'affectation à une chaine est le seul cas un peu compliqué. D'une part, il peut conduire à des conversions, d'autre part la longueur des chaînes est variable, entre la longueur zéro (chaîne vide) et la longueur indiquée au moment de la déclaration. Si l'on affecte à une chaîne une autre chaîne trop longue, cette dernière est tronquée à droite; si c'est une chaîne plus courte que ce qui a été indiqué au moment de la déclaration, la chaîne réceptrice prend la longueur de celle qu'on lui affecte. La longueur d'une chaîne est accessible au moyen de la fonction standard LONGUEUR, à valeur entière.

Pour la conversion d'une chaîne en un nombre, on cherche la représentation d'un nombre à partir du premier caractère à gauche de la chaîne; on commence au premier caractère légal, et on s'arrête au premier caractère illégal; s'il n'y a aucun caractère légal, le résultat est la valeur zéro. Les exemples ci-dessous montrent ce qui se passe:

```
I:="     ";      I=0
I:= "ABRACADABRA";      I=0
I:= "X+Y-2/A";      I=-2
I:= "";      I=0
I:= "256!-2";      I=2.56 (I=3 s'il est de type entier)
I:= "ZUT!+3";      I=1000
```

Pour la conversion d'un nombre en une chaîne, on écrit un entier sous la forme ±XXXXXXXXXX, et un réel sous la forme ±X.XXXXXX!±YY, les X et Y représentant les chiffres. Pour un entier, seuls les chiffres significatifs sont produits, et le signe + n'est pas écrit, si bien que la chaîne résultante peut occuper de 1 à 11 caractères; pour un réel, la longueur est toujours de 13 caractères. Avec les déclarations précédentes, on obtient les résultats suivants:

```
B:= PI;      B="+3.141593!";
CHAINE := 2;      CHAINE="2"
D := AA;      D="+1.250000!+01"
```

```
CHAINE:= PI;      CHAINE="0"
C:= -12*40;       C:="-480"
```

4.133 - Instruction de branchement.

L'instruction de branchement sert à transferer le contrôle à une autre instruction que celle qui la suit immédiatement. Elle a la forme:

allerà étiquette ?(sinon ‖ finsi ?étiquette) ;

Si l'étiquette n'existe pas, cette instruction termine le programme. Les options sinon et finsi seront précisées au paragraphe 4.135.

4.134 - Instructions de boucle.

Un groupe d'instructions à répéter un certain nombre de fois sous contrôle d'une certaine variable doit être encadré par une instruction pour et une instruction finpour. L'instruction pour indique le nom de la variable arithmétique de contrôle, la valeur initiale à lui affecter, la valeur finale à atteindre et le pas de la progression sous la forme:

pour variable := expression-arithmétique ?(pas expression-arithmétique) à expression-arithmétique ;

L'instruction finpour s'écrit:

finpour ?étiquette ;

Quand, au cours de l'exécution des instructions du programme, on tombe sur l'instruction finpour, cela signifie que l'on termine la boucle commencée par la dernière instruction pour exécutée; il n'y a donc aucune obligation que finpour se trouve matériellement placé après le pour correspondant, et un même finpour peut terminer plusieurs pour différents suivant le chemin par où l'on est arrivé, comme le montre l'exemple suivant:

```
si X < 10;
allerà DEUX finsi;
pour I := 0 pas 2 à X;
allerà TROIS;
DEUX:  pour I := 0 pas 2 à 10;
TROIS :  ...
...
finpour;
```

Si l'on tombe sur une instruction finpour et qu'il n'y a plus d'instruction pour à terminer, c'est une erreur; il n'y a aucune restriction sur les instructions que l'on exécute dans une boucle, mais on voit donc que le seul moyen de sortir d'une boucle avant la fin est de modifier la variable pour que la boucle soit automatique-

ment terminée:

```
pour I := 1 à N;
...
si T(I) = 0;
I := N + 1;
finpour
allerà ERREUR;
finsi;
...
finpour;
```

Si on le désire, on peut soit contrôler que l'on termine bien le pour voulu, soit terminer plusieurs pour à la fois, en mettant après finpour l'étiquette de l'instruction pour que l'on termine:

```
BOUCLE :  pour I := 1 à 10;
pour J := 1 à 10;
...
...
finpour BOUCLE;
```

Les expressions arithmétiques après pas et à sont évaluées avant la première itération; si le pas n'est pas indiqué, il vaut 1. Si lors de la première exécution de l'instruction pour on a déjà dépassé la valeur finale, on passe sur les instructions suivantes sans les exécuter jusqu'à trouver une instruction finpour. Etant donné qu'en particulier les instructions de branchement ne sont pas exécutées dans ce cas, l'exemple suivant ne produira pas les résultats escomptés si K est supérieur à L, car on exécutera les instructions comprises entre les étiquettes E5 et E6:

```
pour I := K à L;
...
allerà E2:
E3 :  ...
...
finpour;
E5 :  ...
E6 :  allerà E4 ;
E2 :  si T1(I) = 0;
allerà E3;
finsi;
finpour;
E4 :  ...
```

4.135 - Instructions de décision.

Il existe trois instructions de décision, qui se comportent entre elles de façon semblable à pour et finpour; la forme la plus normale et la plus complète est la suivante:

```
si
...
sinon;
...
finsi;
```

La première séquence d'instructions est exécutée si la condition est satisfaite, et la seconde si elle ne l'est pas; l'une ou l'autre des deux séquences peut être vide, et on peut omettre l'instruction sinon si l'instruction finsi la suit immédiatement.

La condition qui suit si est une comparaison entre deux expressions arithmétiques (au moyen des opérateurs <, <=, =, >=, > et ¬=) ou entre deux expressions de chaîne (au moyen des opérateurs = et ¬=). On peut aussi composer des conditions pour construire des expressions booléennes au sens d'Algol, si ce n'est qu'il n'existe pas de variables booléennes, et que les seuls opérateurs sont | (union), & (intersection) et ¬ (négation).

L'instruction sinon termine l'instruction si que l'on a exécutée en dernier, de même que l'instruction finsi. Par conséquent, on procède de la façon suivante:

- Evaluation de la condition.
- Si elle est satisfaite:
  Exécution des instructions qui suivent jusqu'à trouver sinon ou finsi; si l'on trouve sinon, saut des instructions qui suivent jusqu'à trouver finsi, mais en décomptant les nouveaux si, sinon et finsi qui peuvent survenir dans ce balayage séquentiel.
- Si elle n'est pas satisfaite:
  Saut des instructions jusqu'à trouver sinon ou finsi, où l'on reprend l'exécution; pendant le saut, décompte des nouveaux si, sinon et finsi comme précédemment.

Pour expliquer ce phénomène un peu complexe, il suffit de savoir que l'on utilise à l'exécution du programme une pile dans laquelle on place les instructions pour et si, avec leur étiquette s'il y a lieu. Si l'on tombe sur une instruction finpour, on doit trouver en haut de cette pile une instruction pour, sinon c'est une erreur. Si l'on trouve une instruction finpour avec une étiquette en opérande, on recherche dans la pile une instruction pour avec l'étiquette cherchée, en terminant toutes les instructions pour que l'on rencontre au passage (terminer une instruction pour signifie faire

progresser la variable de contrôle, et reprendre la boucle si elle n'a pas dépassé la valeur finale); si en chemin l'on trouve une instruction si, ou si l'on ne trouve pas une instruction pour avec l'étiquette cherchée, c'est une erreur. Si l'on tombe au cours de l'exécution du programme sur une instruction sinon ou finsi, on doit trouver en haut de la pile une instruction si; on peut mettre en opérande de sinon ou finsi l'étiquette de l'instruction si que l'on termine.

Quand on saute des instructions, on effectue les empilages et dépilages d'instructions pour et si que l'on rencontre. Dans le cas normal où les instructions sont physiquement dans leur ordre logique et où il n'y a pas de chevauchements entre instructions de boucles et instructions de décision, tout se passe comme l'indique l'écriture même du programme. Il ne peut surgir de problèmes que si l'on travaille de facon plus compliquée: si l'on inclut le premier exemple du paragraphe 4.134 dans une instruction de décision, il faudra placer après l'instruction "allera TROIS;" une instruction finpour logiquement inutile et inaccessible, ou bien étiqueter la première instruction pour et mettre son étiquette en opérande de l'instruction finpour qui termine l'exemple.

L'exemple suivant fonctionnera dans tous les cas de figure, grâce à l'utilisation des étiquettes en opérandes de finpour, sinon ou finsi:

```
E10:  si A = B;
si B = C;
réel  identificateur ?(:= nombre) ?&(, identificateur ?(:=
      entier
pour J := 1 à N;
...
finpour E11; (cette instruction équivaut à deux finpour)
sinon E10; (cette instruction équivaut à finsi suivi de sinon)
si C = D;
pour K := 1 à P;
...
finpour;
sinon;
pour L := 1 à Q;
...
finpour;
finsi E10; (cette instruction équivaut à deux finsi)
```

Une instruction allera qui sort d'une instruction de décision pose un problème dans ce contexte, puisque l'on perd ainsi un sinon ou un finsi dans la gestion des piles. Pour que l'ensemble continue

de fonctionner, on utilise l'option sinon ou finsi de l'instruction allerà: l'instruction

    allera E1 finsi;

est apparemment équivalente aux deux instructions:

    allera E1;
    finsi;

En fait, seule la première forme permet de retirer convenablement de la pile le si correspondant; cette forme est traitée exactement comme la deuxième quand il s'agit de sauter des instructions. On trouvera par la suite de nombreux exemples d'utilisation de cette forme de l'instruction allerà.

4.14 - DECLARATIONS GRAPHIQUES.

4.14.1 - Figures.

Les images qu'un programme en Euphémie doit faire apparaître sur l'écran sont constituées d'objets graphiques tels que des points ou des segments. Ces objets sont regroupés en figures, qui servent à indiquer leurs caractéristiques et à permettre leur manipulation; au cours de l'existence d'un objet, la figure dont il fait partie sert en effet à indiquer le système de coordonnées et l'échelle utilisés, l'emplacement de la mémoire de l'ordinateur où doivent être construits les ordres de génération correspondants, le type de ces ordres, l'emplacement de la mémoire d'entretien où doit être placé le fragment de la liste d'affichage ainsi construit, et quelques autres caractéristiques encore. La déclaration d'une figure sert donc à indiquer toutes ces propriétés. Sa forme générale est simple:

    figure nom-de-figure ?(liste-de-propriétés) ;

Si la liste de propriétés est absente, c'est que l'on se contente d'utiliser les propriétés par défaut, qui seront indiquées ci-dessous pour chaque propriété, avec les indications appropriées sur chacune.

    ?(entrée (réelabs | réelrel | entabs | entrel)
        ?(, (réelabs | réelrel | entabs | entrel)))

Cette propriété indique le type des coordonnées numériques qui seront fournies pour la génération des objets de la figure. Ces coordonnées peuvent être des nombres entiers ou réels, et constituer des coordonnées absolues (position sur l'écran) ou relatives (déplacement par rapport aux coordonnées précédentes). Le premier symbole indique le type des abscisses, le deuxième celui des ordonnées, qui est le même que celui des abscisses s'il n'apparaît pas. En cas d'absence de cette propriété, l'option par défaut est réelabs pour

les deux coordonnées (réel absolu). Cette propriété ne sert qu'aux fins d'optimisation: en effet on peut toujours mettre des coordonnées en un mode absolu ou relatif différent de celui de la figure.

?(sortie (optimisé | absolu | incrémentiel))

Cette propriété indique le type des ordres de génération construits pour les objets de la figure; ces ordres peuvent être absolus, auquel cas ils indiquent des positions sur l'écran, ou incrémentiels, auquel cas ils indiquent des déplacements par rapport aux positions atteintes auparavant. On peut utiliser à la fois les deux types d'ordres, suivant la distance à laquelle se trouve le prochain point à atteindre, car cela permet d'optimiser la place occupée en mémoire d'entretien par la liste d'affichage et le temps pris par un cycle de régénération. On verra au paragraphe 4.16 les cas dans lesquels il faut utiliser un type d'ordres plutôt qu'un autre; si cette propriété est absente, l'option par défaut est optimisé. Il faut bien remarquer que le type des coordonnées est sans rapport avec le type des ordres, et que l'on peut très bien générer des odres absolus avec des données relatives, ou des ordres incrémentiels avec des données absolues.

?(zone borne, borne, borne, borne ?(écran borne, borne, borne, borne, borne))

Cette propriété et la suivante constituent les seuls cas où l'on puisse utiliser une variable dans une déclaration; en effet, "borne" peut être une variable ou un nombre, à condition que toutes les variables utilisées aient une valeur au moment où l'on initialisera la figure au moyen d'une instruction début figure (voir le paragraphe 4.15). Il s'agit de définir la zône rectangulaire de l'écran que peut occuper la figure, en donnant les coordonnées des coins en bas à gauche et en haut à droite de cette zone. Si l'option écran n'apparaît pas, les bornes de la zone sont données par rapport aux unités-écran, ce qui correspond à:

écran 0, 0, 4095, 4095

L'option écran permet de se définir un système de coordonnées de l'écran plus pratique, par exemple 0, 0, 1, 1, ou bien -10, -10, 10, 10. Les deux indications suivantes sont équivalentes:

zône 0, 1, 1, 2 écran 0, 0, 4, 8

zône 0, 511, 1023, 1023

Elles correspondent à la figure 4.1. Si la propriété complète est absente, l'option par défaut est que la zône occupe tout l'écran. Il faut noter que les zônes occupées par des figures différentes peuvent fort bien se recouper et se recouvrir.

?(données borne, borne, borne, borne)

Il s'agit cette fois-ci de placer dans la zone de l'écran

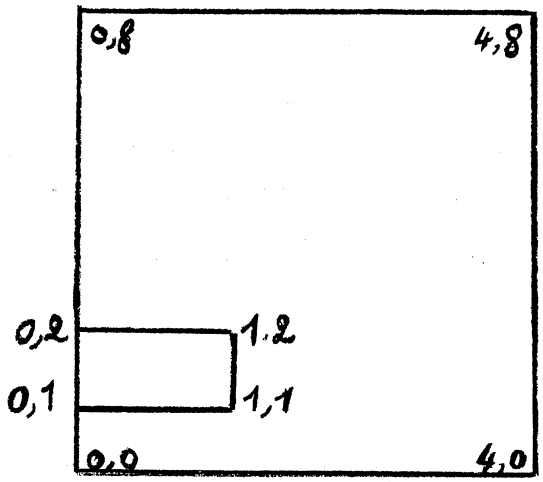


Figure 4.1 - Définition d'une zône sur l'écran.

associée à la figure et définie précédemment le système de coordonnées qui sera utilisé dans les générations d'objets. Plutôt que de donner l'emplacement de l'origine des axes et le module sur chaque axe (ce qui déterminerait les facteurs d'échelle), on préfère donner encore une fois les coordonnées des coins en bas à gauche et en haut à droite de la zône occupée par la figure, mais cette fois-ci par rapport aux systèmes d'axes à définir. C'est ainsi que l'indication:

données -1, -1, 1, 1

concernant une zône carrée place l'origine du système d'axes au centre de ce carré, et définit un module identique sur les deux axes, tel qu'un côté du carré ait pour longueur 2. Par rapport à la même zône, l'indication:

données -1, 0, 1, 10

place l'origine du système d'axes au milieu du côté inférieur du carré, et définit un module sur les ordonnées cinq fois plus petit que sur les abscisses, ce qui fournit un moyen simple de faire subir une affinité aux figures géométriques affichées. Si les propriétés données, zône ou écran sont absentes, on suppose à leur place 0, 0, 4095, 4095, c'est-à-dire que l'on travaille en unités-écran.

?(coupage (ec | ea | fc | fa | libre))

Cette propriété indique de quelle manière on doit traiter les parties des objets génères qui sortent des limites de la figure ou de l'écran; e signifie que l'on s'intéresse seulement aux limites de l'écran, f que l'on considère celles de la figure; on peut alors couper (c) les parties qui tombent en dehors de la zône, comme si la zône constituait une fenêtre sur un espace illimité, ou arrêter (a) la génération d'un objet dès qu'un de ses éléments sort de la zône. Ces options conduisent à quatre possibilités, la cinquième (libre) signifiant que l'on n'effectue aucun contrôle sur les limites de zône, aux risques et périls de l'utilisateur. Les schémas de la

figure 4.2 montrent les résultats d'affichages du même objet pour chaque cas.

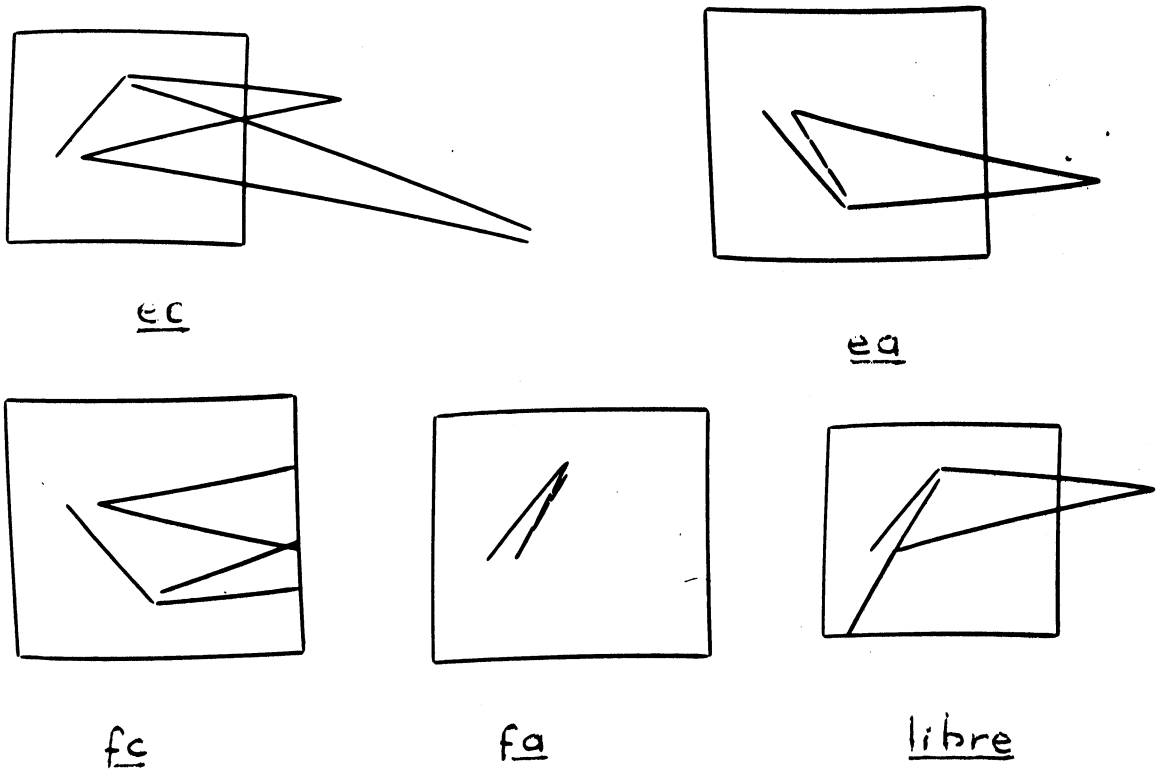


Figure 4.2 - Effet de l'option de coupage.

?(caractères (petits | grands) ?protégés)

Cette propriété concerne uniquement les objets de type texte qui seront générés dans la figure, pour indiquer la taille des caractères et leur protection éventuelle. Si l'on n'indique rien, les caractères sont de petite taille et protégés, c'est-à-dire que l'on ne peut pas les modifier au moyen du clavier alphanumérique. Si l'on indique la taille des caractères mais que l'on omet le mot-clé protégés, les caractères seront générés dans le mode non protége, et

ils pourront être modifiés; ceci permet l'introduction de données grâce au clavier alphanumérique.

?(taille entier-sans-signe)
Cette propriété indique en nombre d'octets la taille de la zône de mémoire de l'ordinateur dans laquelle est construite la partie de la liste d'affichage associée à la figure. Plus cette taille est grande, et moins on a de chances de voir la zône déborder, cas qui nécessite le transfert immédiat de la liste d'affichage déjà construite dans la mémoire d'entretien. La valeur indiquée peut être 128 ou un multiple de 256; l'option par défaut est 256, valeur généralement convenable pour des figures de complexité moyenne.

?(renomme nom-de-figure ?&(, nom-de-figure))
On peut associer à une figure jusqu'à 49 figures dites équivalentes, qui partagent la même zône de la mémoire d'entretien et possèdent exactement les mêmes propriétés; on peut donc avoir au total jugu'à 50 figures équivalentes entre elles, qui sont mutuellement incompatibles, et l'affichage de l'une d'elles implique l'effacement automatique de celle des figures de la famille qui est actuellement affichée (s'il y a lieu). La principale utilité de ce procédé est de permettre la production de dessins animés, chaque figure équivalente représentant un des états de la figure à animer. Les figures nommées après renomme sont déclarées par cette seule apparition, et sont traitées automatiquement toutes ensemble par les ordres afficher, inclure, omettre et fin figure qui concernent l'une quelconque d'entre elles, y compris celle qui est nommée en tête de la déclaration.

4.142 - Objets, séquences et arrêts.

Les objets, séquences et arrêts se déclarent tous de façon semblable:

(point | ligne | segment | texte | position | séquence | arrêt)
identificateur ?&(, identificateur) ?(figure nom-de-figure) ;

Tout objet, séquence ou arrêt déclaré doit appartenir à une figure précise; si cependant on n'indique pas le paramètre figure dans la déclaration, cela signifie qu'il faut prendre la dernière figure citée dans une déclaration d'objet, de séquence ou d'arrêt; si aucune déclaration de ce type ne précède, c'est une erreur.

L'objet point n'a pas besoin d'être précisé davantage; l'objet ligne correspond à un vecteur dont on ne donne que l'extrémité, l'origine étant implicite; l'objet segment correspond à un vecteur dont on donne l'origine et l'extrémité; l'objet texte correspond à

une chaîne de caractères dont l'origine (coordonnées du centre du premier caractère) peut ou non être implicite; l'objet position sert à placer le faisceau lumineux en un certain endroit de l'écran, avant de tracer des lignes ou des textes par exemple.

La nature et l'utilité de la séquence seront précisées au paragraphe 4.15 à propos des instructions début séquence et fin séquence; en ce qui concerne l'arrêt, son rôle sera établi au paragraphe 4.17 à propos de l'instruction mettre arrêt.

Voici pour terminer ce paragraphe 4.14 un exemple de déclaration graphiques:

```
figure DESSIN1 entrée entrel coupage fc sortie absolue zône -1,
     -1, 1, 1, écran -2, -2, 2, 2 données -100, -100, 100, 100
     renomme DESSIN2, DESSIN3, DESSIN4, DESSIN5, DESSIN6;
figure BARATINS entrée entabs zône 1, 1, 74, 10 écran 1, 1, 74,
     52 données 1, 10, 74, 1 caractères petits protégés;
points P1, P2, P3, P4 figure DESSIN1;
lignes L1, L2;
segments S1, S2, S3;
position POZ;
textes T1, T2, T3, T4, T5, T6, T7, T8, T9 figure BARATINS;
segment CADRE;
séquence SUITE figure DESSIN1;
arrêt HALTE;
```

4.15 - INSTRUCTIONS DE GENERATION.

4.151 - Initialisation et terminaison.

Tout objet généré doit faire partie d'une figure, et cette figure doit exister avant que l'on ne génère l'objet. Pour cela, la déclaration de la figure ne suffit pas, il faut encore l'avoir initialisée au moyen de l'instruction:

début figure nom-de-figure ?&(, nom-de-figure) ;

Cette instruction a pour effet de réserver dans la mémoire de l'ordinateur et dans la mémoire d'entretien du terminal les zônes nécessaires, et de mettre en place toutes les propriétés qui ont été indiquées dans la déclaration correspondante. C'est au moment où l'on exécute cette instruction que les variables qui peuvent éventuellement figurer dans la déclaration de figure doivent avoir une valeur.

Pour terminer l'utilisation d'une figure, on utilise l'instruction:

fin figure nom-de-figure ?&(, nom-de-figure) ;
Toutes les utilisations de la figure et des objets qui la composent doivent se trouver entre les instructions début figure et fin figure. Cette dernière instruction n'est en fait utile que si l'on veut récupérer la place occupée par les zônes indiquées plus haut, ou si l'on veut réinitialiser la figure. Pour modifier les propriétés de la figure pendant qu'elle existe, on peut utiliser l'instruction options, qui a exactement la même forme que la déclaration de figure, si ce n'est qu'elle ne contient pas les options taille et renomme.

4.152 - Désignation des objets.

Les objets peuvent être désignés de trois façons: la désignation par nom est celle qui est indiquée par une déclaration de l'objet, elle est unique en ce sens qu'un objet nommé est toujours du même type et fait toujours partie de la même figure, tout au long du programme; la désignation par numéro est donnée au moment de la génération de l'objet, et elle n'est pas forcément unique: deux objets ou plus peuvent avoir le même numéro, dans des figures différentes ou même dans la même figure; enfin, on peut désigner un objet à la fois par nom et par numéro, les deux procédés ayant leur intérêt.

Il est utile de désigner un objet pour pouvoir s'y référer ensuite, par exemple pour le modifier, le faire disparaître ou réapparaître; la désignation par nom évite de plus d'indiquer le type de l'objet dans l'instruction de génération, puisqu'il est déjà dans la déclaration. Dans le cas où l'on ne doit pas se référer à l'objet après l'avoir généré, il est inutile de le désigner d'une façon quelconque, et cela permet de gagner du temps et de la place.

Dans la désignation par nom, le nom de l'objet est une variable dans laquelle l'instruction de génération place une valeur qui permet de retrouver l'objet; si l'on génère plusieurs fois de suite un objet de même nom, son nom prendra successivement plusieurs valeurs, et ne conservera que la dernière: on ne pourra donc pas se référer aux objets précédents, mais ils n'en continueront pas moins d'exister en tant qu'objets sans désignation.

La désignation par numéro permet en quelque sorte de choisir le nom que l'on donne à un objet; quand par la suite on se réfère à l'objet de numéro x de la figure y, on demande en fait la consultation d'une table, caractéristique de la figure y, qui fait correspondre les noms et les numéros. Le premier élément trouvé dans cette table qui porte le numéro x est celui que l'on prend en

considération. Les exemples qui suivront dans le cours de ce chapitre montreront l'utilisation de tous ces moyens de désignation; la définition syntaxique de la désignation par numéro est la suivante:

nom-de-figure numéro expression-arithmétique

4.153 - Génération simple.

L'instruction de génération permet de construire dans la liste d'affichage les ordres nécessaires à l'affichage d'un objet; dans l'instruction de génération simple, chaque objet n'est formé que d'un seul élément. Le symbole fondamental est le symbole de génération, qui peut prendre l'une des quatre formes suivantes:

<== | <#= | <=# | <##

Les deux caractères qui suivent le caractère "<" représentent le mode des coordonnées, le premier pour les abscisses et le second pour les ordonnées; "=" indique des coordonnées absolues, et "#" des coordonnées relatives. Si ces deux modes ne correspondent pas à ce qui a été indiqué au moment de la déclaration de figure (option entrée), on effectue automatiquement les conversions nécessaires, au prix d'une légère perte de temps. Dans le cas de coordonnées relatives pour le premier objet généré dans une figure, il faut savoir qu'elles sont relatives au coin en bas à gauche de l'écran (coordonnées 0, 0 en unités-écran).

Il existe trois formes de l'instruction de génération simple, suivant le type de l'objet que l'on génère: pour un point, une ligne ou une position:

(nom-de-point | nom-de-ligne | nom-de-position | ((point | ligne | position) nom-de-figure)) symbole-de-génération E.A.[8] , E.A. ?(numéro E.A.) ?(inclus | omis) ;

pour un segment:

(nom-de-segment | (segment nom-de-figure)) symbole-de-génération E.A. , E.A. , E.A. , E.A. ?(numéro E.A.) ?(inclus | omis) ;

pour un texte:

(nom-de-texte | (texte nom-de-figure)) symbole-de-génération expression-de-chaîne ?(en E.A. , E.A.) ?(numéro E.A.) ?(inclus | omis) ;

La forme "type nom-de-figure" correspond à un objet qui n'a pas de nom; un objet inclus apparaît sur l'écran quand la figure qui le contient est affichée, un objet omis n'apparaît pas, mais les ordres

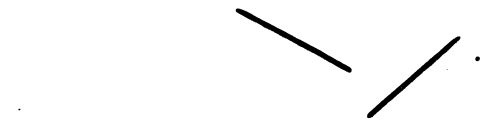
---

[8] E.A. est une abréviation pour expression-arithmétique.

qui le concernent existent cependant dans la liste d'affichage, et on pourra le faire apparaître ensuite au moyen d'une instruction inclure (voir le paragraphe 4.16). Le mode normal, valable si l'on n'indique rien, et le seul que l'on puisse indiquer pour une position ou un objet sans désignation, est inclus. Voici quelques exemples d'instructions de génération simple, correspondant aux déclarations du paragraphe 4.142; les groupes d'instructions 1 et 2 génèrent ce qui apparaît sur la figure 4.3; les groupes 3 et 4 génèrent ce qui apparaît sur la figure 4.4.

```
1 - POZ <== 0, 0;
    P2 <=# 10, 70 numéro 4 omis;
    L1 <## 55, 55 numéro 5;
    P1 <#= 10, 10;
    S3 <== 0, 0, - 70, + 40;

    pour I := 1 à 20;
    point DESSIN1 <## X1(I), Y1(I) numéro I + 200;
    finpour;

2 - position BARATINS <== 1, 1;
    T1 <== "PREMIER EXEMPLE";
    T2 <== "_______________" en 1, 2;

3 - pour I := 1 à 300 pas 3;
    AA := PI / 180.  * I;
    BB := PI / 180.  * (I + 1);
    segment DESSIN1 <== 50 * SIN(AA), 50 * COS(AA), 50 * SIN(BB), 50
        * COS(BB) numéro I;
    finpour;

4 - pour I := 1 à 8;
    AA := 45 * I * PI / 180.;
    POZ <== 100, 100;
    L2 <## 100 * COS(AA), 100 * SIN(AA);
    finpour;
```

On peut remarquer que dans ces exemples tous les nombres sont écrits comme des entiers, ce qui correspond d'ailleurs à ce qui a été indiqué dans les déclarations des figures (entrée entrel pour DESSIN1 et entrée entabs pour BARATINS). Cela n'est cependant pas nécessaire, et d'ailleurs l'instruction de génération segment DESSIN1 ... utilise les fonctions arithmétiques SIN et COS qui ont forcément un résultat réel. Le fait d'utiliser des coordonnées du type de celui qui est indiqué dan s la déclaration de figure permet simplement de gagner un peu de temps, comme le fait d'indiquer le mode prévu, absolu ou relatif.



PREMIER EXEMPLE

Figure 4.3 - Exemples de génération simple.

4.154 - Génération multiple.

L'instruction de génération multiple permet la production d'objets complexes, formés de plus d'un élément: suite de points, suite de lignes consécutives, suite de segments. Elle n'existe ni pour une position ni pour un texte. Sa forme générale est celle de l'instruction de génération simple, à deux différences près: immédiatement après le symbole de génération, on place:

expression-arithmetique fois

qui indique combien d'éléments doivent être générés. D'autre part, les expressions arithmétiques qui indiquent les coordonnées peuvent être remplacées par l'une des deux formes qui indiquent une suite de coordonnées; la première forme ajoute une valeur donnée à la coordonnée à chaque nouvel élément:

expression-arithmétique pas expression-arithmétique

la deuxième forme prend chaque coordonnée dans un tableau, en avançant à chaque nouvel élément d'un nombre de cases du tableau indiqué après le mot-clé pas et valant 1 s'il est omis:

tableau nom-de-tableau [expression-arithmetique] ?(pas expression-arithmétique)

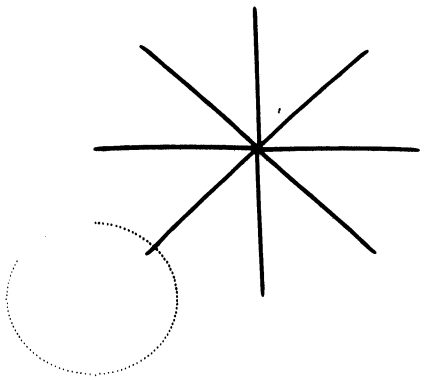


Figure 4.4 - Exemples de génération simple (suite).

On peut utiliser l'une ou l'autre de ces deux formes tout à fait librement, utiliser une forme différente pour chaque coordonnée, ou même utiliser pour une coordonnée la forme qui n'indique aucune variation, c'est-à-dire une simple expression arithmétique. Dans ce dernier cas, la coordonnée correspondante est fixe, ou plus exactement a la même valeur pour chaque élément (ce qui peut correspondre à des coordonnées variables en mode relatif). Les exemples ci-dessous montrent un certain nombre de combinaisons intéressantes; les groupes 5 et 6 produisent ce qui apparaît sur la figure 4.5, 7 et 8 la figure 4.6, 9 et 10 la figure 4.7.

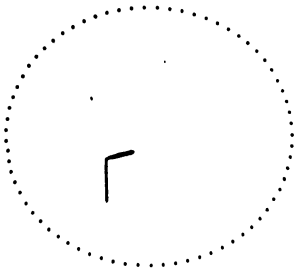


Figure 4.5 - Exemples de génération multiple.

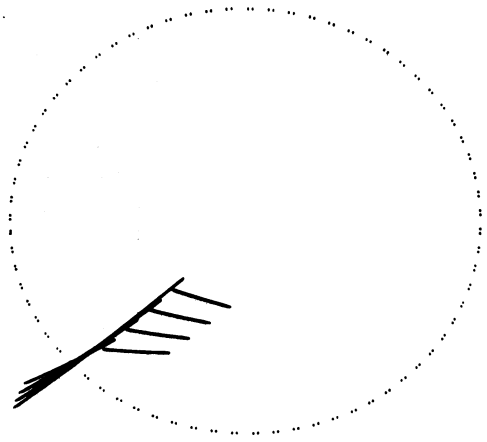


Figure 4.6 - Exemple de generation multiple (suite).

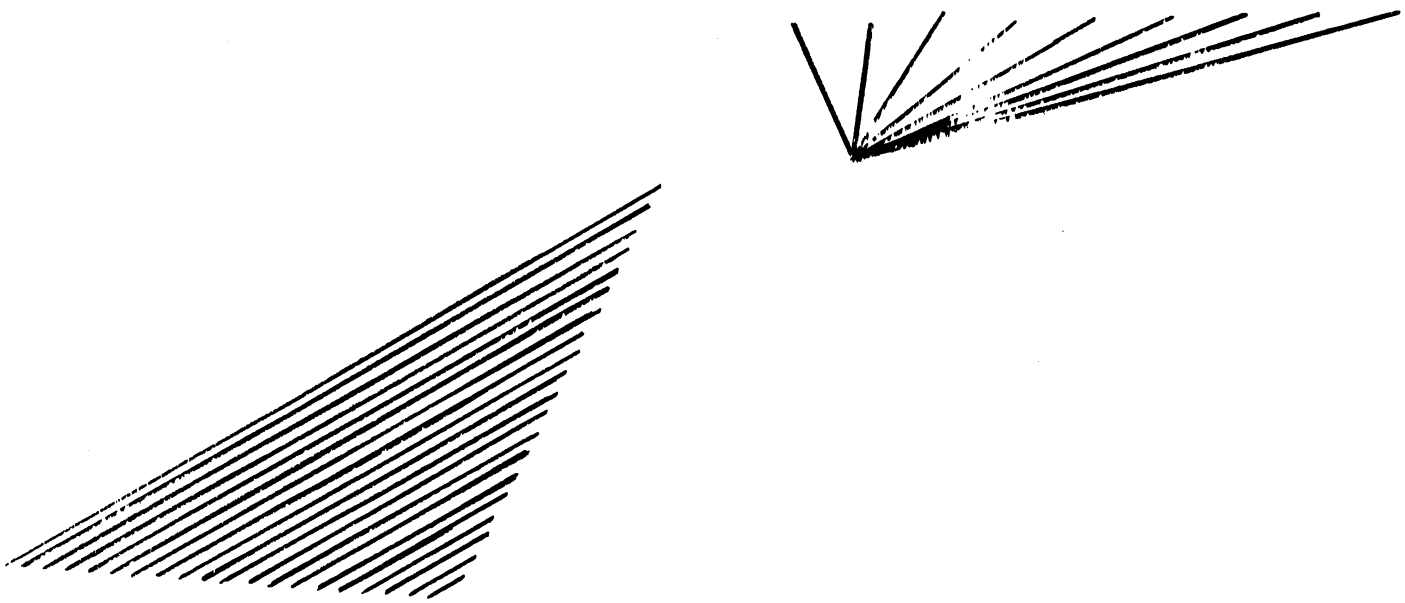


Figure 4.7 - Exemple de génération multiple (suite et fin).

```
5 - POX <== 10, 10;
     L1 <## 8 fois 20, 5 pas 5;

6 - R := 100;
    pour I := 1 a 72;
    AA := 5 * I * PI / 180.;
    X(I) := R * COS(AA);
    Y(I) := R * SIN(AA);
    finpour;
    P4 <== 72 fois tableau X(1), tableau Y(1) numéro 100;

7 - R := 200;
    pour I := 1 pas 2 a 287;
    AA := 2.5 * I * PI / 180.;
    X(I) := R * COS(AA);
    X(I + 1) := R * SIN(AA);
    finpour;
    P5 <==  144  fois tableau x(1) pas 2, tableau X(2) pas 2 numéro
         200;

8 - segment DESSIN1 <== 8 fois 50 pas 10, 50 pas 10,  tableau  X(1),
         tableau X(9);
```

```
9 - S3 <=# 8 fois 20, 10, tableau Y(1) pas 4, 10 pas 0.5;

10 - placer DESSIN1 en 0, 0;
     S2 <## 20 fois 1 pas 1, 2 pas 2, -2 pas -2, -1 pas -1;
```

4.155 - Séquences.

Une séquence est un ensemble d'objets de la même figure générés consécutivement; pour la définir, il faut encadrer les instructions de génération des objets voulus par une instruction:

début séquence nom-de-séquence ?(numéro E.A.) ?(inclus | omis)
;

et une instruction:

fin séquence nom-de-séquence ;

Si entre ces deux instructions figure la génération d'objets qui ne font pas partie de la figure dans laquelle a été declarée la séquence, ils ne font pas non plus partie de celle-ci. Le nom de la séquence peut être utilisé dans une instruction annuler (paragraphe 4.164) avant que l'instruction fin séquence n'ait été exécutée; on voit d'autre part qu'une séquence doit toujours avoir un nom.

Deux séquences de la même figure ne peuvent pas être imbriquées, à plus forte raison se chevaucher, mais il n'y a aucune interaction entre deux séquences de figures différentes; si les séquences S1 et S2 font partie de la figure F1, et les séquences S3 et S4 font partie de la figure F2, le schéma suivant est possible:

```
début séquence S1;
...
début séquence S3;
...
fin séquence S1;
début séquence S2;
...
fin séquence S3;
début séquence S4;
...
fin séquence S4;
...
fin séquence S2;
```

On trouvera dans le paragraphe 4.16 des exemples de définition et de manipulation de séquences.

4.16 - INSTRUCTIONS DE TRANSMISSION.

4.161 - Affichage.

Pour envoyer dans la mémoire d'entretien le fragment de liste d'affichage concernant une figure, et lancer la régénération, on utilise l'instruction:

afficher nom-de-figure ?&(, nom-de-figure) ;

On peut afficher plusieurs figures à la fois, sans aucun inconvénient; il est nécessaire d'afficher une figure pour que les objets qui en font partie apparaissent sur l'écran (sauf s'ils sont dans le mode omis); une fois la figure affichée, on peut lui faire subir certaines modifications qui ne nécessitent pas son réaffichage: cela sera réalisé au moyen des instructions d'omission, inclusion, mise à jour et annulation. Si au contraire on ajoute de nouveaux objets à une figure, il faut la réafficher pour ajouter ces objets à ceux qui apparaissent déjà dans la mémoire d'entretien sur l'écran. Les exemples de la fin de ce paragraphe 4.16 montreront quelques utilisations de l'instruction d'affichage. Rappelons que l'affichage de l'une des figures d'une famille de figures équivalentes implique la disparition de l'écran et de la mémoire d'entretien de celle des figures équivalentes de la famille qui apparaissait sur l'écran.

4.162 - Omission et inclusion.

Les instructions d'omission et d'inclusion s'écrivent de la même façon:

(omettre | inclure) (nom-de-figure | nom-de-séquence | nom-d'objet | nom-d'arrêt | désignation-par-numéro) ;

La désignation par numéro représente forcément un objet, un arrêt ou une séquence; ces deux instructions peuvent donc porter sur un objet ou un arrêt précis, sur une séquence c'est-à-dire une suite d'objets et d'arrêts, et enfin sur une figure c'est-à-dire sur tous les objets et les arrêts qui la composent. Un objet ou un arrêt non désigné est toujours inclus, et son état ne peut être modifié que par l'intermédiaire d'une séquence ou de la figure qui le contient. Il faut remarquer que mettre une figure dans le mode inclus ou omis n'est pas la même chose que de faire ce travail pour tous les objets de la figure; si l'on omet un à un des éléments d'une figure, on ne pourra pas les faire réapparaître en se contentant de demander d'inclure la figure.

Un objet omis n'apparaît pas sur l'écran, bien qu'il soit présent dans la mémoire d'entretien; si c'est un point, une ligne ou un segment, le faisceau lumineux se déplace cependant comme si l'objet apparaissait, alors que ce n'est pas le cas si l'objet est une position ou un texte; un arrêt omis est ineffectif. Si l'on demande

d'omettre une figure, une séquence, un objet ou un arrêt qui l'est déjà, il ne se passe rien.

Un objet inclus apparaît sur l'écran si c'est un point, une ligne, un segment ou un texte; une position incluse déplace le faisceau lumineux; un arrêt inclus provoque une interruption quand le cycle de régénération passe dessus (voir le paragraphe 4.176).

Voici des exemples d'instructions d'omission et d'inclusion utilisant les déclarations et les instructions de génération des paragraphes précédents (des exemples plus justifiés apparaîtront en fin de ce paragraphe 4.16):

```
inclure P2;
omettre L1;
omettre POZ;
inclure SUITE;
omettre DESSIN1 numéro 100;
pour I := 1 a 20;
omettre DESSIN1 numéro 200 + I;
finpour;
omettre BARATINS;
```

4.163 - Mise à jour.

L'instruction de mise à jour sert à modifier un objet précis. Elle a exactement la même forme qu'une instruction de génération simple ou multiple, si ce n'est que l'option (inclus | omis) disparaît, et que le symbole de génération est remplacé par un symbole de mise à jour:

<==> | <=#> | <#=> | <##>

Cette instruction modifie directement dans la mémoire d'entretien les ordres de génération de l'objet indiqué; ceci impose donc un certain nombre de restrictions sur la nature de la mise à jour:

- l'objet mis à jour doit être de même nature que l'ancien;
- le type des ordres générés (propriété sortie de la déclaration de figure ou de l'instruction options) doit être le même;
- la taille et le mode de protection des caractères doivent être les mêmes;
- le mode inclus ou omis ne peut être changé par l'instruction de mise à jour elle-même;
- surtout les ordres générés doivent avoir un encombrement inférieur ou égal à celui de l'objet initial; cette dernière restriction n'est pas particulièrement simple à respecter dans le cas d'ordres incrémentiels, où l'on contrôle mal l'encombrement d'un objet.

Si l'on ne peut pas respecter l'une quelconque de ces restrictions (sauf celle sur le mode inclus ou omis), il faut utiliser l'instruction d'annulation (voir le paragraphe 4.164) et recommencer la génération de la partie de figure ainsi effacée.

Avant d'effectuer une mise à jour, il faut indiquer au système quelle est la position de départ de l'objet, c'est-à-dire les coordonnées actuelles du faisceau lumineux; on peut le faire au moyen de l'instruction placer, qui vient mettre le faisceau lumineux à l'endroit voulu, ou de l'instruction situer, qui indique au système la position, connue par un autre moyen, de ce faisceau; ces deux instructions ont la même forme:

(placer | situer) nom-de-figure en E.A. , E.A. ;

Pour l'instruction placer, les coordonnées sont toujours absolues, quel que soit le mode indiqué dans la déclaration de figure (propriété données).

Si, après avoir mis à jour des objets d'une figure, on veut générer de nouveaux objets dans la même figure, on doit là aussi utiliser l'instruction placer ou l'instruction situer, pour déterminer la position du faisceau lumineux avant cete nouvelle génération.

L'objet à mettre à jour doit exister et être désigné; la forme "type nom-de-figure ..." de l'instruction nécessite donc que l'on utilise un numéro; si l'on désigne un objet par nom et que l'on indique aussi un numéro, ce dernier n'est pas utilisé comme référence, mais vient remplacer le numéro qu'avait l'objet précédemment, ou le supprimer si l'on indique une valeur nulle.

Les instructions suivantes servent à mettre à jour certains des objets générés dans les exemples du paragraphe 4.15:

```
placer DESSIN1 en X1(20), Y1(20);
pour I := 1 à 100;
AA := 250 * PI / I;
BB := 300 * PI / (I + 1);
segment DESSIN1 <==> SIN(AA), COS(AA), SIN(BB), COS(BB)  numéro
     I;
finpour;

reprise BARATINS en 1, 1;
T2 <==> "- - - - - - - - -" en 4, 2;

AA := abscisse DESSIN1;
BB := ordonnée DESSIN1;
P5  <==> 144 fois tableau X(2) pas 2, tableau X(1) pas 2 numéro
```

```
    201;
POZ <==> 0, 0;
S2 <==> 10 fois 4, 2, -2, -4;
situer DESSIN1 en AA, BB;
```

4.164 - Annulation.

L'instruction d'annulation a la même forme que les instructions d'omission et d'inclusion:

annuler (nom-de-figure | nom-d'objet | nom-de-séquence | nom-d'arrêt | désignation-par-numéro) ;

Elle a pour effet de supprimer de la liste d'affichage, non seulement ce qui est indiqué par son opérande, mais encore tout ce qui suit dans la même figure, c'est-à-dire tout ce qui a été généré après. Il s'agit donc d'une annulation à partir de l'endroit indiqué et jusqu'à la fin de la figure dont cet endroit fait partie; dans le cas ou l'opérande est un nom de figure, il s'agit évidemment d'annuler toute la figure. La place occupée dans la mémoire d'entretien par ce qui a été supprimé est récupérée. Si l'on veut générer de nouveaux objets dans la figure dans laquelle se fait la suppression, il faut commencer par une instruction placer ou situer pour indiquer la position de départ du faisceau lumineux.

4.17 - INSTRUCTIONS DE COMMUNICATION.

4.171 - Sources d'interruptions et niveaux.

La communication entre l'utilisateur humain et son programme se fait au moyen de dispositifs qui provoquent des interruptions; celles-ci sont emmagasinées par le système si le programme a prévu de les accepter, mais elles ne sont signalées au programme que quand celui-ci le demande (au moyen de l'instruction état); il n'y a donc aucun traitement asynchrone à effectuer, ce qui simplifie d'autant la programmation.

Les sources d'interruptions sont au nombre de trente-quatre, et numérotées comme suit:

- trente et une touches du clavier de fonctions, numérotées de 1 à 31 (la touche zéro n'est pas utilisable, son usage étant réservé au système);
- la touche "fin de ligne" du clavier alphanumérique, numérotée 32;
- une détection par le pointeur optique, numérotée 34;
- un arrêt programme (généré par une instruction mettre arrêt), numéroté 35.

Il faut donc remarquer qu'il n'existe pas de source d'interruption de numéro 33.

Le programme doit pouvoir effectuer les opérations suivantes:
- indiquer quelles sources d'interruptions il veut reconnaître, en ignorant les autres;
- modifier cet état de choses en autorisant de nouvelles sources et en en interdisant d'autres;
- demander s'il s'est produit une interruption parmi un ensemble de sources indiqué;
- obtenir des renseignements supplémentaires sur une interruption;
- attendre qu'une interruption parmi plusieurs se produise.

Tout ceci se fait par l'intermédiaire d'un même outil, le niveau d'interruptions; on peut le considérer comme un ensemble de trente-quatre files d'attente dont certaines peuvent être ouvertes et d'autres fermées: on peut donc ouvrir et fermer ces files d'attente, ce qui revient à autoriser ou interdire les sources d'interruptions correspondantes, et les examiner pour savoir quelles interruptions se sont produites et dans quelles conditions. Quand arrive une interruption, on regarde si la file d'attente correspondante est ouverte; si non, l'interruption est ignorée; si oui, on ajoute à la fin de la file toutes les informations concernant ce qui s'est passé; ces informations sont extraites des files dans l'ordre dans lequel elles se sont produites.

On n'a besoin normalement que d'un seul niveau, que l'on doit simplement déclarer et initialiser avant de pouvoir l'utiliser. La déclaration a la forme:

niveau identificateur ?&(, identificateur) ;

L'instruction d'initialisation correspondante est:

début niveau nom-de-niveau ?(libre ¦ gardé) ;

Dans certaines applications particulières, on peut avoir besoin de plusieurs niveaux à la fois; un seul est actif à un moment donné, celui qui a été initialisé en dernier; c'est vers lui que sont aiguillées toutes les interruptions qui se produisent, à condition que les portes correspondantes soient ouvertes, mais on peut consulter tous les autres niveaux initialisés avant celui-là; quand on termine l'utilisation d'un niveau au moyen de l'instruction:

fin niveau nom-de-niveau ?(libre ¦ gardé) ;

toutes les interruptions qui pouvaient rester en attente sont perdues, et c'est le niveau précédent qui redevient actif. Il y a donc empilage et dépilage des niveaux. Dans l'instruction début niveau, l'option libre signifie que l'on peut interroger le niveau à tout moment, et que toute interruption signalée à l'utilisateur est

alors retirée de sa file d'attente; l'option gardé signifie au contraire que l'information correspondant à l'interruption n'est retirée de sa file d'attente que si le niveau est actif; ceci permet d'interroger un niveau inactif sans perturber l'information qu'il contient. Dans l'instruction fin niveau, l'option libre indique que l'on veut terminer le niveau indiqué et tous ceux qui ont été initialisés après lui, alors que l'option gardé signifie que l'on ne veut terminer que les niveaux qui ont été initialisés après celui qui est indiqué; ce dernier devient donc actif. Dans les deux cas, l'option prise par défaut est libre, qui n'a d'ailleurs de signification que si l'on utilise plusieurs niveaux simultanément.

L'existence simultanée de plusieurs niveaux est intéressante dans deux situations: définition de commandes hiérarchisées ou existence de sous-programmes. La première situation se présente de façon importante dans le système Eulalie, qui sera étudié au chapitre 6, et l'écriture de ce dernier en langage Euphémie dans l'annexe 4 montre bien cette utilisation. La deuxième situation ne peut se présenter actuellement, puisque le langage Euphémie ne contient pas encore la notion de sous-programme. On peut cependant indiquer rapidement que la définition d'un niveau local à un sous-programme permet de ne pas perturber le programme principal, que l'on peut ne pas connaître.

4.172 - Traitement des sources d'interruptions.

Les instructions traiter et ignorer permettent respectivement d'ouvrir et de fermer les portes d'entrée des files d'attente, qui indiquent quelles sources on veut reconnaître et lesquelles on veut ignorer. Elles ont la même forme:

(traiter | ignorer) nom-de-niveau [E.A. ?&(, E.A.)] ;

Les expressions arithmétiques entre les parenthèses indiquent les numéros des sources d'interruptions concernées, suivant la correspondance indiquée au paragraphe précédent; pour indiquer un éventail important de valeurs, on utilise une valeur négative: (1, -5) a la même signification que (1, 2, 3, 4, 5). Le niveau indiqué n'a pas à être actif; si l'on demande de traiter une source qui était déjà autorisée, la file d'attente correspondante est vidée des interruptions qui pouvaient y être; si l'on demande d'ignorer une source qui était auparavant autorisée, la file d'attente correspondante n'est pas perturbée et pourra toujours être consultée.

La consultation d'un niveau donné se fait au moyen de l'instruction:

état nom-de-niveau [E.A. ?&(, E.A.)] ?(réponse variable)
?(résultat nom-de-tableau) ?(retour | attente) ;

La liste d'expressions arithmétiques entre parenthèses indique quelles sources d'interruptions on désire interroger, avec les mêmes conventions que pour les instructions traiter et ignorer. La variable indiquée après réponse reçoit après exécution de l'instruction état le numéro de la première en date des interruptions qui se sont produites parmi les sources indiquées et relativement à ce niveau; le tableau indiqué après résultat doit avoir au moins dix éléments, et reçoit des informations supplémentaires qui dépendent de la nature de l'interruption et seront précisées dans les paragraphes suivants. L'option attente signifie que si aucune interruption n'est en attente parmi les sources indiquées, on doit attendre qu'il s'en produise une; l'option retour signifie au contraire que dans ce cas on n'attend pas, mais qu'alors la variable indiquée après réponse prend la valeur zéro. Si l'on n'attend qu'une interruption et que l'on n'a pas besoin d'informations supplémentaires, on n'a besoin d'aucune des parties facultatives de l'instruction, l'option par défaut étant attente.

Nous allons voir maintenant l'utilisation de ces instructions de traitement pour chacune des catégories d'interruptions.

4.173 - Clavier de fonctions.

Le clavier de fonctions est le dispositif dont l'utilisation est la plus simple: si l'on désire passer à une certaine phase d'un travail après un appui sur la touche numéro 1, il suffit, après que le niveau SOURCE ait été déclaré et initialisé, d'une instruction:

    traiter SOURCE(1);

au moment à partir duquel on accepte cette interruption, et d'une instruction:

    état SOURCE(1);

juste avant le début de la nouvelle phase.

On peut aussi manipuler les indicateurs lumineux qui figurent sous les touches de fonctions, par exemple pour n'éclairer que ceux qui correspondent à des interruptions autorisées. On dispose pour cela de l'instruction:

    touches ?nom-de-niveau (défaut | éteint | normal | ([E.A. ?&(, E.A.)]));

Il existe un état global des indicateurs, et éventuellement un état par niveau; la disposition initiale de l'état global est l'option éteint, ou tous les indicateurs sont éteints; on peut la modifier au moyen de la forme de l'instruction touches qui n'indique pas de nom de niveau.

L'option normal allume les indicateurs qui correspondent aux touches reconnues par le niveau actif; l'option ou l'on place une liste d'expressions arithmétiques entre parenthèses donne la lsite des touches dont les indicateurs doivent être allumés, avec les mêmes conventions que précedemment; l'option défaut n'a de sens que pour un niveau, et lui donne la même disposition qne l'état global; c'est évidemment l'option par défaut.

Quand on utilise le paramètre résultat de l'instruction état, seul le premier mot du tableau a une signification: il contient le numéro associé au cache placé sur le clavier de fonctions; dans le cas ou il n'y a pas de cache, ce numéro est 255; il est nul si le cache ne porte aucune encoche.

L'exemple suivant montre quelques utilisations des touches de fonctions; la figure 4.8 montre les états successifs du clavier de fonctions.

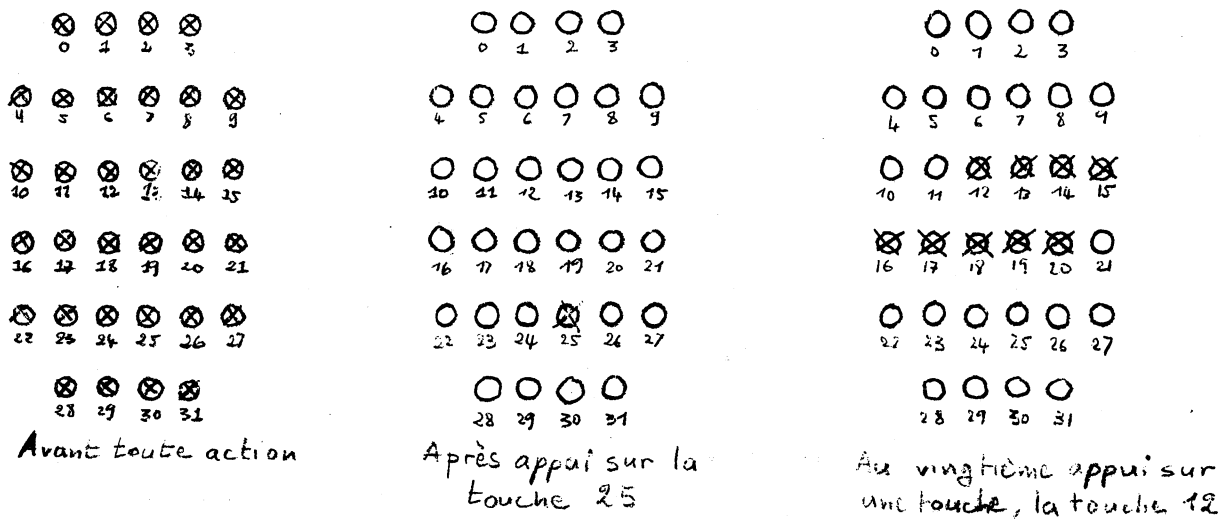


Figure 4.8 - Etats du clavier de fonctions.

```
niveau SOURCE;
entiers I, J;
tableau entier HOP(10);
début niveau SOURCE;
traiter SOURCE(1, -31);
```

```
état SOURCE(1, -31) résultat HOP;
si HOP(1) ¬= 255;
ENCORE:  touches normal;
état SOURCE (1, -31) réponse I retour;
si I ¬= 0;
allerà HEHE finsi;
touches éteint;
allerà ENCORE finsi;
HEHE:  état SOURCE(1, -31) réponse I;
si I ¬= 31;
touches SOURCE(I);
allerà HEHE finsi;
touches SOURCE normal;
J := 1;
pour I := 1 a 31;
traiter SOURCE(J, -I);
état SOURCE(J, -I) réponse J;
finpour;
```

4.174 - Clavier alphanumérique.

L'introduction de texte au moyen du clavier alphanumérique nécessite plusieurs opérations successives:

- Aficher sur l'écran un texte qui recevra celui qui doit être frappé; ce texte peut être quelconque, même si l'on utilise le plus souvent des blancs, et il peut être de petite ou grande taille, mais il doit être en mode non protégé;
- Placer en tête de ce texte un curseur, qui permet physiquement la frappe des caractèrs;
- Attendre une interruption qui signale que le texte a été frappé; cette interruption peut être n'importe laquelle, même si l'on choisit le plus souvent la touche "fin de ligne" du clavier alphanumérique (numéro 32);
- Lire dans la mémoire d'entretien le texte qui y a été placé et qui a dû être modifie;
- Enlever le curseur; cette dernière opération peut se placer avant la précédente, ou ne pas avoir lieu.

Il nous faut donc introduire trois nouvelles instructions qui permettront de placer et enlever le curseur et de lire le contenu de la mémoire d'entretien. On met en place le curseur au moyen de l'instruction:

```
mettre curseur (nom-de-texte | désignation-par-numéro | nom-de-
      séquence) ?(en E.A.)  ;
```

La désignation par numéro doit représenter un texte ou une séquence; placer un curseur dans une séquence indique en fait qu'il faut le

placer en tête du premier texte de cette séquence, ce qui évite d'avoir à donner un nom ou un numéro à ce dernier. L'expression arithmétique après en donne le numéro du caractère du texte sous lequel on veut placer le curseur, au cas où ce n'est pas le premier. S'il y a déjà un curseur sur l'écran, on l'enlève avant de placer le nouveau; il vaut mieux cependant enlever le curser explicitement des que l'on ne veut plus que l'utilisateur frappe de texte, pour éviter que ce qui apparaît sur l'écran ne reflète pas ce qui a été pris en compte par le programme. On le fait au moyen de l'instruction:

enlever curseur nom-de-figure;

La dernière instruction dont on a besoin est donc:

lire E.A. (données | caractères) dans (nom-de-tableau | nom-de-chaîne) ?(jusquà (fin | curseur)) ?(depuis (nom-d'objet | nom-d'arrêt | E.A.)) ?(à (nom-d'objet | nom-d'arrêt | E.A.)) ?(figure nom-de-figure) ?(réponse variable) ;

Bien que cette instruction ne soit généralement utilisée que pour lire ce qui a été frappé au clavier (option caractères), elle peut cependant servir à lire tout ce qui se trouve dans la mémoire d'entretien, ordres de génération et données, ce que l'on indique par l'option données. L'expression arithmétique après lire indique le nombre maximum d'octets à lire, qu'il s'agisse de caractères ou de données; ces octets sont rangés tels quels dans le tableau ou la chaine indiqués après dans, et qui doit être de taille suffisante, sachant qu'il tient quatre octets dans un mot de tableau entier ou réel.

La lecture commence en tête de l'objet indiqué après depuis, ou en tête de la figure indiquée après figure si depuis n'y est pas, ou en tête de la figure dont fait partie l'objet indiqué après à si ni figure ni depuis ne sont présents; il faut donc qu'il y ait dans l'instruction soit un nom d'objet, soit un nom de figure, et au moins l'une des trois options depuis, à et figure; l'expression arithmétique qui peut figurer après depuis ou à est un numéro d'objet de la figure indiquée après figure; la lecture ne se fait qu'à l'intérieur d'une seule figure. Commencer la lecture en tête d'un objet signifie sur l'ordre de génération si l'on lit des données, et sur le premier caractère après cet ordre si l'on lit des caractères.

La lecture peut s'arrêter quand le nombre d'octets indique a ete transmis, ou quand on arrive à la fin de l'objet après depuis si à est absent, ou quand on arrive à la fin de l'objet indique après à, ou quand on arrive à la fin de la figure si ni depuis ni à ne sont présents, ou quand enfin quand on trouve le curseur; on s'arrête sur

celui des évènements précédents qui se produit le premier; avec l'option jusquà fin, on ne tient pas compte du curseur; c'est l'option par défaut en cas de lecture de données, alors que c'est jusquà curseur en cas de lecture de caractères.

La variable après réponse reçoit si elle est présente le nombre d'octets lus s'il est inférieur au nombre indiqué, avec le signe moins si l'on a été arrêté par le curseur et le signe plus autrement; si l'on a lu la quantité indiquée, la valeur qu'elle reçoit est nulle.

Les exemples ci-dessous doivent clarifier un peu l'écriture assez complexe de cette derrnière instruction. Le premier exemple montre les opérations nécessaires pour lire dans une chaîne LIRE un texte de 72 caractères au maximum:

```
figure TEXTE caractères petits entrée entabs;
chaînes 72 caractères BLABLA := "", LIRE;
niveau ARRET;
début figre TEXTE;
début niveau ARRET;
traiter ARRET(32);
texte TEXTE <== BLABLA en 0, 1024 numéro 1;
afficher TEXTE;
mettre curseur TEXTE numéro 1;
etat ARRET(32);
enlever curseur TEXTE;
lire 72 caractères dans LIRE figure TEXTE;
```

Le deuxième exemple montre quelques utilisations plus complexes; le programme pose une question (texte BLABLA) et attend une réponse écrite en gros caractères; il se transfère à l'étiquette COURT si le curseur est resté avant la fin de la zone réservée à la réponse. Plus loin, il affiche 28 lignes du caractère _, en mode grand non protégé; l'utilisateur peut alors placer du texte successivement dans chacune de ces lignes, le curseur progressant quand il appuie sur la touche "fin de ligne" du clavier alphanumérique; quand l'utilisateur a traité les 28 lignes, ou qu'il appuie sur la touche de fonction 31, le programme relit l'ensemble du texte pour lui faire subir un certain traitement, qui n'est pas précisé.

```
figure LIGNES entrée entabs caractères grands protégés données
     1, 35, 52, 1;
textes QUESTION, REPONSE, MESSAGE figure LIGNES;
tableau entier TABLE(442);
niveau ATTENTE;
entiers I, J;
chaînes 52 caractères BLABLA, TEXTE, NUL;
```

```
début figure LIGNES;
début niveau ATTENTE;
...
QUESTION <== BLABLA en 1, 1;
options LIGNES caractères grands; REPOINSE <== NUL en 1, 2;
afficher LIGNES;
mettre curseur REPONSE;
traiter ATTENTE(1, -31);
état ATTENTE(1, -31) réponse I;
lire 52 caractères dans TEXTE depuis REPONSE réponse J;
enlever curseur LIGNES;
si J < 0;
J := -K;
allerà COURT finsi;
si J = 0;
J := 52;
finsi;
...
pour I := 8 a 35;
texte LIGNES <== "_____
-__" en 1, I numéro I;
finpour;
afficher LIGNES;
traiter ATTENTE (32);
pour I := 8 à 35;
mettre curseur LIGNE numéro I;
état ATTENTE(31, 32) réponse J;
si J= 31;
allerà HOP finsi;
finpour;
HOP:   lire  28 * 52 caractères dans TABLE depuis 8 à 35 figure
     LIGNES réponse I;
```

4.175 - Pointeur optique.

Le pointeur optique peut être utilisé pour deux usages: désigner un objet ou indiquer un emplacement de l'écran; le deuxième usage est le plus limité, c'est aussi le plus simple: on utilise pour cela l'instruction:

lirepointeur nom-de-figure dans variable , variable ;

Le nom de figure ne sert qu'à indiquer le type, la base et l'échelle des coordonnées à fournir; celles-ci sont rangées par l'instruction dans les deux variables indiquées. La recherche de la position du pointeur est faite au moyen d'un caractère (la lettre W) qui balaie tout l'écran jusqu'au moment où une interruption signale que le pointeur l'a "vu". L'inconvénient majeur de ce procédé est qu'il

fait disparaître de l'écran l'image courante donc les repères qui peuvent donner une signification à la position du pointeur; ceci impose donc que l'on effectue les opérations dans l'ordre suivant:
- l'opérateur place le pointeur dans la position désirée (de la main droite);
- il indique au programme qu'il est prêt en appuyant (de la main gauche) sur une touche de fonction prévue à cet effet;
- le programme effectue l'instruction lirepointeur: l'image courante disparaît, le caractère de recherche balaie l'écran jusqu'à l'interruption, l'image reparaît;
- le programme peut alors utiliser les coordonnées obtenues pour l'usage prévu auparavant.

L'exemple ci-dessous montre l'utilisation de lirepointeur pour la construction d'un polygone fermé:

```
figure DESSIN;
réels X, Y, U, V;
entiers I,J;
niveau HOP;
début figure DESSIN;
début niveau HOP;
traiter HOP(1, 2, 3);
état HOP(1);
lirepointeur DESSIN dans X, Y;
position DESSIN <== X, Y numéro 1;
J := 1;
ENCORE: état HOP(1, 2) réponse I;
si I = 1;
lirepointeur DESSIN dans U, V;
J := J + 1;
ligne DESSIN <== U, V numéro J;
afficher DESSIN;
allerà ENCORE finsi;
ligne DESSIN <== X, Y numéro J + 1;
afficher DESSIN;
état HOP(3);
fingraphique;
```

Sil'on dispose de la tablette Sylvania, l'utilisation de lirestylet (qui s'écrit exactement comme lirepointeur) est nettement plus simple: la lecture des coordonnées du stylet est faite instantanément, sans que l'image disparaisse de l'écran; il faut cependant procéder exactement de la même façon qu'avec le pointeur optique, car si l'on omet de placer le stylet en contact avec la tablette avant que le programme n'exécute l'instruction lirestylet, on obtient des coordonnées nulles. L'exemple ci-dessous montre l'uti-

lisation de cette instruction pour un dessin à main levée, dont on peut voir le résultat sur la figure 4.9. Le trace commence quand on appuie sur la touche 1 ou la touche 2, et s'arrête quand on lève la main. La touche 2 efface tout le dessin, et la touche 31 termine le travail; les coordonnées du stylet sont lues à la fin de chaque régénération, grâce à l'arrêt programmé mis en fin de figure (voir le paragraphe 4.176).

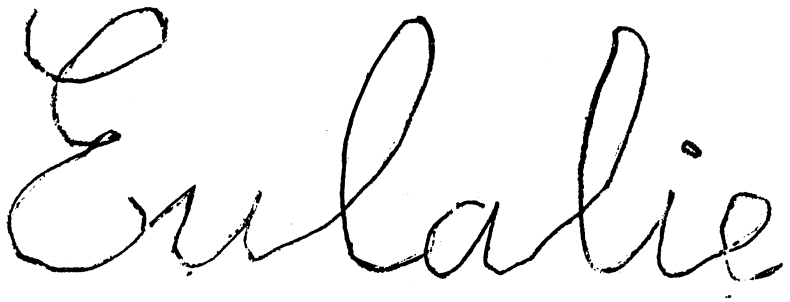


Figure 4.9 - Exemple d'utilisation de la tablette Sylvania.

```
réels X, Y;
entier I:
figure F;
arrêt S figure F;
niveau H;
traiter H(1, 2, 31, 35);
E1:  état H(1, 2, 31) réponse I;
si I = 2;
annuler F;
finsi;
si I = 31;
allerà FIN finsi;
lirestylet F dans S, Y;
placer F en X, Y;
E2:  mettre arrêt S;
```

```
afficher F;
état H(35);
annuler S;
lirestylet F dans X, Y;
si X + Y > 1;
ligne F <== X, Y;
allerà E2 finsi;
afficher F;
allerà E1;
FIN: fingraphique;
```

L'utilisation du pointeur optique comme outil de désignation est plus puissante et plus perfectionnée que la précédente; pour que la désignation d'un objet soit prise en compte, il faut que deux conditions soient satisfaites: l'interruption de numéro 34 doit être autorisée, et la figure dont fait partie l'objet doit être rendue "sensible aux détections" au moyen de l'instruction:

pointeur nom-de-figure (traité | ignoré) ;

Toutes les figures sont non sensibles (pointeur ignoré) au moment de leur initialisation, ce qui permet d'éviter les interruptions parasites.

Si l'on désigne avec le pointeur l'intersection de deux objets qui font partie de deux figures différentes toutes deux sensibles aux détections, l'interruption prise en compte concerne l'objet qui apparaît le premier dans le cyle de régénération; cet ordre dépend normalement de l'ordre dans lequel les figures ont été initialisées. Si l'on désire le modifier, on peut utiliser l'instruction:

ordonner nom-de-figure ?&(, nom-de-figure) ;

Les figures nommées dans cette instruction apparaîtront en premier dans le cycle de régénération, et dans l'ordre indiqué; les figures qui ne sont pas nommées apparaîtront après, et dans un ordre inchangé: si l'ordre initial est A, B, C, D l'instruction

ordonner C, A;

donne l'ordre C, A, B, D.

Après une interruption due à une détection par le pointeur optique, on trouve de nombreux renseignements sur l'objet désigné dans le tableau résultat de l'instruction état:

- mot 1: nom de la figure dont fait partie l'objet;
- mot 2: nom de l'objet (*);
- mot 3: si la détection a eu lieu sur un caractère d'un texte ayant un nom, numéro d'ordre de ce caractère dans le texte; sinon, numéro d'ordre de l'octet ayant généré l'élément detecté, à partir du début de l'objet si celui-ci a un nom, sinon à partir du début de la figure;

- mot 4: numéro de l'objet (*);
- mot 5: nom de la séquence dont fait partie l'objet (*);
- mot 6: numéro d'ordre de l'octet générateur à partir du début de la séquence (*);
- mot 7: numéro de la séquence (*);
- mots 8 et 9: coordonnées absolues du point, du centre du caractère ou de l'extrémité du vecteur détecté, dans le système de coordonnées de la figure;
- mot 10: code EBCDIC, cadré à gauche, du caractère détecté (*).

Dans tous les cas marqués (*), l'information est nulle si elle ne peut être donnée (par exemple l'objet n'a pas de nom ou ne fait pas partie d'une séquence). Les coordonnées trouvées dans les mots 8 et 9 peuvent différer légèrement de celles qui ont été fournies au moment de la génération, car elles sont passées en général par plusieurs conversions successives.

Les exemples ci-dessous montrent des utilisations de l'interruption due au pointeur optique, et constituent en même temps des programmes complets qui utilisent une bonne partie des ressources du langage Euphémie.

Le premier exemple est tiré de la brochure [IBM2]; il montre simplement quelques manipulations élémentaires, dont on voit les résultats sur la figure 4.10.

```
figure CERCLE caractères grands protégés;
figure ETIQUE;
figure ROND;
niveau HALTE;
réels R, PI := 3.141593, D := 1200;
tableau réel TABLE (16), CX, CY (182), entier HOP (10);
entiers I, J, K;
début figures CERCLE, ETIQUE, ROND;
début niveau HALTE;
traiter HALTE (1, -3, 34);
touches normal;
pointeur CERCLE traité;
pointeur ETIQUE traité;
pointeur ROND traité;
texte CERCLE <== "X" en 2047, 2047 numéro -1;
pour I := 1 à 8;
R := 45 * I * PI / 180.;
TABLE (2 * I - 1) := 2047 + D * COS (R);
TABLE (2 * I) := 2047 + D * SIN (R); finpour;
options CERCLE caractères petits protégés;
pour I := 1 à 8;
```

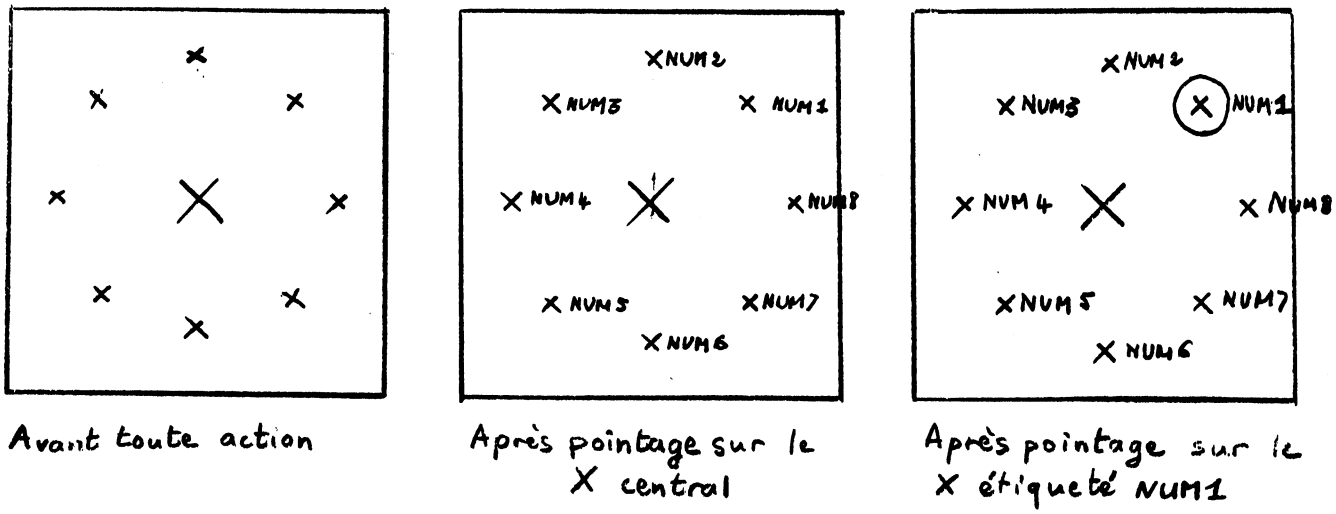


Figure 4.10 - Exemple d'utilisation du pointeur optique.

```
texte CERCLE <== "X" en TABLE (2 * I - 1), TABLE (2 * I) numéro
     I;
texte ETIQUE <== "NUMERO" || I en TABLE (2 * I - 1) + 300,
     TABLE (2 * I) numéro I;
finpour;
HEP:  pour I := 1 a 24;
R := U * PI / 12;
pour J := 1 a 8;
K := 8 * I + J - 8;
CX (K) := TABLE (2 * J - 1) + D * COS (R);
CY (K) := TABLE (2 * J) + D * SIN (R);
finpour HEP;
pour I := 1 a 8;
position ROND <== TABLE (2 * I - 1) + 200, TABLE (2 * I);
ligne ROND <== 24 fois tableau CX (3 * I - 7) pas 8, tableau CY
     (8 * I - 7) pas 8 numéro I;
finpour;
omettre ETIQUE;
afficher CERCLE, ETIQUE, ROND;
ENCORE:  état HALTE (1, -3, 34) réponse I résultat HOP;
E1:  si I = 34;
E2:  si HOP (1) = CERCLE;
```

```
E2:  si HOP (4) < 0;
inclure ETIQUE;
sinon E3;
inclure ROND numéro HOP (4);
sinon E2;
E4:  si HOP (1) = ETIQUE;
omettre ETIQUE;
sinon E4;
E5:  si HOP (1) = ROND;
omettre ROND numéro HOP (4);
sinon E1;
E6:  si I = 2;
pour J := 1 à 8;
omettre ROND numéro J;
finpour;
sinon E6;
E7:  si I = 3;
pour J := 1 à 8;
inclure ROND numéro J;
finpour;
finsi E1;
E8:  si I ¬= 1;
allerà ENCORE finsi E8;
```

Le deuxième exemple permet l'édition d'un texte d'une façon assez élémentaire; les commandes possibles apparaissent sur l'écran comme un "menu", dont on peut pointer les éléments. Certaines commandes (effacer, corriger, supprimer et remplacer) nécessitent le pointage sur un ou plusieurs caractères pour indiquer les opérandes de l'opération. Cet exemple suppose l'existence des instructions lirefichier et ecrirefichier, qui n'existent pas encore dans le langage Euphémie. La figure 4.11 montre quelques exemples d'état de l'écran.

```
figure MENU caractères grands protégés entrée entabs données 1,
     35, 49, 1;
figure  TEXTE  caractères petits protégés entrée entabs données
     1, 52, 74, 1;
figure LIGNE caractères petits entrée entabs données 1, 52, 74,
     1;
textes LIRE, RANGER, EFFACER, CORRIGER, SUPPRIME, REMPLACE
     figure MENU;
texte LIGNE1 figure LIGNE;
niveau QUOI;
chaînes  72 caractères CHAINE1, CHAINE2; entiers I, J, K, L, M,
N, P;
tableau entier QUI (10);
```

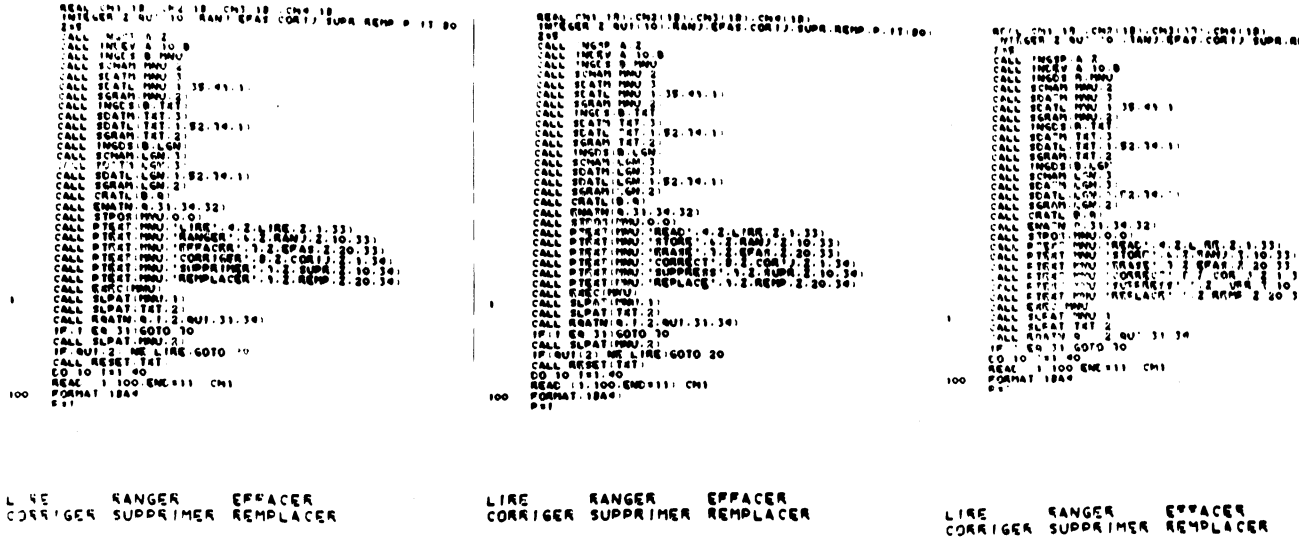


Figure 4.11 - Exemple de manipulation de texte.

```
Début figures MENU, TEXTE, LIGNE;
Début niveau QUOI;
Traiter QUOI (32, 34);
LIRE <== "LIRE" en 1, 34;
RANGER <== "RANGER" en 10, 34;
EFFACER <== "EFFACER" en 20, 34;
CORRIGER <== "CORRIGER" en 1, 35;
SUPPRIME <== "SUPPRIMER" en 10, 35;
REMPLACE <== "REMPLACER" en 20, 35;
afficher MENU;
TOUJOURS:  pointeur MENU traité;
pointeur TEXTE ignoré;
état QUOI (32, 34) résultat QUI;
si QUI (2) = LIRE;
annuler TEXTE;
pour I := 1 à 40;
lirefichier 1 dans CHAINE1;
texte TEXTE <== CHAINE1 en 1, I numéro I;
finpour;
afficher TEXTE;
P := 40;
allerà TOUJOURS finsi;
```

```
si QUI (2) = RANGER;
pour I := 1 à P;
lire 72 caractères dans CHAINE1 depuis I figure TEXTE;
écrirefichier 2 depuis CHAINE1;
finpour;
allerà TOUJOURS finsi;
si QUI (2) = EFFACER;
état QUOI (34) résultat QUI;
I := QUI (4);
J := QUI (3);
E1:  état QUOI (34) résultat QUI;
si I <= QUI (4);
signal;
allerà E1 finsi;
lire 72 caractères dans CHAINE1 depuis I figure TEXTE;
texte TEXTE <==> sélection  CHAINE1  longueur  J  ||  sélection
     CHAINE1 depuis QUI (3) en 1, I numéro I;
allerà TOUJOURS finsi;
si QUI (2) = CORRIGER;
état QUOI (34) résultat QUI;
I := QUI (4);
lire 72 caractères dans CHAINE1 depuis I figure TEXTE;
LIGNE1 <== CHAINE1 en 1, I;
omettre TEXTE numéro I;
afficher LIGNE;
mettre curseur LIGNE1 en QUI (3);
état QUOI (32);
enlever curseur LIGNE;
lire 72 caractères dans CHAINE1 depuis LIGNE1;
annuler LIGNE;
texte TEXTE <==> CHAINE1 en 1, I numéro I;
inclure TEXTE numéro I;
allerà TOUJOURS finsi;
si QUI (2) = SUPPRIME;
état QUOI (34) résultat QUI;
pour I := QUI (2) à P - 1;
lire 72 caracteres dans CHAINE1 depuis I + 1 figure TEXTE;
texte TEXTE <==> CHAINE1 en 1, I numéro I;
finpour;
annuler TEXTE numéro P;
P := P - 1;
allerà TOUJOURS finsi;
si QUI (2) = REMPLACE;
état QUOI (34) résultat QUI;
I := QUI (4);
J := QUI (3);
```

```
E2:  état QUOI (34) résultat QUI;
si I <> QUI (4);
signal;
allerà E2 finsi;
K := QUI (3);
état QUOI (34) résultat QUI;
L := QUI (4);
M := QUI (3);
E3:  état QUOI (34) résultat QUI;
si L <> QUI (4);
signal;
allerà E3 finsi;
N  :=  QUI (3); lire 72 caractères dans CHAINE1 depuis I figure
     TEXTE;
lire 72 caracteres dans CHAINE2 depuis L figure TEXTE;
sélection CHAINE1 depuis J longueur K - J := sélection  CHAINE2
     depuis M longueur N - M;
texte TEXTE <==> CHAINE1 en 1, I numéro I;
allerà TOUJOURS finsi;
```

4.176 - Arrêt programmé.

L'arrêt programmé est un ordre de la mémoire d'entretien qui produit une interruption quand on l'exécute au cours du cycle de régénération. Son utilité principale est de permettre l'animation de dessins, puisque par ce moyen le programme peut modifier l'image après chaque affichage. On génère un arrêt au moyen de l'instruction:

mettre arrêt (nom-d'arrêt | désignation-par-numéro) (inclus | omis) ;

Cette instruction est fonctionnellement équivalente à l'instruction de génération d'un objet; l'option inclus est l'option par défaut; un arrêt omis ne produit pas d'interruption. On peut ensuite modifier l'état d'un arrêt par les instructions inclure et omettre, et le prendre comme opérande des instructions annuler et lire (après depuis ou à).

L'exécution d'un arrêt programmé au cours du cycle de régénération provoque l'interrutpioin de numéro 35; le tableau résultat de l'instruction état contient les mêmes informations qu'après l'interruption due au pointeur optique, sauf pour les mots 8 à 10 qui sont nuls:

- mot 1: nom de la figure;
- mot 2: nom de l'arrêt;
- mot 3: inutile;
- mot 4: numéro de l'arrêt;

- mot 5: nom de la séquence;
- mot 6: inutile;
- mot 7: numéro de la séquence.

Les exemples ci-dessous montrent quelques utilisations de l'arrêt programmé, là encore sous la forme de programmes complets.

Le premier exemple montre le mouvement d'une bielle dont une extrémité glisse sur un segment fixe, et l'autre tourne sur un cercle également fixe; la figure 4.12 montre tous les états de l'image au cours du mouvement.

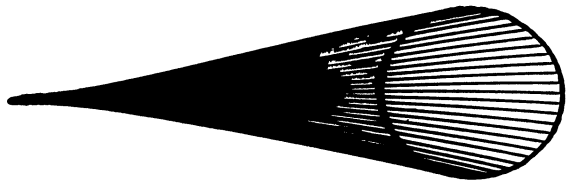


Figure 4.12 - Premier exemple d'utilisation de l'arrêt programmé.

```
figure VUE données 1., 1., 100., 100.;
figure DESSIN données 1., 1., 100., 100.;
segment BIELLE figure DESSIN;
niveau ARRET;
tableaux entiers PX, PY, PZ (100);
entier I;
réels R := 15., PI := 3.141593, RAD;
début figures VUE, DESSIN;
début niveau ARRET;
traiter ARRET (35);
pour I := 1 à 100;
```

```
RAD := I * PI / 50.;
PX (I) := 85.  + R * COS (RAD);
PY (I) := 50.  + R * SIN (RAD);
PZ (I) := 25.  + R * COS (RAD);
finpour;
position DESSIN <== PX (100), PY (100);
ligne DESSIN <== 100 fois tableau PX (1), tableau PY (1);
segment DESSIN <== 10., 50., 40., 50.;
afficher DESSIN;
BIELLE <== PZ (100), 50., PX (100), PY (100);
mettre arrêt VUE numéro 1;
afficher VUE;
HOP:  pour I := 1 à 100:
état ARRET (35);
BIELLE <==> PZ (I), 50., PX (I), PY (I);
finpour;
allerà HOP;
```

Le deuxième exemple montre la rotation d'un cube dans l'espace; l'utilisation de deux figures équivalentes CUBE1 et CUBE2 permet de calculer une position du cube pendant que la précédente apparaît sur l'écran; la figure 4.13 montre quelques états de l'image au cours du mouvement.

```
figure CUBE1 données - 2., - 2., 2., 2.  renomme CUBE2;
tableaux  réels PX := (1., 1., - 1., - 1., 1., 1., - 1., - 1.),
     PY := (- 1., 1., 1., - 1., - 1., 1., 1., - 1.), PZ := (1.,
     1., 1., 1., - 1., - 1., - 1., - 1.), PX1, PPY1, PZ1 (8);
réels AX := 0.1, AY := 0.15, AZ := 0.2;
niveau HALTE;
entiers I, J, K;
arrêt STOP1 figure CUBE1;
arrêt STOP2 figure CUBE2;
début figure CUBE1;
traiter HALTE (35);
mettre arrêt STOP1;
ICI:  pour I := 1 à 8;
PX1 (I) := PX (I) * COS (AX) + PY (I) * SIN (AX);
PY1 (I) := PX (I) * SIN (AX) - PY (I) * COS (AX);
PY (I) := PY1 (I) * COS (AY) + PZ (I) * SIN (AY);
PZ1 (I) := PY1 (I) * SIN (AY) - PZ (U) * COS (AY);
PZ (I) := PZ1 (I) * COS (AZ) + PX1 (I) * SIN (AZ);
PX (I) := PZ1 (I) * SIN (AZ) - PX1 (I) * COS (AZ);
finpour;
K := CUBE1;
CUBE1 := CUBE2;
CUBE2 := K;
```

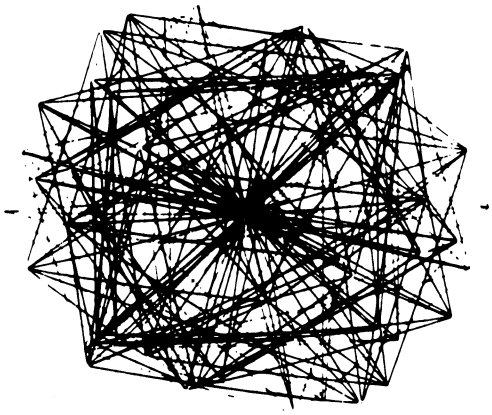


Figure 4.13 - Deuxième exemple d'utilisation de l'arrêt programmé.

```
K := STOP1;
STOP1 := STOP2;
STIP2 := K;
afficher CUBE1;
annuler CUBE2;
E1:  pour I := 1 à 7;
pour J := I + 1 à 8;
segment CUBE2 <== PX (I), PY(I), PX(J), PY (J);
finpour E1;
mettre arrêt STOP2;
état HALTE (35);
allerà ICI;
```

4.18 - INSTRUCTIONS DIVERSES.

4.181 - Fonctions.

Quatre fonctions permettent l'examen de certaines conditions qui concernent les variables ou les instructions graphiques; elles peuvent être utilisées partout où peut être une constante arithmétique. La première permet d'examiner certaines des caractéristiques

d'une figure:

type nom-de-figure (abscisse | ordonnée | sortie | texte)

Le tableau suivant résume les valeurs que peut prendre la fonction type, suivant la question posée:

OPTION	VALEUR	SIGNIFICATION
abscisse	1	réelabs
ou	2	réelrel
ordonnée	3	entabs
	4	entrel
sortie	1	optimisé
	2	absolu
	3	incrémentiel
texte	1	petit protégé
	2	grand protégé
	3	petit
	4	grand

Les deux fonctions suivantes permettent de connaître les dernières coordonnées du faisceau lumineux dans une figure:

(abscisse | ordonnée) nom-de-figure (entier | réel) ?(supposé | effectif)

L'option supposé fournit les coordonnées qu'aurait le faisceau s'il n'y avait pas eu de coupage; s'il n'y en a effectivement pas eu, effectif (l'option par défaut) et supposé fournissent le meme résultat. Si le type indiqué (entier ou réel) ne correspond pas au mode courant des données dans la figure, la fonction prend la valeur négative maximale.

Enfin, la fonction erreur permet de contrôler l'exécution de la dernière instruction graphique:

erreur entier-sans-signe

L'entier indique quel type d'erreur on veut examiner, d'après la codification suivante:

1: coupage (ce n'est pas forcément une erreur);
2: erreur d'échelle;
3: dépassement de capacité de mémoire;
4: paramètre incorrect.

La fonction erreur prend une valeur entière, nulle s'il n'y a pas eu d'erreur de numéro supérieur ou égal à l'erreur examinée, sinon de

valeur égale au numéro d'erreur le plus fort. On peut obtenir plus de renseignements sur la nature de l'erreur en réutilisant la fonction erreur avec le paramètre 9; la réponse est alors la suivante:

1: numéro de l'élément sur lequel a eu lieu le coupage (dans une instruction de génération simple, c'est toujours 1);
2: numéro de l'élément sur lequel il y a eu erreur d'échelle;
3: 1 si l'objet mis à jour occupe plus de place que l'ancien;
   2 si une figure de 128 octets a été débordée;
   3 s'il n'y a plus de place en mémoire centrale;
4: numéro d'ordre du paramètre incorrect (voir [IBM2]).

4.182 - Instructions.

signal;

Cette instruction actionne le signal auditif du terminal; ceci peut par exemple servir à signaler à l'opérateur qu'il a commis une erreur.

fingraphique;

Cette instruction marque la fin logique du programme, c'est-à-dire qu'elle est la dernière instruction exécutée; une instruction fingraphique est toujours supposée exister après la dernière instruction physique du programme.

modifier nom-de-niveau de E.A. ?(enhaut | enbas) ;

Cette instruction ne sert que dans le cas où l'on utilise simultamément plusieurs niveaux, c'est-à-dire assez rarement; elle permet de déplacer un niveau précis dans la pile des niveaux, d'un certain nombre de positions vers le haut (vers le niveau courant) ou vers le bas (vers le premier niveau initialisé); l'option part défaut est enhaut.

4.183 - Textes spéciaux.

Un texte spécial n'utilise pas le générateur de caractères du terminal, ce qui permet d'obtenir des caractères dont on choisit la taille, la largeur, l'espacement, l'orientation, et même la forme; en revanche, ces caractères sont générés au moyen de petits vecteurs, ce qui les rend assez encombrants en mémoire d'entretien (on a intérêt à les placer dans une figure dont les ordres sont optimisés ou incrémentiels). Un texte spécial est un objet au même

titre que les points, lignes, segments et textes mormaux, c'est-à-dire qu'il peut être manipulé par les instructions inclure, omettre et annuler, et qu'il peut être généré et mis à jour. Il est déclare comme un texte normal, sa nature spéciale n'apparaissant qu'au moment de la génération ou de la mise à jour:

spécial entier-sans-signe ((texte nom-de-figure) ¦ nom-de-texte) symbole-de-génération expression-de-chaîne ?(en E.E. , E.A.) hauteur E.A. ?(largeur E.A.) ?(espacement E.A.) ?(orientation nom-de-tableau) ?(numéro E.A.) ?(inclus ¦ omis) ;

L'entier sans signe après spécial est le numéro de la table de caractères utilisée; la seule table fournie normalement porte le numéro 1, elle ne contient que les symboles déjà définis par le générateur de caractères, et avec la même forme; les autres tables doivent être fabriquées suivant les indications données en référence [IBM2]. Le symbole de génération peut aussi être un symbole de mise à jour, toutes les règles données aux paragraphes 4.153 et 4.163 restant valables. Les coordonnées après en sont celles du premier caractère, si elles sont absentes on prend la dernière position du faisceau lumineux.

Les paramètres suivants définissent la taille et l'espacement des caractères; les valeurs données sont des nombres réels de pouces, au moins égaux à 0.125; si la largeur n'est pas indiquée, on prend les deux tiers de la hauteur; l'espacement est du centre du caractère au centre du suivant; s'il n'est pas indiqué, il vaut quatre tiers de la hauteur. A titre d'exemple, l'espacement entre caractères est de 0.164 pouce pour les caractères de petite taille produits par le générateur, et 0.246 pouce pour les caractères de grande taille.

L'orientation est donnée par un tableau de quatre nombres reels qui donnent successivement sin A, cos A, sin B et cos B, A et B étant définis par la figure 4.14; si orientation est absent, A et B sont nuls, c'est-à-dire que le texte est horizontal avec les caractères verticaux. Un texte spécial en mode omis déplace le faisceau lumineux comme quand il est inclus.

4.19 - SYMBOLES DE BASE COMPOSES.

On trouvera ci-après tous les symboles de base composés du langage Euphémie, dans leurs formes française et anglaise. La partie soulignée de chaque symbole représente la forme minimale qu'il doit prendre pour être reconnu; la partie entre parenthèses n'est pas prise en compte car elle dépasse la taille maximale de

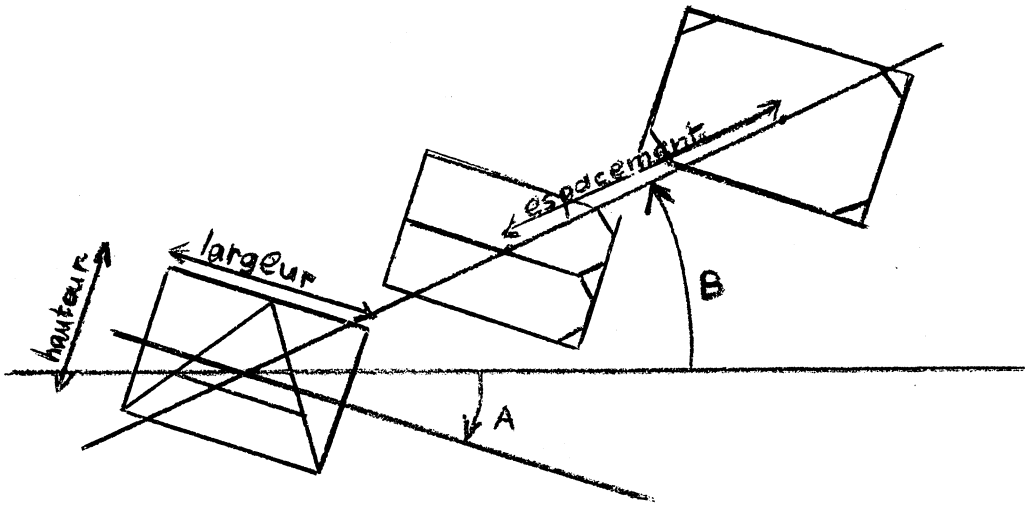


Figure 4.14 - Description des textes spéciaux.

huit caractères; elle peut donc être remplacée par n'importe quelle suite de lettres. Ainsi on trouvera:

CARACTER(ES)

Les symboles suivants sont reconnus comme celui-là:

CARA CARAC CARACT CARACTE CARACTER CARACTERE CARACTERES CARACTERIEL CARACTERISTIQUE

De meme ORDONNER et ORDONNERENT sont le même sybole.

4.191 - Symboles intervenant dans les expressions.

SELECTIO(N)	SELECT	ABSCISSE	ABSCISSA
DEPUIS	FROM	ORDONNEE	ORDINATE
LONGUEUR	LENGTH	ENTIER	INTEGER
PAS	STEP	REEL	REAL
TYPE	TYPE	SUPPOSE	SUPPOSED
SORTIE	OUTPUT	EFFECTIF	ACTUAL
TEXTE	TEXT	ERREUR	ERROR

4.192 - Symboles déterminant les instructions.

LIGNES	LINES	MODIFIER	MODIFY
POINTS	POINTS	SI	IF
SEGMENTS	SEGMENTS	SINON	ELSE
POSITION	POSITION	INCLURE	INCLUDE
ETAT	STATUS	OMETTRE	OMIT
LIRE	READ	SITUER	SITUATE
TRAITER	PROCESS	POINTEUR	LIGHTPEN
IGNORER	IGNORE	SPECIAL	SPECIAL
FINGRAPH(IQUE)	ENDGRAPH(ICS)	PLACER	PLACE
ALLERA	GOTO	AFFICHER	DISPLAY

DEBUT	BEGIN	METTRE	PUT
FIN	END	ORDONNER	ORDER
LIREPOIN(TEUR)	READPEN	ANNULER	RESET
TOUCHES	KEYS	ENLEVER	REMOVE
POUR	FOR	OPTIONS	OPTIONS
FINPOUR	ENDFOR	SIGNAL	ALARM
LIRESTYL(ET)	READSTYL(US)	COMMENTA(IRE)	COMMENT

4.193 - Symboles déterminant les déclarations.

REELS	REALS	POSITION(S)	POSITION(S)
ENTIERS	INTEGERS	ARRETS	STOPS
LIGNES	LINES	TABLEAUX	ARRAYS
POINTS	POINTS	SEQUENCE(S)	SEQUENCE(S)
SEGMENTS	SEGMENTS	NIVEAUX	LEVELS
CHAINES	STRINGS	FIGURE	FIGURE
TEXTES	TEXTS		

4.194 - Symboles introductifs.

DONNEES	DATA	ATTENTE	WAIT
DANS	INTO	NUMERO	NUMBER
EN	IN	FIGURE	FIGURE
JUSQUA	UNTIL	TAILLE	SIZE
FIN	END	GARDE	KEEP
CARACTER(ES)	CHARACTE(RS)	ENTREE	INPUT
A	TO	REELABS	REALABS
REPONSE	RESPONSE	REELREL	REALREL
TABLEAU	ARRAY	ENTABS	INTABS
SORTIE	OUTPUT	ENTREL	INTREL
SEQUENCE	SEQUENCE	RENOMME	RENAMES
ARRET	STOP	COUPAGE	SCISSORI(NG)
DEFAUT	DEFAULT	EC	SC
ETEINT	OFF	EA	SS
NORMAL	NORMAL	FC	FC
PAS	STEP	FA	FS
DE	BY	LIBRE	FREE
ENHAUT	UP	CURSEUR	CURSOR
ENBAS	DOWN	ABSOLUES	ABSOLUTE
RETOUR	RETURN	OPTIMISE	OPTIMIZE(D)
HAUTEUR	HEIGTH	INCREMEN(TIEL)	INCREMEN(TAL)
LARGEUR	WIDTH	PETITS	SMALL
ESPACEME(NT)	SPACING	GRANDS	LARGE

ORIENTAT(ION)	ORIENTAT(ION)	PROTEGES	PROTECTE(D)
ZONE	AREA	ECRAN	SCREEN

4.2 - PRICIPES DE CONCEPTION DU LANGAGE EUPHEMIE.

Si l'on suit un plan historique, les principes de conception du langage Euphémie peuvent s'exposer en quatre étapes: les outils dont nous sommes partis (nous, c'est-à-dire M. Cleemann et moi-même), fournis par GSP; la définition des déclarations et instructions graphiques nécessaires, au moyen du langage Macro-Euphémie; la définition du langage Euphémie lui-même, avec son symbolisme et ses particularités syntaxiques; enfin les projets d'extensions prévisibles dans diverses directions. Comme ce plan historique est en même temps assez logique, et qu'il se raccorde bien au schéma de conception des outils et des langages que j'ai proposé au chapitre précédent, c'est celui auquel je me tiendrai.

4.21 - L'ENSEMBLE DE SOUS-PROGRAMMES G.S.P.

Pour définir un langage de programmation graphique du niveau des figures, il faut d'abord disposer des outils correspondants; dans le cas qui nous occupe, c'est-à-dire la définition d'un langage destiné au terminal graphique IBM 2250 [IBM3], pour fonctionnemnt sous l'un ou l'autre des systèmes d'exploitation OS/360 [IBM4] et CP/CMS [IBM5], le constructeur fournissait l'ensemble d'outils voulus, sous le nom de "Graphic Subroutine Package" ou G.S.P. [Ru68, IBM2]. C'est cet ensemble d'outils que je vais décrire brièvement, pour en dégager les avantages et les inconvénients, et montrer ce que lui doit Euphémie.

4.211 - Les outils fournis par G.S.P.

G.S.P. se présente extérieurement comme un ensemble de cinquante-sept sous-programmes, prévus pour pouvoir être appelés sans modifications dans quatre contextes différents: Fortran, Cobol, PL/I et le langage d'assemblage de l'ordinateur 360. Leur utilisation n'est cependant réellement praticable qu'en Fortran et en langage d'assemblage; en Cobol, chaque appel de sous-programme doit être encadré des ordres "ENTER LINKAGE." et "ENTER COBOL.", ce qui rend l'écriture d'une lourdeur absolument rédhibitoire, même si l'on est habitué à la lourdeur inhérente au langage Cobol lui-même; en PL/I, les particularités de communication des paramètres de procédures et de codification des constantes arithmétiques rendent

l'écriture presque aussi lourde qu'en Cobol, et occasionnent des erreurs continuelles.

Une des notions fondamentales de G.S.P. est le "fichier graphique" (graphic data set), qui correspond à la figure d'Euphémie. Dans un fichier graphique on peut créer et manipuler des "éléments", qui correspondent à des objets dans ma terminologie (positions, lignes, points, segments, textes et arrêts); pour se référer à ces "éléments", on dispose de deux moyens, la "clé" (qui correspond au nom des objets en Euphémie) et la "corrélation" (qui correspond au numéro). Les autres outils importants sont la séquences, le sous-programme (non réalisé dans Euphémie car il nécessite le terminal 2250 de modèle 3), et le niveau d'interruptions (attention level), qui est très exactement celui d'Euphémie.

Tous ces outils doivent être manipulés au moyen de sous-programmes aux noms plus ou moins ésotériques (ces noms sont en effet limités à cinq caractères, pour compatibilité avec l'ordinateur IBM 1130 [IBM12]), dont certains jouent le rôle de déclarations, et d'autres le rôle de véritables instructions. On répartit ces sous-programmes en dix catégories plus ou moins logiques: initialisations; terminaison; définition des options; génération de l'image; identification (séquences et sous-programmes graphiques); contrôle de l'image; entrée au clavier alphanumérique; traitement des interruptions; pointeur optique; divers.

4.212 - Réalisation des outils de G.S.P.

Le mode de fonctionnement interne de G.S.P. est décrit en détail en référence [IBM14]; quelques indications apparaissent aussi en référence [Cl69]; je ne parlerai ici que des quelques points qui sont nécessaires.

La mémoire d'entretien du terminal est divisée en trente-deux zones de 256 octets, allouées dynamiquement aux figures; la liste d'affichage correspondant à une figure occupe une ou plusieurs zones qui peuvent être disjointes; les listes d'affichage correspondant à chacune des figures sont considérées comme autant de sous-programmes, appelés successivement au cours d'un cycle de régénération. Ce procédé rend très simple l'omission complète d'une figure comme sa restauration; cependant, la gestion dynamique de la mémoire ne descend pas plus bas que ce niveau.

L'ensemble de G.S.P. est réalisé sous forme modulaire; à chaque sous-programme correspond un module (sauf dans deux cas précis), et de plus seize modules servent à certains problèmes communs à tous

les sous-programmes; au total, il s'agit donc de soixante-douze modules de cinq cents instructions en moyenne, physiquement bien séparés les uns des autres, mais logiquement très imbriqués. La communication entre ces modules se fait, comme c'est la coutume dans l'ensemble du système OS/360, au moyen de nombreux "blocs de contrôle", c'est-à-dire de zones de mémoire de longueur fixe, acquises dynamiquement, et reliées les unes aux autres par de multiples pointeurs. Il y a ainsi par exemple un bloc pour chaque figure, un bloc pour chaque niveau, un bloc pour chaque interruption produite et non encore traitée; le nom d'une figure ou d'un niveau est alors un pointeur sur le bloc correspondant.

Le nom d'un objet (sa "clé" dans la terminologie de G.S.P.) ne désigne pas un bloc de contrôle, mais donne simplement l'adresse en mémoire d'entretien des ordres de génération de l'objet, ainsi que leur longueur. Par l'intermédiaire d'une table liée à chaque figure, G.S.P. est capable de retrouver toutes les caractéristiques de l'objet au moyen de ces deux informations. Le numéro de l'objet (sa "corrélation") est simplement placé dans la table à côté de sa "clé", et c'est par une consultation séquentielle élémentaire que G.S.P. retrouve la "clé" associée à une valeur de corrélation donnée.

Le traitement des interruptions passe par plusieurs intermédiaires avant d'arriver au programme lui-même sous la forme d'une réponse au sous-programme RQATN (instruction état d'Euphémie); le travail fait par G.S.P. lui-même se superpose en effet au "support graphique de base" [IBM7], tributaire lui-même de la gestion générale des interruptions par le système OS/360 (ou par CP/CMS si c'est le système que l'on utilise). C'est ainsi qu'il peut se construire des files d'attente à quatre niveaux différents, si l'on compte le niveau technologique introduit par le canal de communication auquel est connecté le terminal.

L'ensemble de G.S.P. doit pouvoir fonctionner sur une mémoire de taille réduite, aussi ne peut-on placer les modules en mémoire qu'au moment de leur utilisation. Si l'on tient compte de ce que, pour afficher un simple point, il faut passer en général par sept modules, on peut s'attendre à des performances pour le moins médiocres dans la plupart des cas. En fait, si le programmeur choisit judicieusement les modules qui doivent être en permanence en mémoire, le résultat n'est pas si mauvais qu'on pourrait le penser, et ce malgré les nombreuses raisons de ralentissement déjà vues. La raison en est qu'en travail conversationnel, le temps de réponse reste à peu près toujours acceptable tant que l'ordinateur n'a pas à effectuer de processus répétitif. Si de plus, comme c'estle cas

quand on utilise G.S.P. avec le système CP/CMS, on peut supprimer le chargement dynamique, on aboutit à des performances très honnêtes, qui font de G.S.P. un bon outil dans la plupart des cas.

4.213 - Utilisation de G.S.P.

Les outils fournis par G.S.P. sont indiscutablement d'un grand intérêt, même si l'on peut leur faire quelques reproches mineurs, tels que la complexité d'utilisation des niveaux d'interruptions, ou certaines restrictions de fonctionnement mal expliquées. En revanche, l'utilisation de ces outils au moyen d'appels très nombreux de sous-programmes, dans un langage de programmation qui n'est pas prévu pour cela, présente des inconvénients graves; même si l'on n'est pas rebuté par ces inconvénients, l'écriture du moindre programme est fastidieuse et lassante, et sa mise au point encore plus. Cela ne signifie pas pour autant que les utilisateurs se désintéressent de G.S.P.: en fait ils ne disposent de rien d'autre, et doivent donc se débrouiller avec ce qu'ils ont (l'expérience montre d'ailleurs que l'intensité d'utilisation d'un langage de programmation n'a rien à voir avec ses qualités intrinsèques).

Les principaux des inconvénients de l'utilisation de G.S.P. dans le langage Fortran sont les suivants:
- les paramètres des sous-programmes sont généralement très nombreux: souvent une dizaine, parfois plus, le record pouvant atteindre trente-neuf pour le sous-programme RQATN et cinquante-trois pour INGDS;
- l'emplacement des paramètres dans l'instruction d'appel de sous-programme est impératif parceque significatif; cela conduit à une technique très lourde pour signaler l'omission de certains paramètres;
- les paramètres sont confondus dans un anonymat complet, et rien ne distingue une variable graphique d'une valeur de coordonnée ou d'un numéro d'option;
- les sous-programmes sont tout aussi anonymes sous leurs noms obscurs parceque trop abrégés; lesdites abréviations ne sont pas faites de façon suffisamment systématique, et rien ne distingue les sous-programmes qui servent à proprement parler de déclarations de ceux qui font une génération effective ou de ceux qui traitent les problèmes de communication.

Les exemples ci-dessous montrent, par comparaison avec Euphémie, les résultats auxquels on arrive:

- Initialisations (en G.S.P.) ou déclarations (en Euphémie):

```
CALL INGSP(IBM,ZUT)
CALL INDEV(IBM,10,IMAG)
CALL INGDS(IMAG,F)
CALL SGDSL(F,0.,-200.,100.,200.)

figure F zone 0., - 200., 100., 200.;
point P figure F;
texte SPEC;
```

- Génération d'un point:

```
CALL PPNT(F,X,Y,ZUT,P)

P <== X, Y;
```

- Génération de dix segments:

```
CALL SDATM(F,2,2)
CALL      PSGMT(F,0.,0.,1.5,2.0,ZUT,ZUT,ZUT,10,ZUT,ZUT,ZUT,ZUT,
           0.1,0.1,0.25,0.#1)
segment F <## 10 fois 0 pas 0.1, 0 pas 0.1, 1.5 pas 0.25, 2.0
           pas 0.#1;
```

- Lecture de la mémoire d'entretien:

```
CALL GSPRD(F,TEXTE,-80,1,TERME,ZUT,P1,ZUT,P2)

lire 80 caractères dans TEXTE depuis P1 à P2 réponse TERME;
```

- Génération de texte spécial:

```
CALL PLSTR(F,1,TEXTE,80,2.,ZUT,ZUT,TABLE,ZUT,SPEC,ZUT,25.,0.)

spécial SPEC <== TEXTE en 25., 0. hauteur 2. orientation
     TABLE;
```

4.22 - LE LANGAGE MACRO-EUPHEMIE.

Macro-Euphémie n'est pas un véritable langage puisqu'il s'agit en fait d'un ensemble de macros-instructions à utiliser dans un programme écrit dans le langage d'assemblage de l'ordinateur 360. Il a été défini comme un intermédiaire rapidement construit qui permette, d'une part d'utiliser commodément G.S.P. en langage d'assemblage, d'autre part de définir la forme des instructions graphiques à inclure dans un langage spécialisé qui s'appuie sur G.S.P., en l'occurence Euphémie. On trouvera dans [C169] une description complète du langage Macro-Euphémie, qui ne sera ici que présenté.

4.221 - Le macro-assembleur.

Le langage d'assemblage de l'ordinateur 360 [IBM13] fournit des moyens de macro-traitement importants en volume et en complexité, sinon en rigueur ou en efficacité. Ces moyens permettent en quelque sorte de considérer que l'on écrit un programme qui doit être exécuté par l'assembleur, et dont le résultat est de produire une suite d'octets qui constitueront les instructions et données d'un programme dans le langage de l'ordinateur 360. J'appellerai les composants du langage d'assemblage des directives, pour éviter la confusion avec les instructions du langage machine. Une instruction du langage Macro-Euphémie a donc pour résultat de générer une suite d'instructions et de données. Les deux notions importantes qui permettent ce fonctionnement sont les variables d'assemblage et les macros-instructions.

Une variable d'assemblage est une variable de ce programme qu'exécute l'assembleur; c'est donc un symbole que l'on utilise pour la valeur qu'il désigne, contrairement aux symboles ordinaires qui apparaissent en étiquettes d'instructions ou de données du programme, et qui servent à représenter une adresse dans le programme produit. Une variable d'assemblage n'a de signification que pendant l'assemblage lui-même; elle peut au cours de ce dernier changer de valeur, on dispose pour cela de directives d'affectation tout à fait classiques. Il existe trois types de variables, suivant le type de la valeur qu'elles peuvent désigner: arithmétique, logique (booléenne) ou alphanumérique (chaîne); il existe une directive d'affectation pour chacun des types, avec pour partie droite une expression arithmétique, logique ou alphanumérique qui peut atteindre une grande complexité. Les variables peuvent être des tableaux à une dimension.

A côté des instructions du langage machine, qui doivent en fait être considérées du point de vue du langage d'aassemblage comme des directives de génération de données, l'assembleur reconnaît les directives de génération habituelles (définition de constantes); ces directives constituent l'analogue des instructions d'écriture d'un langage classique. Les autres directives importantes sont: les trois directives d'affectation; la directive de branchement, qui nécessite la définition d'étiquettes du langage d'assemblage, sans aucun rapport avec les étiquettes du programme à générer; la directive de branchement conditionnel; les directives de déclaration des variables d'assemblage; la directive ineffective. Il n'existe pas de directive d'itération, ce qui alourdit l'écriture des processus répétitifs.

La macro-instruction correspond à la procédure d'un langage évolué: sa définition constitue une déclaration, qui ajoute une nouvelle directive au langage d'assemblage. Les paramètres formels de la macro-instruc tion sont utilisés dans la définition comme des variables d'assemblage, ils peuvent être remplacés à l'appel des procédures par à peu près n'importe quoi (avec toutefois des restrictions importantes). Une particularité intéressante du langage d'ssemblage de l'ordinateur 360 est qu'il existe deux façons d'écrire les paramètres: la première est celle que l'on trouve dans tous les langages classiques, le paramètre se reconnaît à son emplacement dans la liste; la deuxième façon utilise un mot-clé pour désigner le paramètre, ce qui permet d'une part d'écrire les paramètres effectifs dans un ordre quelconque, d'autre part d'affecter une valeur initiale aux paramètres formels au moment de la définition de la macro-instruction. Comme d'autre part on peut vérifier dans le corps de la macro-instruction si un paramètre existe ou non, c'est-à-dire s'il lui correspond un paramètre effectif, on arrive à une grande souplesse d'écriture des appels de macros-instructions.

Une dernière notion intéressante est celle des fonctionnelles, qui permettent d'examiner un grand nombre de caractéristiques des variables d'assemblage et des symboles normaux: type, nature, longueur, précision, existence, dimension, etc.

Au total, ce langage d'assemblage semble donc très puissant. En fait, les défauts qui apparaissent dès que l'on commence à l'utiliser sont nombreux et graves, et la définition et la mise au point de macros-intructions nombreuses et complexes constituent en fait un travail long et difficile. Ces défauts peuvent se résumer brièvement: lourdeur, syntaxe pénible, manque de rigueur et surtout manque de généralité; certaines notions sont mal définies, certaines idées ne sont pas poussées jusqu'au bout, l'ensemble est disparate, mal bâti et finalement décevant. Enfin, le plus gros défaut n'apparaît qu'une fois qu'il est trop tard: l'assemblage d'un programme utilisant intensivement les macros-instructions est d'une lenteur désespérante, et l'on doit réfléchir à deux fois avant de se décider à en faire un.

4.222 - Construction de Macro-Euphémie.

Le langage Macro-Euphemie est formé de quarante instructions, qui s'écrivent au moyen de trente-cinq macros-instructions, lasquelles utilisent dans leur définition une vingtaine de macros-instructions auxiliaires. Certaines de ces macros-instructions ne génèrent rien, elles se contentent de faire certaines vérifications et de ranger

des valeurs dans des variables d'assemblage, tandis que les autres génèrent des instructions qui ne sont pratiquement que des appels à des sous-programmes de G.S.P., le plus gros travail étant la constitution des listes de paramètres.

La transmission des valeurs entre les macros-instructions par l'intermédiaire des variables d'assemblage permet d'éviter un des ennuis particuliers à G.S.P., c'est-à-dire la nécessité de répéter très fréquemment le nom des figures utilisées: une macro-instruction de déclaration associe au nom d'un objet celui de la figure dont il fait partie, et la macro-instruction de génération pourra le retrouver dans une variable d'assemblage à une dimension.

Les deux procédés d'écriture des paramètres permettent d'associer un mot-clé à tous les paramètres facultatifs, ce qui évite le procédé lourd de la variable fictive qui sert dans G.S.P. à indiquer l'omission d'un paramètre; d'autre part, le choix d'un mot-clé significatif rend l'écriture des instructions parlante, ce qui est un avantage considérable.

L'ensemble de macros-instructions défini ne concerne que les aspects graphiques de l'utilisation du terminal. En effet, l'adjonction de macros-instructions qui auraient permis l'écriture d'instructions non graphiques (affectation, aller à, si, pour) n'aurait rien ajouté d'un point de vue théorique, et se serait montrée inintéressante pour l'usage pour lequel est prévu Macro-Euphémie. En effet, à côté de son utilité comme première approche de la définition d'Euphémie, ce langage est réellement utilisable et utilisé: il sert à l'écriture de toutes les parties graphiques du système Eulalie, qui sert à l'implantation du langage Euphémie, et sera décrit au chapitre 6.

4.223 - Utilisation de Macro-Euphémie.

On trouvera en annexe 4 des exemples de programmes utilisant Macro-Euphémie, et pris dans certains modules du systeme Eulalie. Pour voir rapidement les avantages de Macro-Euphémie sur la simple écriture d'appels des sous-programmes de G.S.P., on peut considérer les exemples suivants, qui reprennent ceux qui ont été donnés au paragraphe 4.213.

1°) - En langage d'assemblage uniquement:

```
CALL INGSP,(IBM,ZUT),VL
CALL INDEV,(IBM,DIX,IMAG),VL
CALL INGDS,(IMAG,F),VL
```

```
          CALL SGDSL,(F,ZEROP,MDEUCENP,CENTP,DEUCENP),VL
          ...
          CALL PPNT,(F,X,Y,ZUT,P),VL
          ...
          CALL SDATM,(F,DEUX,DEUX),VL
          CALL PSGMT,(F,ZEROP,ZEROP,UNCINQP,DEUXP,ZUT,ZUT,ZUT, X
               DIX,ZUT,ZUT,ZUT,ZUT,ZEROUNP,ZEROUNP,ZVINCINP,     X
              ZQUAUNP),VL
          ...
          CALL GSPRD,(F,TEXTE,MQUA20,UN,TERME,ZUT,P1,ZUT,P2),VL
          ...
IBM       DS   F
ZUT       DC   F'K'
DIX       DC   F'10'
IMAG      DS   F
F         DS   F
ZEROP     DC   E'0'
MDEUCENP  DC   E'-200'
CENTP     DC   E'100'
DEUCENP   DC   E'200'
X         DS   E
Y         DS   E
P         DS   F
DEUX      DC   F'2'
UNCINQP   DC   E'1.5'
DEUXP     DC   E'2'
ZEROUNP   DC   E'0.1'
ZVINCINP  DC   E'0.25'
ZQUAUNP   DC   E'0.41'
TEXTE     DS   CL80
MQUA20    DC   F'-80'
UN        DC   F'14'
TERME     DS   F
P1        DS   F
P2        DS   F
```

2°) - Avec Macro-Euphémie:

```
          GRAPHIQ ENTIER=TERME,REEL=(X,Y),ETAB=(TERME,20)
          FIGURE F,ZONE=(0.,-200.,100.,200.)
          POINT P,FIGURE=F
          ...
          GENERER P,'<==',COOR=(X,Y)
          ...
          GENERER F,'<##',OBJET='SEGMENT',COOR=(0.,0.,1.5,2.), X
              NB=10,XPAS=(0.1,0.25),YPAS=(0.1,0.41)
```

...
LIRE DANS=TEXTE,DEPUIS=P1,A=P2,REPONSE=TERME

L'avantage le plus spectaculaire de Macro-Euphémie est qu'il évite la définition explicite des constantes nécessaires, qui rendent extrêmement pénible l'utilisation de G.S.P. directement en langage d'assemblage. Les autres avantages sont ceux qui ont été exposés auparavant à plusieurs reprises, c'est-à-dire principalement la clarification et la simplification.

L'inconvénient majeur est la durée des assemblages, qui rend très coûteuses les retouches successives à faire aux programmes. A l'usage, on découvre un autre inconvénient par rapport à l'utilisation directe de G.S.P., qui est que l'on ne peut plus tenir compte de certaines particularités de conventions de ce système. Par exemple, un paramètre des sous-programmes de génération peut prendre les valeurs 1, 2 ou 3, qui indiquent respectivement la génération en mode inclus, la génération en mode omis et la mise à jour. En changeant simplement la valeur de ce paramètre, on peut donc transformer une instruction de génération en une instruction de mise à jour, ce qui n'est pas possible avec Macro-Euphémie. En fait, cet inconvénient peut se tourner en avantage par la clarification qu'il apporte dans la programmation, en rendant plus rigoureuses la syntaxe et la sémantique des instructions.

Au total, l'usage de Macro-Euphémie est d'un bilan nettement positif, et la forme qu'il indique pour les instructions graphiques est suffisamment claire et pratique pour que l'on puisse sans crainte la transposer presque sans modifications dans le langage Euphémie.

4.23 - LES PARTICULARITES DU LANGAGE EUPHEMIE.

4.231 - Symbolisme adopté.

Quand on doit choisir les symboles qui serviront à composer un langage de programmation, on peut les fabriquer de deux façons différentes: avec des caractères spéciaux (par exemple :=, [,], =>) ou avec des mots en toutes lettres (par exemple PROCEDURE, DEBUT, SECTION). Le premier système est parfois plus séduisant, pour plusieurs raisons, dont la moindre n'est pas l'asepct cryptographique et pseudo-mathématique que prend le langage; le modèle du genre est le langage APL. Le deuxième satisfait ceux qui persistent à penser qu'une langue naturelle peut servir de langage de programmation, et son application "jusqu'au bout" conduit au langage Cobol.

Ces deux modèles me semblent également mauvais en ce qu'ils ont d'extrême: on peut bien sûr écrire en une demi-ligne d'APL ce qui dans un autre langage prendrait une page, mais il faudra un certain temps pour décrypter ce que l'on a fait, et un programme en APL est parfaitement illisible, même par un initié; au contraire, un programme en Cobol peut ressembler à un roman-fleuve écrit en mauvais anglais, ce que l'on peut juger plus ou moins agréable, mais on en a vite assez de devoir écrire trois pages pour additionner deux nombres et écrire le résultat. Un autre inconvénient d'APL, et non des moindres, est que le jeu de caractères qu'il utilise ne se rencontre pas partout, ce qui réduit les possibilités d'implantation du langage, ou conduit à donner des équivalents en toutes lettres aux symboles impossibles à écrire.

Le mieux est donc de choisir un juste milieu: utiliser les caractères spéciaux quand ils permettent de constituer des symboles à peu près universellement admis et compris; se contenter du jeu de caractères dont on dispose; pour le reste, utiliser des symboles en toutes lettres, avec des possibilités d'abréviations, au moins pour les symboles trop longs tels que IDENTIFICATION ou ENVIRONNEMENT, mais sans aboutir non plus à des symboles tels que NXTRGT ou PLSTR, tout aussi indécryptables que ceux d'APL, et moins agréables à l'oeil.

Pour définir Euphémie, nous disposions de vingt-six caractères spéciaux, ce qui peut sembler confortable, mais certains d'entre eux sont difficilement monnayables (? F @ ¢) et nous ne les avons pas utilisés; les signes ¬] & ont été réservés pour les trois opérations booléennes classiques, non encore réalisées. Le reste a été utilisé au mieux, et sert à composer uniquement des opérateurs et des délimiteurs.

En ce qui concerne les symboles formés de lettres, on peut choisir encore entre deux possibilités: écrire ces symboles exactement comme des identificateurs, en les considérant ou non comme des mots réservés, ou les en distinguer en les encadrant entre deux caractères speciaux, en général des apostrophes. Le premier système pose des problèmes syntaxiques parfois graves (risques d'ambiguïtés), n'améliore pas la lisibilité du programme, et restreint beaucoup les possibilités d'abréviations. C'est la raison pour laquelle nous avons adopté le second système, un peu fastidieux pour l'utilisateur masi beaucoup plus sûr; par conséquent l'espace n'a pas de signification syntaxique en Euphémie, contrairement à ce qui se passe dans des langages tels que PL/I [IBM8] ou AED [Ro64], caractéristiques de la première méthode.

La décision suivante à prendre concerne le formalisme des instructions; les deux caractéristiques importantes d'Euphémie à ce sujet sont les suivantes:
1°) - Seules deux instructions ne commencent pas par un symbole caractéristique, ce sont les instruction d'affectation classique et l'instruction de génération ou mise à jour, qui constitue une sorte d'instruction d'affectation graphique; ce point n'a d'ailleurs rien d'original.
2°) - Les délimiteurs redondants ont été systématiquement évités, sauf dans un cas.

Un délimiteur redondant est un symbole de base qu'un analyseur syntaxique ne fait que sauter, en se contentant de vérifier qu'il est là, et qui n'apporte donc aucune information syntaxique sur le contexte. Par exemple, en Algol, dans l'en-tête de déclaration de procédure:

```
procédure P (X); entier X;
```

on trouve deux délimiteurs redondants, qui sont la parenthèse fermante et le point-virgule qui la suit, puisque le symbole entier suffit à indiquer que commencent les spécifications. Le cas le plus fréquent délimiteur redondant se trouve dans à peu près tous les langages, et il ne nous a pas été possible de le supprimer car il est trop entré dans les moeurs: il s'agit de la parenthèse fermante ailleurs que dans une expression, et du crochet de tableau fermant dans le cas où il est distingué de la parenthèse. Pour prendre un exemple en Euphémie, l'écriture:

```
tableau entier X, Y de 10, Z, T de 20;
```

serait plus logique que celle qui est utilisée:

```
tableau entier X, Y (10), Z, T (20);
```

De même en PL/I, les paires de parenthèses sont légion, et l'utilisateur peut se demander pourquoi il doit écrire:

```
CLOSE FILE (SYSIN);
```

alors que:

```
CLOSE SYSIN;
```

suffirait, et que d'ailleurs il doit écrire:

```
GOTO DEBUT;
```

et non pas:

```
GOTO LABEL (DEBUT);
```

4.232 - Instructions de boucle et de décision.

Les instructions qui servent à l'expression des itérations (pour et finpour) et des conditions (si, sinon et finsi) présentent des caractéristiques assez spéciales, expliquées en détail aux paragraphes 4.134 et 4.135. Ces particularités proviennent de raisons qui seront plus complètement exposées au chapitre 6, mais que l'on

peut cependant résumer ici: le compilateur du langage Euphémie est incrémentiel, et le langage produit est interprété.

Le compilateur est incrémentiel, c'est-à-dire qu'une instruction doit pouvoir être analysée seule; pour que cela soit possible jusqu'au bout, il faut que la validité d'une instruction ne dépende pas des instructions qui la précèdent (ce qui imposerait un retour en arrière) ou qui la suivent (et qu'on ne connaît pas encore). Il ne faut donc pas qu'une instruction si ne puisse être validée que par l'apparition un peu plus tard d'une instruction finsi. On peut rapprocher ce problème des particularités syntaxiques du langage POP-2 [Po68], lui aussi compilé de façon incrémentielle, et qui pousse cela jusqu'au système.

Comme l'instruction si et l'instruction finsi ont cependant un lien évident entre elles, il faut établir celui-ci à un moment quelconque: si on ne le fait pas à la compilation du programme, il faut le faire à son exécution. Cela est rendu possible par le fait que le compilateur produit un langage intermédiaire qui est ensuite interprété; l'utilisation d'une pile suffit à résoudre les problèmes de retrouver l'emplacement de l'instruction que l'on termine, et éventuellement en terminer plusieurs à la fois. Comme l'interprétation des expressions arithmétiques nécessite elle aussi l'utilisation d'une pile, et que l'on peut prendre la même, ce système ne complique pratiquement rien, et on peut encore une fois le rapprocher de celui qui est pratiqué dans le langage POP-2. Il en résulte cependant quelques complications du fait que l'on ne peut pas interpréter les instructions que l'on saute, si bien qu'une suite très complexe d'instructions de décision, d'itération et de branchements mélgangées peut conduire à des résultats inattendus. Ce point peut sembler peu important, à côté des avantages de cette particularité syntaxique au moment de la composition du programme par l'utilisateur.

4.233 - Séparation des déclarations.

Les problèmes posés par les déclarations sont importants et délicats: faut-il admettre des déclarations implicites? faut-il prévoir une structuree de blocs? les déclarations doivent-elles être statiques ou dynamiques? Les réponses données à ces questions résultent de plusieurs considérations: le langage Euphémie n'est pas destiné à des traitements numériques intensifs; les types de variables sont nombreux, et il existe certaines imbrications (entre objets et figures par exemple); enfin et surtout, la compilation est incrémentielle et conversationnelle.

La dernière considération conduit, par mesure de simplification, à rejeter la structure de blocs, qui n'a d'ailleurs rien d'indispensable. Les autres considérations conduisent à concevoir des déclarations explicites, entièrement statiques, et complètement séparées des instructions, ce qui simplifie aussi bien l'écriture du programme par l'usager que sa compilation incrémentielle, et n'introduit que peu de restrictions d'utilisation. Le problème reste celui des tableaux, qui ne peuvent dans l'état actuel du langage avoir de bornes variables; il s'agit d'ailleurs d'un sujet à revoir entièrement dans une extension possible du langage Euphémie, l'usage des tableaux étant dans la version présente particulièrement restreint.

Les problèmes de compilation que posent les déclarations seront exposés au chapitre 6.

4.24 - EXTENSIONS POSSIBLES A EUPHEMIE.

Le langage Euphémie tel qu'il est à présent est très évidemment incomplet: je vais donner pour terminer ce chapitre 4 une liste non exhaustive des extensions que l'on peut prévoir s'il est possible de continuer à travailler dessus. Le côte graphique du langage étant celui qui a été le plus étudié, c'est uniquement du côté des aspects non graphiques que l'on peut pour l'instant prévoir quelque chose. Je donnerai chaque fois des exemples de la forme extérieure que peut prendre l'instruction ou la notion à ajouter.

Parmi les instructions simples qui manquent évidemment, on peut signaler:

- l'instruction d'affectation multiple:

```
I, T (N), X := 40;
sélection TEXTE longueur N, CHAINE := "ZUT";
```

- l'instruction pour avec énumération de valeurs:

```
pour I := 1, 2, 3;
pour T (I) := 1 à N, M pas 7 à K, 10;
```

- une instruction qui équivale à sinon suivi de si et supprime la nécessité de finsi en cascade:

```
si I = 6;
...
ousi I = 4;
...
finsi;
```

- des instructions d'entrée et sortie, qui sont déjà supposées exister dans le deuxième exemple du paragraphe 4.175; une forme très simple suffirait sans doute:

```
lirefichier 1 dans CHAINE;
écrirefichier 2 depuis "FIN DU FICHIER";
lirefichier N dans I enregistrement P;
```

Du côté des variables, la carence la plus évidente est du côté des tableaux; il faudrait ajouter:

- des tableaux à plusieurs dimensions et à borne inférieure explicite:

```
tableau entier T1, T2 (- 4 à 20), réel PX (- 10 à 10), PY (20,
    0 à 5);
```

- des tableaux de chaînes:

```
tableau chaîne 10 caractères TC1, TC2 (54), 28 caractères HOP
    (- 4 à 7, 10);
```

- des tableaux d'étiquettes:

```
tableau étiquette A (10) := (E4, E4, E2, E6, E6);
A (5) := E1;
allerà A (I);
```

- des tableaux de variables graphiques:

```
tableau point P (22) figure F, segment S, T (- 4 à 0) figure Z;
P (I) <== X (I), Y(I);
```

- d'autre part, il serait utile de disposer de variables booléennes, ou au moins d'expressions booléennes utilisant la négation, l'union et l'intersection.

Le gros point qui reste est alors celui des procédures; celles-ci devraient pouvoir être écrites soit en Euphémie, soit dans un autre langage, ce qui implique qu'elles puissent être extérieures au programme qui les appelle. Il n'est d'ailleurs pas certain qu'il soit nécessaire de pouvoir écrire des procédures internes, l'exemple du langage Fortran montrant qu'il est possible de s'en passer: cela aurait l'avantage de simplifier fortement les aspects syntaxiques du corps de procédure. Je ne veux en tout cas considérer ici que la forme de l'appel de procédure.

L'idee qui semble la meilleure, et qui est d'ailleurs mise en application dans la description du système Eulalie donnée en langage Euphémie en annexe 4, consiste à prévoir que la déclaration d'une procédure revient à ajouter temporairement une nouvelle instruction au langage, avec le système habituel de mots-clés significatifs et de valeurs par défaut. Cette méthode, qui conduit en quelque sorte à un langage extensible, présente l'avantage primordial de donner une forme simple et parlante à l'appel d'une procédure à nombreux paramètres.

La déclaration de la procédure doit préciser le type de chaque paramètre, la forme des mots-clés, la définition de valeurs

initiales, et même la définition de valeurs d'options, comme le montre l'exemple suivant:

```
procédure chaîne BARATIN entier LONGUEUR option MODE (ABSOLU =
    1, RELATIF = 2, INCREMEN = 3) étiquette FIN := ERREUR réel
    EN (4) texte TEXTE chaîne QUOI := "__________";
```

Dans le corps de la procédure, on utilise les identificateurs qui apparaissent dans l'en-tête comme nom des paramètres correspondants, le nom de la procédure elle-même servant à désigner le ou les paramètres qui suivront immédiatement l'appel de la procédure, sans mot-clé les précédant. Si l'on écrit l'instruction d'appel suivante:

```
baratin "DONNEZ LA DATE" texte HOP mode absolu en 0, 0, 10, 10;
```

les paramètres prendront les valeurs:

```
BARATIN = "DONNEZ LA DATE"
LONGUEUR indéfini (ceci doit pouvoir être reconnaissable)
MODE = 1 (traduction d'une option)
FIN = ERREUR (valeur par défaut)
EN = (0, 0, 10, 10) (c'est un tableau de quatre éléments)
TEXTE = HOP
QUOI = "__________"
```

Ce systeme présente encore quelques problèmes, notamment dans l'accès aux paramètres écrits sous certaines formes, mais il n'est pas plus difficile à traiter par le compilateur et l'interprète que les instructions graphiques normales. Si la procédure BARATIN définie ci-dessus correspond en fait à un sous-programme écrit en Fortran, l'en-tête de celui-ci pourrait s'écrire:

```
SUBROUTINE SUBUN(BARATI,LONG,MODE,FIN,EN,TEXTE,QUOI)
```

le corps de la procédure dans le programme en Euphémie étant:

```
fortran SUBUN;
```

On peut aussi mettre la déclaration de la procédure sous la forme:

```
fortran SUBUN chaîne BARATIN entier ...
```

Le premier identificateur nomme le sous-programme extérieur, le deuxième nomme la procédure et le premier paramètre; le symbole fortran peut être remplacé par euphémie ou assembleur par exemple.

5 - LA PROGRAMMATION GRAPHIQUE CONVERSATIONNELLE.

Les deux chapitres précédents nous ont conduits assez loin dans l'étude des langages de programmation qui permettent d'utiliser un terminal graphique. Mais un langage de programmation n'est pas une fin en soi, il faut le mettre entre les mains des utilisateurs par l'intermédiaire d'un compilateur, lequel doit être inclus dans un système qui permette de le faire fonctionner. J'appellerai pour simplifier programmation graphique l'ensemble de ce travail dont le but est de résoudre un problème grâce au terminal graphique, et qui nécessite l'utilisation d'un langage de programmation, du compilateur associé et d'un système d'exploitation. Le présent chapitre va étudier un cas particulier de programmation graphique, qui est celui où l'homme et la machine mènent un véritable dialogue, et que j'appellerai programmation graphique conversationnelle. Le chapitre suivant décrira un système de programmation graphique conversationnelle qui permet le travail dans le langage Euphémie, c'est-à-dire le système Eulalie.

5.1 - NATURE ET INTERET DU DIALOGUE.

5.11 - NECESSITE DU TRAVAIL CONVERSATIONNEL.

Un exemple simple et réel nous montrera rapidement l'intérêt du travail conversationnel, ou plutôt l'intérêt que lui portent les utilisateurs d'un terminal graphique. Il s'agit de la situation qui règne à l'Institut de Mathématiques Appliquées de Grenoble, et à laquelle le système Eulalie devrait en principe pouvoir remédier. L'ordinateur IBM 360 modèle 67, auquel est connecté le terminal graphique IBM 2250, est utilisé de deux façons différentes: six heures par jour en moyenne, il fonctionne avec le système en temps partagé CP/CMS [IBM5], pour une trentaine de terminaux classiques fonctionnant comme des machines à écrire; le reste du temps, il fonctionne, en liaison avec un deuxième ordinateur IBM 360 modèle 40, avec le système classique mais multiprogrammé OS/360, version MVT [IBM4].

Les utilisateurs du terminal graphique ont à leur disposition les mêmes outils programmés dans l'un et l'autre des deux systèmes. Ils peuvent ainsi travailler soit en langage d'assemblage, avec quelques appels de sous-programmes qui résolvent uniquement les problèmes

posés par les opérations d'entrée et sortie [IBM7], soit en Fortran, en utilisant l'ensemble de sous-programmes G.S.P. [IBM2] dont j'ai parlé au chapitre précédent. Les inconvénients de ces deux softwares sont les mêmes avec les deux systèmes; or, le terminal n'est utilisé que pendant les six heures où fonctionne le système conversationnel CP/CMS, et ne sert absolument à rien pendant le reste du tenps. Devant une utilisation aussi catastrophique d'un instrument particulièrement coûteux, il est bon de réfléchir sur les causes de la situation: elles tiennent purement et simplement à la notion de programmation conversationnelle, et les utilisateurs préfèrent ne travailler sur le terminal que deux ou trois heures par semaine, à condition de pouvoir le faire ainsi (il est certain que la situation serait fondamentalement différente s'il ne s'agissait pas de programmer le terminal, mais d'utiliser un système existant et bien au point.)

Supposons donc un utilisateur quelconque qui s'aperçoit que le terminal graphique pourra lui faciliter la résolution d'un certain problème, et prenons-le après qu'il se soit aperçu qu'il devrait programmer son problème à un niveau assez élémtaire, qu'il devrait pour cela l'écrire en Fortran et apprendre, outre peut-être ce dernier langage, le langage G.S.P. qui est tout de même assez difficile; supposons-le donc prêt à écrire son programme au propre, après l'avoir conçu au brouillon au prix d'un travail dont la durée ne nous intéresse pas ici, et voyons ce qu'il doit faire suivant qu'il désire utiliser le système CP/CMS ou le système OS-MVT.

Avec le système OS-MVT, auquel est fidèle M. Dupont:
- recopie du programme sur des feuilles spéciales: 10 mn;
- attente des feuillles dans un bac "départ perforation": 12 h;
- perforation et verification des cartes: 10 mn;
- attente des cartes dans un bac "entrée moniteur": 12 h;
- lecture des cartes par l'ordinateur: 10 s;
- attente du travail dans la file d'attente sur disques: 1 h;
- travail du programme (compilation, chargement, exécution): 1s;
- travail de M. Dupont avec son programme, si l'on arrive jusque là, si M. Dupont est là, et avant la première erreur qui arrête tout: 1 s;
- impression d'une analyse de la mémoire: 1 mn;
- sortie des cartes et des résultats: 1 h;
- étude de l'analyse de mémoire par M. Dupont, corrections au programme et retour à la première étape: 20 mn, ou moins si l'erreur est particulièrement triviale.

Au total, M. Dupont a travaillé pendant trente minutes plus une seconde, son programme pendant deux secondes, et il y a eu plus de

vingt-quatre heures d'attente. Un tel délai pourrait être raccourci par divers moyens, mais ceux-ci correspondraient toujours à demander plus de travail à M. Dupont, et à moins bien utiliser l'ensemble de l'installation. Il est bien évident que pendant ce temps d'attente M. Dupont peut faire autre chose, mais il n'en reste pas moins que le travail productif sur le problème précis à résoudre est fait par à-coups, et s'étale sur une période beaucoup trop longue. D'autre part, M. Dupont n'a guère de moyens de connaître, et encore moins de choisir, l'heure à laquelle son programme passera en machine, ce qui présente quelques inconvénients puisqu'il devra être là à ce moment précis.

Pour M. Durand, qui travaille avec le système CP/CMS, en tulisant le terminal de type machine à écrire placé à côté du terminal graphique, les opérations sont les suivantes:
- frappe du programme: 20 mn (M. Durand tape à la machine avec seulement les index);
- compilation, chargement, exécution du programme, début de travail avec M. Durand: 10 s (l'ordinateur semble tourner plus lentement qu'avec le système OS, car il sert plus d'utilisateurs à la fois);
- examen du programme en mémoire pour déterminer les causes d'erreur: 5 mn;
- corrections au programme et retour à la deuxième étape: 5 mn.

Au total, si M. Durand s'est réservé une heure d'utilisation du terminal, il peut faire tourner son programme au moins cinq fois sinon plus, en avançant notablement son travail à chaque fois. Il aura peut-être dû attendre trois jours cette heure d'utilisation, mais cela en vaut indiscutablement la peine, et il est certain qu'au bout d'une semaine ou deux il sera notablement gagnant par rapport à M. Dupont.

Tout ceci n'est pas très nouveau, et beaucoup d'utilisateurs se contentent ou se satisfont de systèmes non conversationnels. Là où les contraintes que cela impose deviennent positivement insupportables, c'est quand il s'agit d'utiliser de cette façon un dispositif qui est fondamentalement conversationnel, c'est-à-dire le terminal graphique. Quand les utilisateurs peuvent comparer les deux modes de travail, ils préfèrent ne travailler sur le terminal qu'une heure ou deux par semaine, mais le faire de façon conversationnelle.

5.12 - NATURE DU TRAVAIL CONVERSATIONNEL.

On a pris l'habitude d'appeler conversation ce que l'on devrait plutôt appeler dialogue, étant donné qu'il n'y a que deux interlocu-

teurs: l'homme et la machine. A partir du moment où chacun des deux partenaires peut recevoir l'information transmise par l'autre et lui en transmettre, on pourrait dire qu'il y a conversation ou dialogue; par conséquent, le travail de programmation à l'aide d'un système classique pourrait être considéré comme conversationnel. En fait, il faut ajouter une nouvelle condition pour qu'il y ait réellement un dialogue et non pas deux monologues parallèles: il faut que l'échange d'informations se fasse d'une façon régulièrement alternée, dans un sens et dans l'autre, chaque transmission d'information pouvant modifier la façon dont se déroulera la suite du dialogue.

Dans le cas du système séquentiel classique, il n'y a donc pas de conversation, parceque l'utilisateur prépare toutes ses répliques sans pouvoir connaître celles de son adversairte; si tout se passe comme prévu, le monologue de la machine s'ajustera bien avec le sien, mais la moindre discordance bloquera tout le système. On pourrait considérer qu'il n'y a qu'une différence d'échelle ou d'ordre de grandeur avec le travail au moyen d'un système en temps partagé: dans le premier cas, le dialogue se fait de façon macroscopique, au moyen de paquets de cartes entrelardés de cartes de contrôle du côté de l'utilisateur, au moyen de résultats entrecoupés de messages divers du côté du système; dans le deuxième cas, il se faitde façon microscopique, au moyen de messages dans les deux directions.

En fait, il y a là la même différence que lorsque l'on échange des informations avec un interlocuteur situé aux antipodes, entre le moyen des lettres qui prennent le bateau et celui du téléphone: il est parfaitement évident que l'urgence de l'information n'entre pas vraiment en ligne de compte; ce qui est important, c'est que chaque message d'un des partenaires peut avoir une influence instantanée sur les actions et les réponses de l'interlocuteur. C'est ce côté interactif ou de réaction réciproque entre deux individus capables de décision qui constitue l'aspect fondamental du travail conversationnel, à condition bien sûr que l'on n'oublie pas qu'il y a en effet un individu capable de décision dans le système avec lequel on dialogue, en l'occurence le ou les auteurs de ce système.

On peut pousser un peu plus loin la comparaison précédente: le coup de téléphone que j'envoie à Melbourne ou Santiago du Chili me coûte beaucoup plus cher que la lettre, et il ne me permet de transmettre qu'une quantité d'information beaucoup plus réduite; il peut de plus se produire plusieurs incidents fâcheux: j'oublie le problème du décalage horaire et réveille mon important interlocuteur à deux heures du matin, ou bien je n'obtiens que sa concierge, ou

bien il y a tant de parasites sur la ligne que la conversation est inaudible. De même, l'utilisation d'un système conversationnel peut me paraître financièrement moins rentable que celle d'un système classique, je dois m'astreindre à des horaires dont je ne suis pas maître, je peux perdre une heure à essayer de faire fonctionner un terminal récalcitrant, les erreurs de transmission détruisent tout mon travail, etc. Pourtant, on ne peut pas plus se passer de la programmation conversationnelle une fois qu'on a commencé à l'utiliser, que l'on ne peut se passer du téléphone. Il y a là quelque analogie avec la drogue, car sinon comment pourrait-on expliquer que les usagers se plient (avec ou sans récriminations) à n'importe quels horaires, ne sont découragés par aucun incident, et continuent de réclamer encore et toujours plus de temps partagé une fois qu'on leur en a fait goûter?

5.13 - DIALOGUE AVEC LE SYSTEME ET DIALOGUE AVEC LE PROGRAMME.

Je voudrais lever ici certaines ambiguïtés gênantes au sujet du travail conversationnel. La première est celle qui résulte de la confusion fréquente entre système conversationnel et système en temps partagé; le mieux est de définir clairement les termes utilisés, en commençant par la notion de simultanéité qui est ici nécessaire.

On parle de simultanéité quand deux opérations se déroulent en même temps; ceci peut n'être qu'une notion très relative, suivant l'échelle à laquelle on se place. Dans un système d'exploitation séquentiel, les utilisateurs peuvent très bien penser que leurs programmes se déroulent simultanément, puisqu'ils n'ont aucun moyen de connaître et de contrôler l'ordre dans lequel ils s'exécutent. Quand une opération d'entrée ou sortie se déroule en même temps qu'un programme, la simultanéité n'est vraie qu'à l'échelle macroscopique, puisque les accès à la mémoire ne se font qu'un à un. Il n'y a donc simultanéité vraie que quand plusieurs organes sont capables d'effectuer le même travail indépendamment les uns des autres. Il y a simultanéité apparente, la seule qui nous intéresse réellement, chaque fois que le résultat d'un certain travail ne dépend pas de l'ordre dans lequel s'effectuent les opérations élémentaires, et que cet ordre est choisi de façon arbitraire ou aléatoire.

On peut alors parler de partage de temps ou de temps partagé quand plusieurs personnes emploient simultanément les ressources d'un même ordinateur. Etant donné ce que je viens de dire de la simultanéité, l'utilisation d'un système séquentiel classique serait

déjà du partage de temps; on restreint donc la définition de ce terme au cas où la simultanéité, apparente ou réelle, procure un temps de réponse assez bas pour que chaque utilisateur puisse s'imaginer disposer de toute la machine à lui tout seul.

Il faut remarquer que dans la définition précédente n'intervient nulle part la notion de travail conversationnel. L'utilisation conversationnelle d'un gros ordinateur nécessite le temps partagé si l'on veut avoir un bon rendement et un temps de réponse acceptable pour plusieurs personnes en même temps, mais ce n'est plus vrai dès que l'on s'écarte un peu de ce cas (en particulier s'il y a un seul utilisateur conversationnel, comme c'est le cas avec le système Eulalie sous le contrôle du système OS/MVT). Un bon exemple de la disjonction entre ces deux notions est donné par le systeme CP/CMS [IBM5]: le système CP a pour but le temps partagé, mais il ne s'occupe absolument pas de savoir comment est utilisé le temps qu'il partage entre les machines virtuelles qu'il simule; le système CMS est conversationnel, mais il ignore totalement le temps partagé puisqu'il ne connaît qu'un seul utilisateur. Le fait de séparer ces deux fonctions fondamentales et de les faire traiter par des systèmes bien distincts clarifie notablement le problème.

Une dernière distinction est nécessaire, qui a d'ailleurs déjà été évoquée au dernier paragraphe du chapitre 2: il s'agit de savoir avec qui exactement dialogue l'utilisateur d'un système conversationnel. Deux cas peuvent se produire: ou bien le dialogue s'établit avec le système, c'est-à-dire avec un programme préexistant, supposé déjà au point, et fabriqué par une autre personne que l'utilisateur; ce dialogue se fait au moyen du langage de commande associé au système et défini par ses auteurs; ou bien le dialogue s'établit avec le programme que l'utilisateur vient lui-même d'écrire, et qui n'est pas forcément au point; dans ce cas, le dialogue se fait au moyen du langage de communication établi par le programme. La distinction entre ces deux formes de dialogue est particulièrement importante quand l'utilisateur converse avec un programme qui renferme des erreurs: dans cette situation en effet, il faut s'adresser parfois au programme, parfois au système si le programme réagit d'une façon non désirée (il répond de travers, ne dit rien quand on l'interroge, bavarde à tort et à travers sans qu'on puisse l'arrêter, etc.). Pour préciser cette distinction d'une dernière façon, on peut remarquer que la commande est le dialogue de l'utilisateur avec quelqu'un d'autre, et que la communication est le dialogue de l'utilisateur avec lui-même. Quand je converse avec un programme que j'ai écrit et mis au point il y a déjà assez longtemps, je peux considérer que mon programme est passé au rang de système, et son langage de communication au rang de

langage de commande, car j'ai oublié la majeure partie du fonctionnement interne dudit programme, pour ne plus me souvenir que de la manière de l'utiliser.

5.2 - LES SYSTEMES GRAPHIQUES CONVERSATIONNELS.

La notion de travail conversationnel étant supposée à peu près claire, on voit qu'elle peut figurer dans divers contextes, dont trois vont nous intéresser: le travail avec un système graphique conversationnel ne renferme aucune programmation, et le seul langage utilisé sera le langage de commande; le travail avec un système de programmation conversationnelle non graphique nous permettra de voir ce qui est réellement spécifique du dialogue entre l'homme et l'ordinateur; enfin, le travail sur un système de programmation graphique conversationnelle permettra de faire la fusion des propriétés des deux précédents, du moins au point de vue de la programmation, de la commande et de la communication.

5.21 - SKETCHPAD.

Le système Sketchpad [Su63] a été réalisé par I. E. Sutherland, au Massachusetts Institute of Technology, en 1963. Il date donc en quelque sorte de la préhistoire des traitements graphiques, et ses aspects particulièrement spectaculaires en ont fait un véritable monstre sacré, pour ne pas dire un minotaure, qui a fortement influencé de très nombreux travaux ultérieurs: beaucoup n'ont pas encore renoncé à essayer de refaire Sketchpad sur un nouveau système ou dans de nouvelles conditions, au lieu de tenter des réalisations originales et peut-être plus utiles.

Sketchpad est défini par son auteur comme un système de communication graphique entre l'homme et la machine, ou comme un outil de dialogue au moyen de dessins linéaires. Plus précisément, l'utilisateur de Sketchpad peut fabriquer et manipuler des dessins en conversant avec l'ordinateur, ce qui n'est pas tout à fait la même chose. Le matériel utilisé est un terminal à écran de faibles dimensions, connecté à un ordinateur TX-2, machine unique en son genre, asez puissante et très spéciale. Il n'y a pas de mémoire d'entretien: c'est donc l'ordinateur lui-même qui se charge de la régénération de l'image, mais il peut le faire en simultanéité avec des calculs. Les outils de communication comprennent un pointeur optique, qui est utilisé de façon intensive par Sketchpad, et de très nombreux boutons poussoirs, boutons rotatifs, leviers, touches,

voyants, etc., qui nécessitent une gymnastique importante de l'utilisateur, placé quasiment à l'intérieur de l'ordinateur.

Il n'y a pas de véritable langage de commandes, celles-ci étant envoyées au moyen de certaines touches; les opérandes sont le plus souvent fournis au pointeur optique, soit par désignation d'un objet existant (effacement, deplacement, rotation, etc.), soit par poursuite du pointeur quand il faut fournir des coordonnées (extrémité d'un vecteur, centre d'un cercle, etc.). Les variations de position et d'échelle se font sans envoi de commandes, par simple rotation de boutons, ce qui est rendu possible par le fait que l'ordinateur peut consulter ces boutons à chaque régénération de l'image.

Les schémas construits sont forcément imparfaits, puisqu'on ne fournit jamais de coordonnées numériques. On utilise pour les améliorer (géométriquement sinon esthétiquement) une des possibilités les plus importantes du système Sketchpad, qui consiste à indiquer quelles doivent être les propriétés géométriques et topologiques de l'image: rendez parallèles cette droite et celle-ci, liez ce point à ce segment, placez ce point sur ce cercle, etc. Après chaque modification à l'image, ou moins souvent si on le désire, une commande permet de demander au système de satisfaire les contraintes imposées par ces propriétés, dans la mesure du possible. Les figures construites peuvent être conservées pusi réutilisées comme des sous-programmes, chaque exemplaire étant caractérisé par des paramètres de position, d'échelle et d'orientation.

Par le jeu de nombreuses manipulations, rendues particulièrement spectaculaires par le fait qu'elles se font de façon continue (le centre d'un quadrilatère suit le déplacement du pointeur optique, un arc de cercle se construit pendant que je bouge la main, l'ensemble d'une figure tourne pendant que je la montre, le pointeur optique semble réellement être un "crayon lumineux", etc.), l'utilisateur peut donc construire assez rapidement des schémas quelconques. Pour l'ordinateur, le travail à fournir est très important, et Sketchpad s'appuie sur une structure de représentation des données extrêmement complexe, pour ne pas dire compliquée. Il s'agit d'une structure en anneau d'un style devenu classique, formée de blocs de mots fortement enchaînés les aux autres dans une hiérarchie très stricte. Cette structure doit être consultée à chaque manipulation de l'image, et certaines opérations sont si complexes que l'auteur a renoncé à les faire se dérouler de façon continue; il s'agit en particulier de l'opération de satisfaction des contraintes, qui nécessite des passages à la fois itératifs et récursifs dans la structure en anneau.

Une question intéressante à se poser est de savoir quelle est l'utilité de Sketchpad, non pas du point de vue de la recherche sur les traitements graphiques, mais pour celui qui veut s'en servir. Les applications que propose l'auteur lui-même sont les suivantes:
- Fabrication et modification de schémas; ceci ne fait que redire ce que précisément Sketchpad est capable de faire; le pourquoi de ces schémas n'est pas indiqué.
- Compréhension par l'ingénieur des opérations qui peuvent être décrites de façon graphique; ceci sert à justifier l'énorme complexité de la satisfaction des contraintes, mais il n'est pas certain que l'utilisation d'un jeu de Meccano ne permettrait pas de le faire aussi vite et de façon moins coûteuse.
- Utilisation comme moyen d'entrée topologique pour des simulateurs de circuits ou autres; cette application n'est que proposée car la communication avec des simulateurs n'est pas réalisée; d'autre part, une bonne partie de Sketchpad est parfaitement inutile dans ce contexte.
- Fabrication de dessins hautement répétitifs; cette application ne fait que monter en épingle un aspect particulier de la première, et on ne précise toujours pas ce que l'on peut faire avec ces dessins.

En fait, l'application qui me semble la plus nette m'est pas explicitement citée par l'auteur, mais elle apparaît fréquemment entre les lignes: Sketchpad est une excellente distraction pour celui qui l'utilise et pour les spectateurs, comme le signalent d'ailleurs certaines phrases d'articles de Sutherland: "Il fallut moins de cinq minutes pour fabriquer ce schéma, mais on s'amusa avec pendant plus d'une heure"; ou plus loin: "De même que l'on peut animer des liaisons, on peut faire pédaler un pantin à bicyclette, ou onduler la chevelure de Néfertiti". Au moment où l'on pense que l'auteur va présenter enfin une application pratique, concernant des schémas électroniques, on constate qu'après dix heures de travail avec le système l'utilisateur potentiel se décida à utiliser du papier et un crayon!

L'intérêt principal de Sketchpad me semble donc être qu'il a permis de définir et d'étudier certaines opérations fondamentales ou au moins importantes des traitements graphiques. Il s'agit à coup sûr d'un très intéressant et utile travail de recherche, mais son intérêt pratique me semble beaucoup moins certain. Une dernière critique importante concerne le matériel utilisé par Sketchpad; ce système s'appuie sur un ordinateur assez puissant, qui ne sert qu'à lui quand il fonctionne; il fait un usage intensif d'outils de communication très particuliers; enfin, ses aspects les plus spectaculaires ne sont rendus possibles que par le fait qu'il n'y a pas de mémoire d'entretien. On peut tirer de cela les conclusions sui-

vantes: si l'on tient absolument à reproduire Sketchpad, il faut disposer d'un gros ordinateur pour soi tout seul: en effet, les modifications continues de l'image ne sont réellement possibles que si c'est l'ordinateur lui-même qui se charge de son entretien, et si ces modifications dépendent de lois complexes, en particulier de la satisfaction de contraintes, un petit ordinateur ne peut absolument pas suffire. Toutes les possibilités les plus intéressantes du système Sketchpad sont donc incompatibles avec le temps partagé.

5.22 - GENIAL.

Le système GENIAL [Lu68-2, Lu70] a été construit par Michel Lucas, à l'Institut de Mathématiques Appliquées de Grenoble, de 1968 à 1970, et il est encore susceptible de quelques extensions. Ses buts étaient à l'origine d'essayer de refaire ce qu'il était possible du système Sketchpad, sur un matériel beaucoup moins puissant que le TX-2; son utilisation principale est l'enseignement des techniques graphiques aux étudiants et la fabrication de dessins animés.

Le matériel utilisé peut être qualifié de rudimentaire, en particulier si on le compare avec celui qu'emploie Sketchpad. Là encore, le terminal graphique est commandé directement par l'ordinateur, un PDP-8, mais celui-ci est beaucoup moins puissant que le TX-2, et les performances de l'ensemble sont telles que l'on ne peut afficher plus d'un millier de points sans voir l'image clignoter. Les possibilités arithmétiques du PDP-8 sont extrêmement réduites, puisqu'il ne sait pas faire de multiplications ni de divisions (sauf si l'on écrit les sous-programmes correspondants). Les outils de communication sont réduits à un pointeur optique et un télétype.

Il n'existe aucun dispositif pratique qui puisse jouer le rôle d'un clavier de fonctions, aussi travaille-t-on au moyen d'un langage de commandes d'un type assez classique, frappé sur le télétype. Les opérandes des commandes peuvent être donnés sous forme numérique (coordonnées, numéro de figure) ou au moyen du pointeur optique, encore une fois par désignation ou par poursuite du pointeur pour envoi de coordonnées.

Comme avec Sketchpad, on peut construire et utiliser des figures, mais il intervient alors deux différences fondamentales. D'une part, on ne peut pas déformer une figure, ni même la faire tourner dans le plan de l'écran. D'autre part, on peut au contraire animer les objets d'une figure, non pas comme comme dans Sketchpad par manipulations demandées à l'utilisateur (le mouvement s'arrêtant dès

que l'on ne fait plus rien), mais par description dans la figure elle-même des mouvements à appliquer. Les mouvements peuvent être décrits comme des translations rectilignes et uniformes, alternatives et continues, ou comme des parcours le long de chemins décrits par points ou avec des figures (arcs de cercles, polygones, chaînes de caractères). Comme d'autre part on peut combiner ces mouvements entre eux, on peut arriver à des animations extrêmement complexes. On peut enfin combiner les affichages de figures, animées ou non, en séquences de durée déterminée qui se succèdent dans l'ordre voulu et composent un véritable film.

Le langage de commande de GENIAL est d'une utilisation très classique et linéaire, et d'une forme assez rudimentaire. La structure de données sur laquelle s'appuie le système est également simplifiée; elle utilise principalement des tables à une seule dimension, et ignore la gestion dynamique de la mémoire. L'animation des figures est faite par la modification des coordonnées à chaque cycle de régénération, suivant les paramètres fournis lors de la définition du mouvement.

La comparaison entre les possibilités de Sketchpad et celles de GENIAL est assez intéressante: les facilités de dessin de Sketchpad sont si complètes et si spectaculaires qu'elles ont fini par constituer une fin en elles-mêmes, et que l'on n'a pas jugé utile d'aller plus loin; d'autre part, le travail nécessaire à la construction de ce qui existe était loin d'être mince. Au contraire, avec GENIAL, les possibilités de dessin pur ont été assez facilement réalisées, mais on en a très vite fait le tour; étant données les limites étroites imposées par le PDP-8 et le terminal, on ne peut faire que des schémas simples, et on s'aperçoit vite que cela ne mène pas extrêmement loin: c'est ce qui a conduit à étendre les possibilités du système dans une direction nouvelle et originale, l'animation des dessins.

Une autre comparaison intéressante concerne les moyens de dialogue avec le système: d'un côté, Sketchpad, avec ses nombreux outils très diversifiés, jouant tous plus ou moins le rôle de commandes, mais d'une façon mal définie, et nécessitant une dexterite certaine de la part de l'utilisateur (l'auteur est en particuleir muet sur la détection et la correction des erreurs de manipulation); de l'autre côté, GENIAL, avec en tout et pour tout un télétype et un pointeur optique, mais un langage de commandes bien défini et assez restreint, des règles de travail très précises, et des moyens simples de détecter et de signaler les erreurs.

Pour conclure, cette comparaison que je viens de faire entre Sketchpad et GENIAL me semble bien illustrer les avantages de la définition précise d'objectifs raisonnables et limités, mais qui ne ferment pas la porte à des adjonctions ultérieures.

5.3 - LES SYSTEMES DE PROGRAMMATION CONVERSATIONNELLE.

Un système de programmation conversationnelle a pour fonction principale de permettre à ses usagers la création et l'utilisation de programmes. Il ne s'agit donc pas, comme dans le cas précédent, de fournir des programmes déjà tout faits, et l'utilisateur aura un peu plus de travail à faire pour obtenir des résultats; en contrepartie, il pourra adapter ses programmes à ses propres besoins, et ne devra pas s'astreindre à faire cadrer tant bien que mal ses problèmes particuliers avec ceux que peut résoudre le système.

Le travail sur un programme précis se déroule généralement en trois grandes phases: composition et modification du programme en langage source, compilation et exécution. On peut distinguer trois catégories de systèmes suivant qu'ils confondent ou non certaines de ces phases; pour les systèmes à compilation différée, tels que CP/CMS [IBM5], T.S.S./360 [IBM15], les trois phases sont complètement séparées; pour les systèmes à compilation immédiate, tels que Diamag 2 [SBV68, Ve68], RAX [IBM6], BRUIN [BU..], POP-2 [Po68], les phases de composition ou modification et de compilation sont confondues; enfin, pour les systèmes dits "machines de bureau", qui sont généralement les mêmes que les précédents, l'exécution de chaque instruction peut suivre immédiatement sa frappe et sa traduction.

Les grandes fonctions que doit assurer un système de programmation conversationnelle sont les suivantes:
- Composition et modification des programmes: ce travail, que par défaut de traduction on appelle édition, est fait soit par un module spécial du système, le programme éditeur, pour les systèmes à compilation différée, soit par le compilateur lui-même en cas de compilation immédiate ou de fonctionnement en machine de bureau.
- Conservation des programmes: un utilisateur doit pouvoir travailler sur plusieurs programmes à la fois, et ne pas avoir à les réintroduire chaque fois qu'il veut les exécuter; étant donné que les modifications sont fréquentes, et qu'elles peuvent faire varier la taille des programmes, on doit conserver les fichiers correspondants sur des dispositifs à accès direct (généralement des disques) où l'on organise l'information suivant une structure de listes.

- Compilation des programmes: ce travail est fondamentalement différent suivant que la compilation est différée ou immédiate; dans le premier cas, on utilise les méthodes de compilation classiques et bien connues, et on peut en général utiliser des compilateurs existants, réalisés pour des systèmes séquentiels non conversationnels; ceci explique que la compilation différée concerne beaucoup plus de langages que la compilation conversationnelle. Dans le deuxième cas au contraire, le compilateur doit avoir été conçu entièrement pour être conversationnel, avec des méthodes de compilation nouvelles et très élaborées; si, chaque fois qu'une instruction est frappée, le compilateur la traduit entièrement, ce qui en rend possible l'exécution immédiate, on dit que la compilation est incrémmentielle; l'application de ce procédé aux langages de programmation classiques est très difficile, et l'on trouve souvent plus commode de définir un nouveau langage de programmation, spécialement étudié pour cela; c'est en particulier toujours le cas dans les systèmes du type "machine de bureau".

- Exécution des programmes: cette fonction est la seule caractéristique fondamentale des systèmes de programmation par rapport aux autres systèmes: ils permettent d'exécuter des programmes fournis par les utilisateurs; cette exécution nécessite l'inclusion de sous-programmes, elle peut aussi nécessiter l'interprétation du programme plutôt que son exécution directe; cette dernière méthode est particulièrement adaptée à la compilation incrémentielle et à l'exécution immédiate.

- Mise au point des programmes: la principale implication de la possibilité d'exécution des programmes des utilisateurs est que les programmes exécutés renfermeront neuf fois sur dix des erreurs; le système doit donc fournir, d'une part des moyens de détecter ces erreurs en se protégeant contre elles, d'autre part des moyens pour l'utilisateur de corriger ces erreurs; c'est ce que l'on inclut dans l'ensemble dit de mise au point; l'interprétation du programme facilite particulièrement la réalisation de cette fonction, puisque en fait le programme de l'utilisateur ne prend jamais réellement le contrôle.

- Gestion des programmes: le travail d'un utilisateur sur plusieurs programmes se traduit finalement par la création et la manipulation de nombreux fichiers; le système doit fournir des moyens de les gérer dans leur ensemble, sans considération de leur contenu; il faut au moins pouvoir consulter la liste des fichiers ou des programmes, détruire un fichier ou le créer en en copiant un autre; il peut être intéressant d'avoir aussi des possibilités de concaténation ou de fractionnement de fichiers, bien que ce ne soit pas indispensable.

- Gestion des utilisateurs: ces derniers peuvent être assez nombreux, et travailler chacun sur plusieurs dizaines de programmes

ou de fichiers; ceci impose la reconnaissance poar le système d'un utilisateur au moyen d'un nom qui le désigne et d'un mot de passe qui évite les fausses manoeuvres, volontaires ou non; à cela doivent correspondre les possibilités d'introduire auprès du système de nouveaux utilisateurs, d'en retirer d'autres en conservant ou non leurs programmes, de comptabiliser le temps de travail de chacun, etc.

Il n'entre pas dans mon propos d'étudier les caractéristiques, les qualités et les défauts des différents systèmes de programmation conversationnelle cités plus haut. Leur existence me permet simplement de dégager les sept fontions fondamentales de la programmation conversationnelle: la composition des programmes - leur conservation - leur compilation - leur exécution - leur mise au point - leur gestion - la gestion des utilisateurs. Toutes ces fonctions doivent se retrouver si le système devient en plus orienté vers la programmation graphique.

5.4 - LES SYSTEMES DE PROGRAMMATION GRAPHIQUE CONVERSATIONNELLE.

Un système de programmation graphique conversationnelle présente avec les systèmes graphiques conversationnels tels que Sketchpad ou GENIAL la différence majeure que c'est un système de programmation, c'est-à-dire qu'il n'est pas fait pour résoudre une application particulière, mais pour permettre à ses utilisateurs de résoudre comme ils l'entenent leurs propres problèmes. Il a donc en fait beaucoup plus de rapports avec les systèmes de programmation conversationnelle, mais les différences sont là encore notables: un système de programmation graphique conversationnelle est fait pour permettre la composition et l'exécution de programmes graphiques, c'est-à-dire de programmes qui utilisent un terminal graphique; étant donné que ledit terminal est un dispositif éminemment conversationnel, il est à peu près évident qu'un tel système utilisera comme moyen de dialogue le terminal lui-même, et non pas un dispositif quelconque du type d'une machine à écrire.

Une deuxième différence importante est que, dans la majeure partie des cas, les centres de calcul ne disposent que d'un seul terminal graphique, quand ils ont un: le systeme n'a donc aucun besoin de servir plusieurs utilisateurs à la fois; en revanche, il est bon qu'il puisse s'insérer dans un système multi-programmé, sauf si l'ordinateur utilisé est très petit. Une dernière remarque importante est que le système n'a pas besoin de fournir de nombreux langages de programmation: un seul suffit, à condition qu'il soit bien adapté aux traitements graphiques.

Un système de programmation graphique conversationnelle est donc un sous-système utilisable dans un cadre de multi-programmation ou de temps partagé, et qui présente les caractéristiques suivantes:
- Il permet la composition, la compilation, l'exécution, la mise au point et la conservation de programmes.
- Ces programmes utilisent principalement le terminal graphique, et sont donc écrits dans un langage de programmation graphique associé au système.
- Le travail de l'utilisateur avec le système se fait entièrement avec le terminal graphique, du début jusqu'à la fin.

Sans aller plus loin dans la définition d'un tel système , je peux signaler tout de suite qu'à ma connaissance il n'existe pas, tout au moins à l'heure actuelle, de système de programmation graphique conversationnelle. Un système tel que C.I.C. [LTD69], par exemple, est un système de programmation conversationnelle qui utilise comme terminal un terminal graphique, mais les programmes qu'il permet de composer ne sont pas graphiques. Un système tel que GULP [Pa68] ne permet pas vraiment de composer des programmes, mais de fabriquer un petit système avec lequel on pourra converser: il s'agit en quelque sorte de la construction d'un langage de communication.

Les trois caractéristiques que je viens de définir ont quelques implications:
- Si l'on veut réellement faire de la programmation graphique, il faut qu'au moment de l'exécution des programmes le système se fasse le plus petit possible: en particulier, il ne faut pas qu'il se réserve une portion des outils de communication, ou, ce qui serait encore pire, une portion de l'écran. Par conséquent, quand un programme s'exécute, il doit avoir le contrôle complet de toutes les possibilités du terminal.
- La programmation graphique est toujours délicate et complexe, ce qui signifie que les erreurs sont très fréquentes; un système qui ignorerait systématiquement toutes les erreurs serait tout aussi catastrophique qu'un autre qui arrêterait le programme à chaque erreur. Il faut en fait que ce soit le programme lui-même qui puisse entièrement contrôler tous les cas litigieux, pour ignorer l'erreur, ou tenter de la corriger, ou annuler le traitement correspondant, ou tout autre chose.
- Comme il faut cependant laisser à l'utilisateur la possibilité d'intervenir pendant le déroulement de son programme, indépendamment des procédés de communication qui ont pu être prévus, il faut réserver au système un moyen d'interruption qui permette à tout moment de lui rendre le contrôle. Ceci sera particulièrement utile

dans le cas d'un programme qui tourne en rond, ou qui refuse de reconnaître les procédés de communication que l'on a prévus.

Ces quelques considérations mises à part, on peut revoir ce que deviennent dans un système de programmation graphique conversationnelle les sept fonctions principales dégagées au paragraphe précédent:

- Composition et modification des programmes: ceci doit se faire à l'aide du seul terminal, c'est-à-dire que les textes sont frappés sur le clavier alphanumérique et apparaissent sur l'écran; on peut tirer parti du fait que cela permet de voir une bonne partie du programme d'un seul coup; on trouvera en [Ad-70] des considérations sur ce problème dans le cas le plus général.
- Comservation des programmes: il n'y a ici rien de particulier, à part le fait que l'édition au moyen du terminal nécessite que l'on puisse accéder à d'assez gros morceaux des programmes d'une manière rapide.
- Compilation des programmes: le dialogue au moyen du terminal graphique est très serré, aussi est-il à peu près obligatoire que la compilation soit conversationnelle; comme on n'utilise qu'un seul langage, spécialisé dans les traitements graphiques, on n'a qu'un compilateur à construire, et on peut d'ailleurs s'arranger dans la définition du langage pour que sa compilation conversationnelle et incrémentielle soit relativement facile; une autre remarque importante est que cette compilation doit être la plus complète possible, pour éviter qu'un nombre important d'erreurs ne soient détectées qu'au cours de l'exécution.
- Exécution des programmes: il n'y a aucun besoin qu'elle soit immédiate, car cela signifierait que l'on devrait voir sur l'ecran à la fois les instructions du programme et les résultats qu'elles produisent; on n'a pas besoin de performances extraordinaires, car les réactions de l'utilisateur humain seront toujours plus lentes que celles de son programme; l'interprétation semble donc la meilleure solution, avec en plus l'avantage qu'elle est d'ordinaire plutôt économique du point de vue de la place en mémoire, quantité dont sont friands les traitements graphiques.
- Mise au point des programmes: il faut évidemment utiliser au mieux le terminal graphique lors des opérations de mise au point; comme d'autre part on travaille dans un langage de programmation assez éloigné de la machine, il faut que cette mise au point se fasse de façon symbolique, c'est-à-dire à l'aide de tous les noms utilisés dans le programme, et non pas avec des adresses absolues et des valeurs hexadecimales, octales ou autres; il est certain que l'exécution interprétative facilite énormément tout le travail; à part cela, les méthodes elles-mêmes sont encore très mal connues à l'heure actuelle, et il m'est difficile de préciser davantage ce

point, qui est certainement une direction de recherche très ouverte.
- Gestion des programmes: le seul point particulier est que ces programmes sont tous écrits dans le même langage, ce dont on peut sans doute tirer parti dans la conception de leur mode de rangement sur un dispositif à accès direct.
- Gestion des utilisateurs: il n'y a pas deux utilisateurs au même moment s'il n'y a qu'un seul terminal, mais les utilisateurs se succèdent dans le temps; il faut donc assurer les fonctions citées plus haut, avec l'énorme simplification qu'on ne fera jamais qu'une seule chose à la fois.

Je viens donc d'établir en quelque sorte le cahier des charges d'un système de programmation graphique conversationnelle; le chapitre suivant présentera un système qui s'efforce de satisfaire à ces conditions. C'est le système Eulalie, centré sur le langage de programmation graphique Euphémie, dont j'ai parlé au chapitre précédent.

6 - LE SYSTEME EULALIE.

Le système Eulalie est le système de programmation graphique conversationnelle destiné à permettre l'utilisation du langage Euphémie, décrit au chapitre 4, sur le matériel constitué par le terminal graphique IBM 2250 de modèle I (ou III) connecté à un ordinateur IBM 360. Le systeme Eulalie est en fait un sous-système, qui peut être considéré pour une exploitation normale comme un programme d'utilisateur classique mais conversationnel; il peut fonctionner sous contrôle des systèmes d'exploitation OS/360 (options PCP, MFT ou MVT) ou CMS (en système autonome ou sous contrôle du système CP); il nécessite sur un dispositif à accès ditect, en général une pile de disques 2311 ou 2314, une place qui dépend du nombre d'utilisateurs, du nombre et de la taille des programmes qu'ils utilisent: 20 cylindres sur une pile de disques 2314 sont déjà une quantité suffisante pour une demi-douzaine d'utilisateurs travaillant de façon normale, ou une vingtaine travaillant peu.

Ce chapitre est divisé en deux parties: la première permet à l'utilisateur potentiel d'apprendre à utiliser le système Eulalie, tandis que la deuxième expose quelques-uns des principes de conception et de fonctionnement de ce système.

6.1 - MANUEL D'UTILISATION DU SYSTEME EULALIE.

6.11 - PRESENTATION ET MODE DE TRAVAIL.

Le système Eulalie s'appelle comme un programme normal, aussi bien avec le système OS (carte de commande "// EXEC PGM=EULALIE") qu'avec le système CMS (commande "EULALIE"); une fois qu'il a le contrôle, on n'a plus à dialoguer qu'avec lui jusqu'au moment où il se termine. L'utilisateur du système Eulalie n'a donc aucun rapport avec le système hôte, et n'a aucun besoin de le connaître: il envoie des commandes à l'aide du clavier de fonctions, donne des réponses au système et compose son programme au moyen du clavier alphanumérique; sur l'écran apparaissent les messages du système et ce que frappe l'utilisateur; le pointeur optique ne sert pas pendant le dialogue avec le système, mais il peut être utilisé dans les programmes.

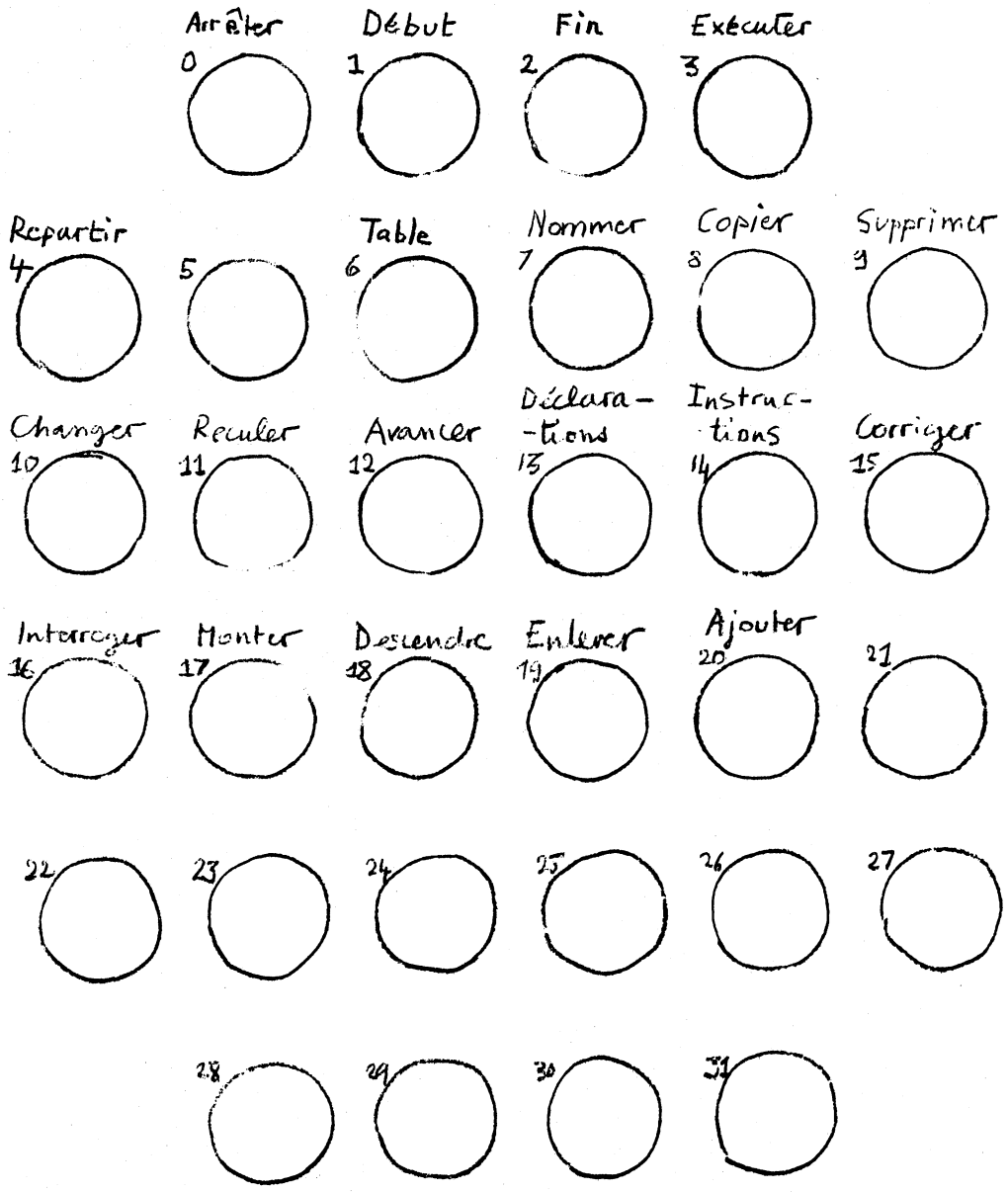


Figure 6.1 - Emplacement des fonctions sur le clavier.

Le système Eulalie a pour rôle de permettre la composition et l'exécution de programmes graphiques écrits dans le langage Euphémie. Pour lui faire faire ce que l'on désire, on communique avec lui au moyen de commandes, qu'on lui envoie dans tous les cas au moyen des touches du clavier de fonctions. La figure 6.1 montre l'emplacement des commandes sur le clavier, sous la forme du cache que l'on place sur ses touches. A chaque moment du dialogue ne sont éclairées que les touches qui correspondent à une commande autorisée: si l'on appuie sur une touche éteinte, il ne se passe rien.

Quand le nom d'une commande n'est pas suffisant pour déterminer complètement ce que l'on désire, le système demande la frappe d'informations au moyen du clavier alphanumérique. Il le fait toujours de façon explicite, c'est-à-dire qu'il affiche sur l'écran un message qui indique ce que l'on doit frapper, par exemple "DONNEZ VOTRE MOT DE PASSE" ou "NOMMEZ VOTRE PROGRAMME". A l'emplacement où apparaîtra le texte frappé vient se placer le curseur, petit trait placé en-dessous de la ligne, qui avance chaque fois que l'on tape un caractère. Si l'on s'est trompé en composant le texte, on peut déplacer le curseur sans modifier les caractères corrects, au moyen de trois touches: la touche "BACKSPACE" déplace le curseur d'une position vers la gauche, la touche "ADVANCE" le déplace d'une position vers la droite, la touche "JUMP" le ramène au début de la zône de composition du message; cette dernière zône est de longueur limitée, et le curseur ne peut pas en sortir: si l'on continue de frapper des caractères alors que le curseur est arrivé en fin de zône, ils modifient le dernier caractère et le curseur ne bouge pas. Si l'on maintient appuyée la touche "CONTINUE" en même temps qu'une autre touche, le caractère correspondant est indroduit dans plusieurs positions consécutives sans que l'on ait à le frapper plusieurs fois; l'introduction s'arrête quand on relâche la touche "CONTINUE". Une fois que l'on a terminé la composition d'un message (réponse ou ligne de programme), il faut le signaler au système: pour cela, on appuie à la fois sur la touche "ALTN CODING" et sur la touche "5" (le mot "END" est inscrit au-dessus de la touche, sur le bâti du clavier). Une dernière remarque importante est qu'il est inutile de frapper des caractères si le curseur n'apparaît pas sur l'écran; si on le fait par erreur, il faut appuyer à la fois sur les touches "ALTN CODING" et "SHIFT" pour déverrouiller le clavier (si on ne le fait pas, même les appuis sur des touches de fonctions ne sont pas pris en compte). Si l'on commet une erreur en frappant un texte (mot de passe incorrect, erreur dans une instruction), le système la signale par un message et actionne le signal auditif (petit coup de sifflet) pour attirer l'attention de l'utilisateur.

6.12 - LE LANGAGE DE COMMANDE.

6.121 - Principes d'organisation.

Le langage de commande s'exprime donc au moyen du clavier de fonctions; il est forme de vingt commandes différentes, réparties dans des contextes hiérarchisés, représentés sur la figure 6.2. Certaines commandes font descendre dans cette hiérarchie, c'est-à-dire qu'elles autorisent en même temps un groupe de commandes compatibles; d'autres n'ont aucun effet sur la hiérarchie, elles sont stationnaires; d'autres enfin font remonter dans la hiérarchie, c'est-à-dire qu'elles ramènent à l'état où l'on était avant la commande qui a permis de descendre. Les commandes descendantes sont celles qui mènent à une série d'actions en nombre indéfini, comme l'envoi de nouvelles commandes, la correction de textes, l'interrogation du système. Les commandes stationnaires n'ont qu'un effet unique et limité, comme le passage à la page suivante d'un texte. Les commandes ascendantes indiquent la fin du traitement commencé par une commande ascendante.

Pour l'utilisateur, l'emploi de cette hiérarchie est très simple, et l'allumage des seules touches correspondant à des commandes autorisées rend la situation encore plus claire. Pour passer d'un traitement a un autre, il faut envoyer un certain nombre de commandes ascendantes qui font gravir la hiérarchie jusqu'au point d'où l'on peut redescendre vers le nouveau traitement. La commande ascendante la plus utilisée est la commande fin, qui est valable dans tous les cas; un utilisateur en train de corriger un programme et qui veut en exécuter un autre doit envoyer cinq commandes successives, dont trois fois la commande fin; s'il veut laisser la place à une autre personne, il doit envoyer cette commande quatre fois de suite.

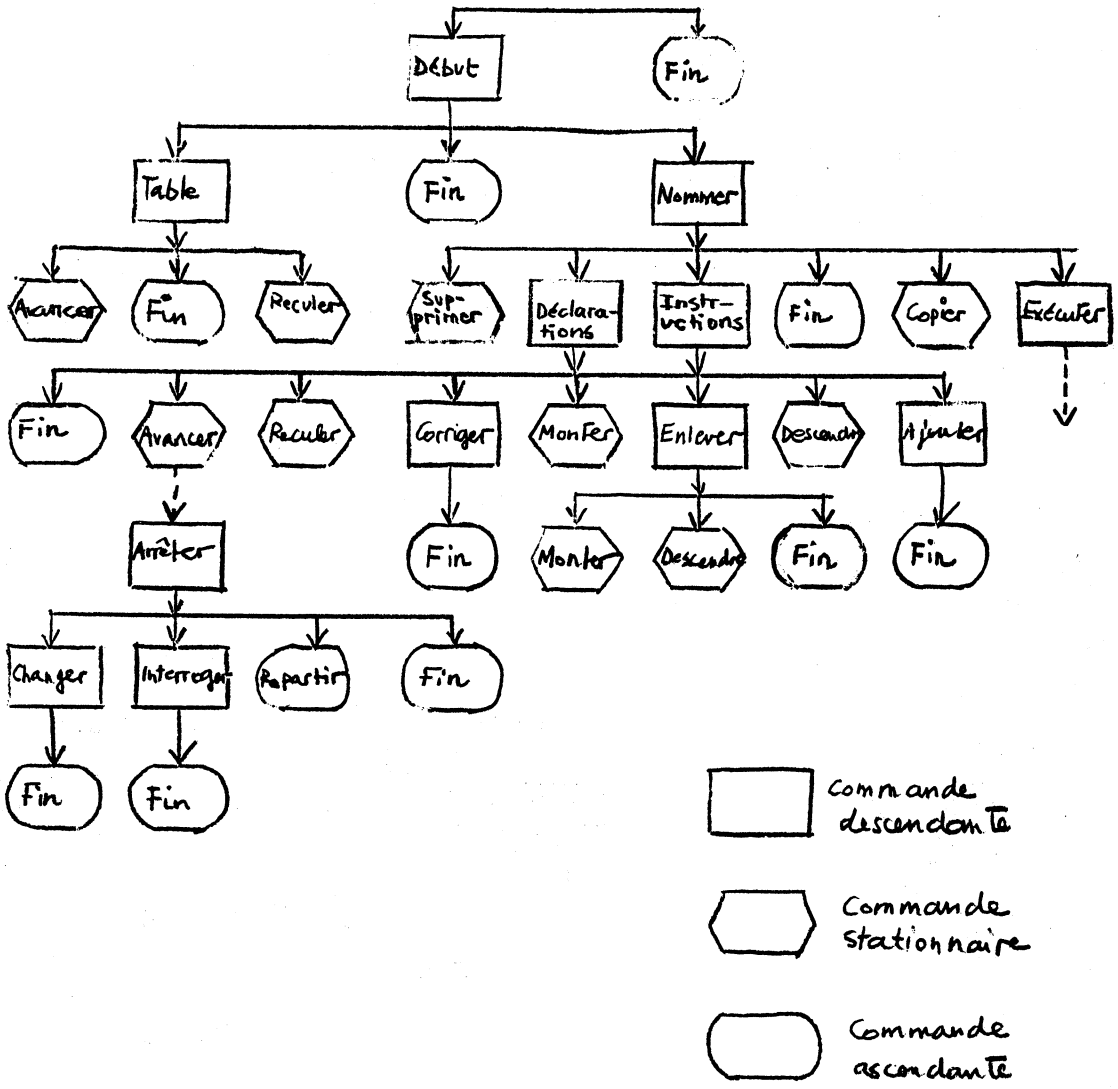


Figure 6.2 - Hiérarchie des commandes.

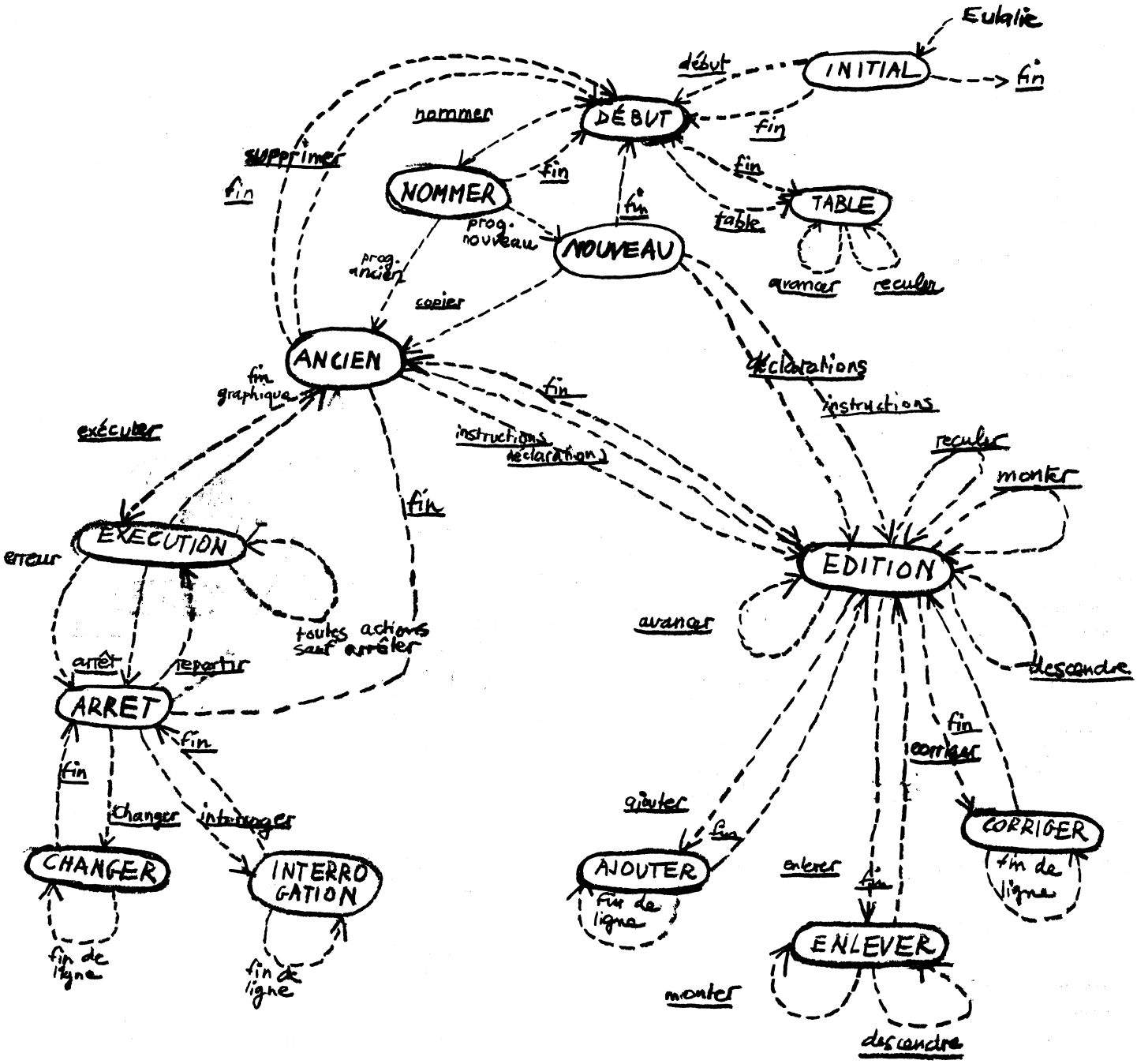


Figure 6.3 - Diagramme d'états du système Eulalie.

Le schéma hiérarchique de la figure 6.2 simplifie quelque peu la structure du langage de commande, car il ne fait pas intervenir les demandes d'informations du système ni la commande représentée par l'appui sur la touche "fin de ligne" (touches "ALTN CODING" et "END"). La figure 6.3 représente le fonctionnement des commandes d'une manière plus complète, sous la forme d'un diagramme d'états. Chaque cercle représente un etat, les fleches montrent les possibilites de passage d'un état a l'autre. Les noms des commandes sont soulignés, et l'om constate que l'on peut parfois passer d'un état à un autre sans l'envoi explicite d'une commande; en particulier, l'exécution de l'instruction fingraphique ramène de l'état EXECUTION à l'état ANCIEN comme si l'on avait envoyé la commande arrêter. On peut remarquer également que le principe hiérarchique est violé par la commande fin dans l'état ARRET, puisqu'elle saute un état en remontant; c'est la seule entorse faite aux règles exposées précédemment, car les états NOUVEAU et NOMMER n'ont qu'une existence temporaire, et l'on peut en fait considérer que les trois états NOMMER, ANCIEN et NOUVEAU n'en constituent qu'un, auquel on arrive par la commande nommer.

Dans l'exposé des commandes qui va suivre, on trouvera le dialogue entre l'utilisateur et le système exposé de la façon suivante: la lettre U représente une action de l'utilisateur, et S une action du système; un mot souligné représente l'envoi d'une commande; les messages apparaissent en lettres capitales, et les commentaires explicatifs en minuscules; certaines actions sont numérotées pour que l'on puisse s'y référer ensuite.

6.122 - Début du travail d'un utilisateur.

Quand le système Eulalie vient d'être appelé, ou qu'un utilisateur vient de s'en aller, seules sont autorisées les commandes début et fin. La commande fin termine l'utilisation du système Eulalie et rend le controle au systeme hôte. La commande début permet à un utilisateur de se présenter, le dialogue étant le suivant:

U début
1 - S DONNEZ VOTRE NOM :
U Le nom peut avoir jusqu'a 20 caractères, absolument quelconques.
fin de ligne
Si le nom donné n'apparaît pas dans la table des noms d'utilisateurs:
S JE NE VOUS CONNAIS PAS
2 - S COMMANDE DEBUT POUR RECOMMENCER, FIN POUR TERMINER
signal auditif
La commande début ramene en 1, la commande fin termine

l'utilisation du système.
Sinon:
S DONNEZ VOTRE MOT DE PASSE :
U L'emplacement prévu pour le mot de passe est recouvert de caractères enchevêtrés, pour qu'une personne étrangere ne puisse rien lire; le mot de passe peut avoir jusqu'a 16 caractères, également quelconques.
fin de ligne
Si le mot de passe ne correspond pas a ce qui apparaît dans la table des noms d'utilisateurs:
S MOT DE PASSE INCORRECT
retour en 2.
Sinon:
S Les messages disparaissent de l'écran, et l'on passe à l'état DEBUT, c'est-à-dire que sont allumées sur le clavier de fonctions les touches nommer, table et fin. Si l'on envoie à ce moment la commande fin, on est ramené à l'état initial, qui permet à un nouvel utilisateur de se presenter.

6.123 - Consultation de la table des programmes.

L'envoi de la commande table, qui n'est autorisée que dans l'état DEBUT, fait passer à l'état TABLE, où sont autorisées les commandes avancer, reculer et fin. Cet état a pour seul but de permettre de consulter la table qui donne pour chaque programme de l'utilisateur son nom, la date de sa création, la date de sa dernière utilisation et son encombrement sur disques, calculé en nombre d'éléments (voir la deuxième partie de ce chapitre pour la notion d'éléments sur disques).

Une ligne de la table apparaît sous la forme:
ESSAITROIS 01/04/70 17/04/70 00000257
Les programmes sont rangés dans la table dans l'ordre inverse de leur création, c'est-à-dire que le premier programme créé apparaît en dernier dans la table. Quarante-huit lignes de la table peuvent tenir en même temps sur l'écran; on appelle page un groupe de vingt-quatre lignes, et les commandes avancer et reculer permettent de "feuilleter" ces pages: s'il apparaît sur l'écran les pages de numéros n et n+1, la commande avancer permet de montrer les pages n+1 et n+2, et la commande reculer permet de montrer les pages n-1 et n; si les pages que l'on veut faire apparaître n'existent pas, les commandes n'ont aucun effet; le texte "*** FIN DE LA TABLE ***" apparaît sur l'écran si la dernière page affichée est incomplète. Si l'utilisateur n'a encore jamais travaillé sur le système, ou qu'il a détruit tous ses programmes, le message "*** TABLE VIDE ***" apparaît au milieu de l'écran.

La commande fin ramène à l'état DEBUT.

6.124 - Début du travail sur un programme.

La commande nommer, qui n'est autorisée que dans l'état DEBUT, permet de commencer le travail sur un programme précis, le dialogue étant le suivant:

U nommer
S NOMMEZ VOTRE PROGRAMME :
U Le nom peut avoir jusqu'à 10 caractères, absolument quelconques.
fin de ligne

Si le nom donné n'apparaît pas dans la table des programmes de l'utilisateur:

S CE PROGRAMME N'EXISTE PAS ENCORE
On passe à l'état NOUVEAU, ou sont autorisées les commandes déclarations, instructions, copier et fin; l'envoi de la commande fin ramène à l'état DEBUT, les trois autres transforment le programme nouveau en programme existant, c'est-à-dire que l'on reviendra ensuite à l'état ANCIEN.

Si le nom apparaît dans la table:

S CE PROGRAMME A ETE CREE LE JJ/MM/AA, UTILISE LE JJ/MM/AA.
IL OCCUPE XXXXXXXX ELEMENTS.
On passe à l'état ANCIEN, où sont autorisées les commandes déclarations, instructions, exécuter, supprimer et fin.

Si un programme nommé n'existe pas encore, on peut donc:
- considérer que c'est une erreur (on a frappé un mauvais nom) et envoyer la commande fin qui ramène à l'état DEBUT;
- commencer la création du programme en envoyant la commande déclarations (voir le paragraphe 6.13);
- constituer le programme en en recopiant un autre, le dialogue étant le suivant:

U copier
S NOM DU PROGRAMME A COPIER :
U Le nom peut avoir jusqu'à 10 caractères.
fin de ligne

Si le nom donné n'apparaît pas dans la table des programmes de l'utilisateur:

S PROGRAMME INCONNU
Signal auditif et retour à l'état NOUVEAU.

Si le nom apparaît dans la table, il n'y a pas de réponse explicite du système, mais on effectue la copie demandée et on passe à l'état ANCIEN.

Si un programme donné existe déjà, on peut:
- considérer que c'est une erreur et revenir à l'état DEBUT par la commande fin;
- supprimer complètement le programme en envoyant la commande supprimer, qui ramène elle aussi à l'état DEBUT avec le commentaire:
LE PROGRAMME XXXXXXXXXX EST SUPPRIME.
- modifier le programme en envoyant les commandes déclarations ou instructions (voir le paragraphe 6.13);
- exécuter le programme en envoyant la commande exécuter (voir le paragraphe 6.14).

Pendant tout le cours du travail sur un programme précis (sauf son exécution), il apparaît en bas de l'écran le texte "- PROGRAMME NNNNNNNNNN", ou les N sont remplacés par le nom du programme. Pendant la composition ou la modification, ce texte est précédé d'une des indications suivantes:

ON AJOUTE DES DECLARATIONS
ON ENLEVE DES DECLARATIONS
ON CORRIGE DES DECLARATIONS

ou les mêmes textes avec INSTRUCTIONS au lieu de DECLARATIONS.

6.13 - COMPOSITION ET MODIFICATION DES PROGRAMMES.

6.131 - Principes généraux.

Les déclarations et les instructions des programmes sont traitées de façon complètement séparée, c'est-à-dire que si l'on veut passer de la création ou de la modification des déclarations à celle des instructions, ou vice-versa, il faut remonter jusqu'à l'état ANCIEN.

Toute modification aux déclarations peut se répercuter de façon importante sur l'ensemble des instructions, qu'il s'agisse d'une adjonction, d'une correction ou d'une suppression. Dans son état actuel et de facon extrêmement provisoire, le système Eulalie interdit toute modification autre qu'une adjonction aux déclarations, c'est-à-dire que les commandes enlever et corriger ne sont pas utilisables. Dans une étape ultérieure, et nous l'espérons assez proche, on procèdera de la manière suivante quand l'utilisateur enverra la commande fin après avoir modifié des déclarations: on recompilera les instructions du programme une à une en les faisant défiler sur l'écran; en cas d'erreur (par exemple on aura changé la signification d'une variable, ou bien on en aura supprimé une autre), on demandera la correction immédiate de l'instruction fautive; on continuera ainsi jusqu'à la fin des instructions, avant de retourner à l'état ANCIEN; au cas où l'utilisateur demanderait

terminer le travail sur ce programme avant d'avoir corrigé toutes les erreurs, on le forcerait à les corriger à la prochaine utilisation de ce programme, et en tout cas avant toute tentative d'exécution.

Le travail sur le programme est fait non pas par ligne mais par déclaration ou par instruction; le compilateur n'entreprend l'analyse du texte frappé ou corrigé que s'il y trouve un point-virgule; dans le cas contraire, et quel que soit le mode de travail, il ajoute une ligne blanche sous la ligne en cours pour que l'on puisse compléter le texte introduit. Si le texte renferme un point-virgule, le compilateur l'analyse et le traduit immédiatement; deux cas peuvent alors se produire: si le texte renferme une erreur, le système affiche un message explicatif en bas de l'écran, actionne le signal auditif et attend que l'on corrige l'erreur; si le texte est correct, le système le remplace par sa retraduction à partir de la forme intermédiaire qu'il vient de conserver sur disque (voir la deuxième partie de ce chapitre).

La retraduction d'une instruction ou d'une déclaration apparaît suivant un modèle normal qui peut différer notablement de celui dont s'est servi l'utilisateur; les règles principales sont les suivantes:
- les lignes qui suivent la première ligne d'une instruction ou d'une déclaration sont décalées à droite de quatre caractèrees;
- les espaces entre symboles sont sans signification; en conséquence, on n'en place qu'un, sauf dans les cas suivants ou l'on n'en met pas: avant la parenthèse droite, la virgule et le point-virgule; après la parenthèse gauche;
- les symboles de base composés sont notés dans leur forme française la plus longue, mais au masculin singulier;
- les symboles, identificateurs et constantes ne sont jamais coupés entre une ligne et la suivante; les chaînes peuvent être coupées si elles dépassent dix caractères; dans ce dernier cas, on le signale en plaçant en troisième caractère de la ligne où apparaît le second fragment de la chaîne le signe -;
- les différentes options des déclarations ou instructions complexes sont placées toujours dans le même ordre, qui est celui qui a été donné dans la description du langage au chapitre 4;
- sauf dans le cas de la déclaration de figure, les options sont toujours indiquées explicitement, même si elles ont été prises par défaut;
- à l'intérieur d'une chaîne, le caractère '"' n'apparaît pas double, comme a dû le taper le programmeur, mais simple.

Ainsi, l'instruction suivante:

```
'SPEC' N 'TEXT'ILVOIT<== TEXTE 'IN' X , Y 'HEIG' HAUT
'LARGE' LARGE 'ESPACE' ESPACE 'ORIENT'OR ;
```

sera réaffichée de la façon suivante:

```
'SPECIAL' N 'TEXTE' ILVOIT <== TEXTE 'EN' X, Y 'HAUTEUR' HAUT 'LARGEUR'
   LARGE 'ESPACEMENT' ESPACE 'ORIENTATION' OR;
```

L'instruction:

```
'ETAT NIVEAU ( 1, -20, 31) 'RESULT'TABLE 'REPON'OUESTCE;
```

devient:

```
'ETAT' NIVEAU (1, -20, 31) 'REPONSE' OUESTCE 'RESULTAT' TABLE 'ATTENTE'
    ;
```

L'instruction:

```
'SELECTION'BARATIN'DEPUIS'DEBUT+DEPLACEMENT 'LONGUEUR'EXTREMITE+20 :=
ANALYSE || ",:""'?" || Q || "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
|| 'SELECTION' CHAINE 'LONGUEUR' ABS (N - M);
```

devient:

```
'SELECTION' BARATIN 'DEPUIS' DEBUT + DEPLACEMENT 'LONGUEUR' EXTREMITE
   + 20 := ANALYSE || ",:"'?" || Q || "ABCDEFGHIJKLMNOPQRSTUVWXYZ012
  -3456789" || 'SELECTION' CHAINE 'LONGUEUR' ABS (N - M);
```

6.132 - Déplacements dans le texte.

Le texte affiché, declarations ou instructions, est réparti en "pages" de vingt-quatre lignes comme dans le cas de la table des noms de programmes (voir le paragraphe 6.123). Les commandes avancer et reculer ont le même effet que précédemment; si les pages n et n+1 sont affichées sur l'écran, la commande avancer fait apparaître les pages n+1 et n+2, et la commande reculer fait apparaître les pages n-1 et n. Le texte affiché peut ne pas occuper exactement quarante-huit lignes, car on ne montre pas d'instructions incomplètes (je parlerai par la suite simplement d'instructions, pour ne pas avoir à répéter continuellement "déclarations ou instructions"): si la dernière instruction de la deuxième page affichée ne peut pas tenir en entier sur l'écran, on ne la montre pas du tout, si bien qu'il peut rester quelques lignes inoccupées en bas d'écran. De même, un changement de page peut produire un déplacement de moins de vingt-quatre lignes, car on commence toujours en haut d'écran sur la première ligne d'une instruction.

Les opérations de modification au texte se font toujours relativement à une instruction précise, qui est l'instruction courante. Elle est repérée par deux marqueurs dans les marges de gauche et de droite: "=>" a gauche et "<=" a droite; il reste donc 70 caractères pour le texte. Quand on affiche du texte pour la première fois (juste après les commandes déclarations ou instructions) ou quand on change de page (par les commandes avancer ou reculer), l'instruction

courante est celle qui apparaît tout en haut de l'écran. On dispose pour déplacer les marqueurs des commandes monter et descendre: la commande monter place les marqueurs une instruction au-dessus de l'instruction courante, la commande descendre fait la même chose une instruction au-dessous; la commande monter n'a aucun effet si l'on est en haut de l'écran, de même que la commande descendre si l'on est en bas d'écran.

le travail de l'utilisateur se présente donc de la façon suivante: avec la commande déclarations ou la commande instructions, il détermine sur quelle partie de son programme il va travailler; les deux premières pages du texte apparaissent sur l'écran. Avec les commandes monter, avancer ou reculer, il amène sur l'écran la partie du texte qui l'intéresse. Avec les commandes monter ou descendre, il amène les marqueurs devant l'instruction précise sur laquelle il veut travailler. Il peut alors effectuer les modifications voulues à l'aide des commandes ajouter, corriger et enlever.

6.133 - Adjonctions.

La commande ajouter permet d'insérer des instructions après l'instruction courante, pointée par les marqueurs; si l'instruction courante est la dernière du programme, on peut par ce moyen compléter ou étendre celui-ci. Quand on envoie la commande ajouter, le système insère une ligne blanche juste en-dessous de l'instruction courante; les marqueurs descendent se placer devant cette ligne blanche, indiquant ainsi qu'elle va constituer l'instruction courante. Le curseur apparaît en tête de ligne, et on peut frapper le texte voulu; quand on envoie la commande fin de ligne, on peut se trouver dans l'une des trois situations suivantes:
- Si la ligne correspond à une instruction complète (qui renferme un point-virgule) et correcte, ladite instruction est traduite, retraduite et réaffichee; tout se passe ensuite comme si l'on venait d'envoyer la commande ajouter: insertion d'une ligne blanche, descente des marqueurs devant, mise en place du curseur; l'utilisateur peut donc continuer ses adjonctions.
- Si la ligne correspond à une instruction complète (qui renferme un point-virgule) mais incorrecte, le système affiche un message expliquant l'erreur en bas d'écran, actionne le signal auditif, et attend que l'utilisateur corrige cette erreur; on recommence alors comme avant.
- Si la ligne correspond à une instruction incomplète (qui ne renferme pas de point-virgule), le système insère simplement une nouvelle ligne blanche; les marqueurs ne bougent pas, puisqu'ils pointent la première ligne d'une instruction.

Au cours des adjonctions de lignes blanches, il peut se trouver que l'on arrive en bas d'écran; dans ce cas, le système décale tout le texte d'une page, c'est-à-dire que tout se passe comme si l'on avait envoyé la commande avancer, si ce n'est que les marqueurs restent sur l'instruction en cours d'établissement.

Si l'on oublie le point-virgule final d'une instruction, le système présume que celle-ci ne tient pas sur une ligne, et en ajoute une autre; on se trouve alors devant une instruction sur deux lignes (ou plus), dont la dernière ligne est entièrement blanche; ceci ne présente aucun inconvénient, puisque le système replace les lignes quand il réaffiche le texte.

Quand on travaille sur une instruction qui occupe plusieurs lignes, le curseur ne peut se déplacer que sur une ligne à la fois; pour le faire passer au début de la ligne suivante, il faut utiliser la touche "JUMP" du clavier alphanumérique; si l'on est dans la dernière ligne d'une instruction, la touche "JUMP" ramène le curseur au début de la première ligne de la même instruction. On ne peut en aucun cas faire sortir le curseur de l'instruction courante.

Avant tout travail sur un programme, l'écran est entièrement vide après que l'on ait envoyé la commande déclarations ou la commande instructions; en particulier, les marqueurs sont invisibles. On les voit apparaître tout en haut de l'écran dès que l'on envoie la commande ajouter.

Pour sortir de l'état AJOUTER, il faut envoyer la commande fin; deux cas peuvent alors se produire:
- Si l'instruction courante est complète et correcte, le système la traduit, la retraduit et la réaffiche, puis l'on sort de l'état AJOUter sans rien faire de plus; les marqueurs restent donc en place, et si l'on envoie à nouveau immédiatement la commande ajouter, tout se passe comme si l'on avait simplement envoyé la commande fin de ligne au lieu de la commande fin.
- Si l'instruction courante est incomplète ou incorrecte, on la supprime complètement et on replace les marqueurs sur l'instruction qui la précédait, puis l'on sort de l'état AJOUTER; ceci évite d'obliger l'utilisateur à composer une instruction correcte pour la supprimer ensuite, quand il a commence d'écrire une instruction par erreur. D'autre part, un cas particulier d'instruction incomplète est constitué par une ligne vide; on peut donc envoyer la commande fin immédiatement après le fin de ligne d'une instruction correcte: le système affiche une ligne blanche puis l'enlève aussitôt.

6.134 - Corrections.

La commande corriger permet de modifier des instructions sans en ajouter ni en supprimer; on commence à la ligne courante, et on continue jusqu'à ce que l'utilisateur envoie la commande fin ou jusqu'à ce que l'on arrive au bout du programme; dans ce dernier cas, la sortie du mode CORRIGER est automatique. Quand on envoie la commande corriger , le système place donc le curseur en tête de l'instruction courante; à l'aide des touches "ADVANCE", "JUMP" et "BACKSPACE", on peut alors amener le curseur sous les parties de l'instruction que l'on veut corriger, puis frapper les caractères voulus. De même qu'avec la commande ajouter, trois cas peuvent se produire quand on envoie la commande fin de ligne; dans le cas d'une instruction incomplète ou incorrecte, les choses se passent exactement comme au paragraphe précédent; dans le cas d'une instruction correcte, les marqueurs viennent devant l'instruction suivante, en tête de laquelle se place le curseur, et l'on peut poursuivre les corrections.

Si l'on doit corriger deux instructions séparées par exemple par deux instructions correctes, il est tout aussi simple de faire comme si l'on corrigeait tout de même ces deux instructions que de sortir de l'état CORRIGER pour y rentrer presque immédiatement: dans un cas, on envoie deux fin de ligne qui font simplement avancer les marqueurs; dans l'autre cas, il faut envoyer successivement les commandes fin, descendre, descendre et corriger.

Si l'on veut réduire la longueur d'une instruction, il est inutile d'effacer toute la partie qui ne sert a rien: le système ignore tout ce qui suit le point-virgule final d'une instruction correcte, et le fait disparaître au moment du réaffichage.

On sort de l'état CORRIGER par la commande fin; le système réagit de la même façon qu'avec l'état AJOUTER, c'est-à-dire qu'une instruction incorrecte ou incomplete est supprimée, et que dans ce cas les marqueurs remontent d'une instruction.

6.135 - Suppression.

La commande enlever permet de supprimer une ou plusieurs instructions; étant donné qu'il s'agit d'une manipulation parfois dangereuse, on procède d'une facon qui permet d'éviter les conséquences fâcheuses de certaines fausses manoeuvres. Les seules commandes autorisées dans l'état ENLEVER sont monter, descendre et fin. Au moment ou l'on envoie la commande enlever, le système repère la position de l'instruction courante; les commandes monter et

descendre permettent alors de déplacer les marqueurs jusque devant la première instruction à ne pas supprimer; la suppression n'a effectivement lieu que quand on envoie la commande fin; on sort alors de l'état ENLEVER.

Si donc on a commis une fausse manoeuvre en envoyant la commande enlever, il suffit d'envoyer immediatement la commande fin: les marqueurs étant restés au même endroit, le système constate qu'il n'a rien à supprimer, et ne fait donc rien du tout.

Pour faire une suppression qui s'étend sur plusieurs pages, on doit procéder par étapes, puisque les commandes avancer et reculer ne sont pas autorisées dans l'état ENLEVER. Comme d'ailleurs le système tasse les textes après les suppressions, on n'a pas beaucoup de travail à faire: supposons que les marqueurs soient devant l'instruction numéro n du programme; si l'on envoie les commandes enlever, vingt fois descendre et fin, le tout quatre fois, on aura au total supprimé quatre-vingts instructions, qui évidemment ne tenaient pas sur l'écran, et sans jamais avoir demandé explicitement de changement de page. En effet, juste avant la première commande fin, les marqueurs sont devant l'instruction n+20; on efface les instructions n a n+19, c'est-à-dire que l'instruction n+20 vient à la place de l'instruction n, tout en conservant les marqueurs: on voit donc sur l'écran les instructions n+20 a n+40, dont une partie au moins pouvait être invisible avant la suppression.

6.14 - EXECUTION ET MISE AU POINT DES PROGRAMMES.

6.141 - Exécution.

La commande exécuter n'est autorisée que dans l'état ANCIEN, mais dans ce cas elle l'est toujours: tout programme ancien est donc considéré comme exécutable, même s'il ne renferme que des déclarations. Apres l'envoi de la commande exécuter, le système Eulalie semble disparaître pour laisser la place au programme; en particulier, toutes les touches de fonctions s'éteignent, y compris la touche 0, qui est pourtant le seul dispositif qui reste sous le contrôle du système. L'utilisateur a d'ailleurs intérêt à remplacer le cache du clavier de fonctions par un autre qui indique le langage de communication accepté par son programme.

Il existe trois facons de sortir de l'état EXECUTION, c'est-à-dire du déroulement du programme. La facon normale est l'exécution par le programme de l'instruction fingraphique (cette instruction peut être implicite si l'on tombe sur la fin physique des instruc-

tions, en particulier dans le cas d'un programme incomplet); on est ramené alors à l'état ANCIEN, où l'on peut ré-exécuter le programme (exécuter), le modifier (déclarations ou instructions) ou passer à autre chose (fin).

Les deux autres façons de sortir de l'état EXECUTION correspondent à une situation anormale. Dans le premier cas, cette situation est détectée par le système Eulalie, qui trouve une erreur dans le programme: étiquette indéfinie, mauvaise terminaison d'instructions pour ou si, ou erreurs de calcul (division par zéro, dépassement de capacité de nombre réel, indice de tableau trop petit ou trop grand, etc.); l'image qu'affichait éventuellement le programme disparaît de l'écran, le système affiche un message explicatif, actionne le signal auditif et passe dans l'état ARRET. Dans le deuxième cas, la situation anormale est détectée par l'utilisateur lui-même, le programme n'effectuant pas ce que l'on pourrait attendre; dans ce cas, l'utilisateur envoie la commande arrêter (touche zéro), et l'on passe là aussi dans l'état ARRET, qui va être étudié à présent.

6.142 - Mise au point.

On arrive donc dans l'état ARRET, soit quand on envoie la commande arrêter, soit quand le système détecte une erreur au cours de l'exécution du programme. Le système signale sur l'écran l'instruction exacte où s'est produit l'arrêt en indiquant le nom de l'étiquette du programme la plus proche avant cette instruction, et le nombre d'instructions qui les séparent. Si au cours de l'exécution du deuxième exemple du paragraphe 4.175 le système signale:

```
INDICE DE TABLEAU TROP GRAND
INSTRUCTION HEP + 8
```

on saura que l'erreur s'est produite dans l'instruction qui commence par position ROND <== ... ", sur l'une des utilisations du tableau TABLE.

S'il n'y a pas d'instruction étiquetée avant l'instruction où l'on s'arrête, le système donne un numéro d'ordre depuis le début du programme; avec le même exemple que précédemment, le message:

```
DEPASSEMENT DE CAPACITE DE NOMBRE REEL
INSTRUCTION 11
```

signale l'instruction qui commence par "TABLE (2 * I) := 2047 ..." (on remarquera que les déclarations n'interviennent pas dans la numérotation).

Une fois que l'on est dans l'état ARRET, on a le choix entre quatre actions, qui correspondent à quatre commandes. La commande fin est la plus simple: elle ramène directement à l'état ANCIEN,

comme si l'on avait exécuté dans le programme l'instruction fingraphique. C'est donc un moyen simple de terminer l'exécution d'un programme dans lequel on n'a pas prévu de moyen extérieur d'arrêt. Les autres commandes vont être maintenant étudiées en détail.

6.1421 - INTERROGATION DU SYSTEME. Une première action intéressante est d'essayer de déterminer la cause de l'anomalie qui a conduit à l'arrêt du programme. Pour cela, on peut demander au système la valeur de n'importe quelle variable du programme, ainsi que celles que l'on obtient au moyen des fonctions type, abscisse, ordonnée et erreur. Le dialogue avec le système est le suivant:

```
      U Interroger
1 -   S VALEUR DE :
      U  Il donne le nom d'une variable du programme (variable
         graphique, variable numérique ou chaîne), ou écrit l'appel
         d'une des quatre fonctions indiquées ci-dessus.
         fin de ligne
      Si le nom donné n'existe pas dans le programme, ou si l'appel
         de fonction est mal écrit:
      S  Message explicatif;
         Signal auditif;
         Retour en 1.
      Sinon:
      S  Affichage de la valeur demandée, dans le type correspondant à
         celui de la variable; pour une variable graphique, la valeur
         est donnée en hexadécimal, et généralement inutilisable si
         l'on n'a pas une bonne connaissance de G.S.P.
         Retour en 1.
```

On sort de l'état INTERROGATION en envoyant la commande fin au lieu de fin de ligne.

6.1422 - MODIFICATIONS. La deuxième action intéressante est d'essayer de corriger la cause de l'anomalie. Le système Eulalie ne le permet que d'une facon limitée au moment de l'arrêt du programme, en ce sens que l'on n'effectue pas de corrections au programme lui-même: on ne peut que donner quelques instructions, qui seront executées en différé juste avant que l'on ne relance le programme (voir le paragraphe suivant). Ces instructions peuvent servir à changer la valeur d'une variable numérique (par une instruction d'affectation), à modifier l'état d'une variable graphique (par une instruction telle que fin figure, afficher, omettre, ou une instruction de génération ou de mise a jour), ou même à corriger une erreur syntaxique concernant les isntructions pour et si (en ajoutant une instruction finpour ou finsi manquante, par exemple).

L'envoi de la commande changer permet de commencer la frappe d'instructions exactement comme si l'on était sous contrôle de la commande ajouter. Il y a cependant une différence majeure: les instructions frappées ne sont pas insérées dans le programme. A part cela, on peut très bien finalement écrire un véritable petit programme, avec instructions de branchement si c'est nécessaire, mais ce n'est pas très utile: si l'on a réellement de nombreuses corrections a faire, il vaut beaucoup mieux les effectuer sur le programme lui-même.

La commande fin termine l'état CHANGER dans les mêmes conditions que si l'on était dans l'état AJOUTER; si l'om revient à l'état CHANGER au cours du même arrêt, on peut ajouter des instructions à celles qui étaient déjà frappées; toutes ces instructions sont perdues pour un arrêt ultérieur.

6.1423 - REPRISE DE L'EXECUTION. Apres avoir reconnu et éventuellement corrigé la cause de l'anomalie qui a conduit à l'arrêt du programme, on peut tenter de reprendre son exécution (si la reprise est impossible, la commande fin permettra de revenir à l'état ANCIEN). On envoie donc la commande repartir, à laquelle le système répond en demandant où l'on doit repartir, d'après le dialogue suivant:

U repartir
1 - S OU ?
U L'utilisateur donne le nom d'une étiquette de son programme, suivi éventuellement d'un déplacement, sous la forme: NNNNNNNN + XXX. Dans le premier cas indiqué au paragraphe 6.142, la réponse "HEP + 7"" renverrait a l'instruction "pour I := 1 a 8;"; dans le deuxième cas, la réponse "8" ou "+ 8" renverrait a l'instruction qui commence par "texte CERCLE <== ...". Rien n'empêche de repartir sur l'instruction qui a provoque l'erreur, ou inmédiatement après.
fin de ligne
Si la réponse n'est pas correcte:
S Message explicatif;
signal auditif;
retour en 1.
Sinon:
S L'exécution reprend à l'endroit indique, de la façon suivante:
- on restaure l'état de l'écran et du clavier de fonctions tels qu'ils étaient au moment de l'arrêt;
- on exécute les instructions qui ont été frappées sous le contrôle de la commande changer;
- on relance le programme a l'endroit indiqué.

L'ordre dans lequel sont faites les deux premières opérations permet que les instructions introduites au moyen de la commande changer modifient précisement l'état de l'écran ou du clavier de fonctions.

6.15 - TRAVAIL DU PROGRAMMEUR-MAITRE.

On appelle programmeur-maître l'utilisateur unique qui a le privilège d'introduire auprès du système Eulalie de nouveaux utilisateurs, et d'en retirer d'autres. Le nom et le mot de passe de ce programmeur-maître sont déterminés au moment de l'initialisation de la zône sur disques réservée au système (voir le paragraphe 6.221), mais ils peuvent être modifiés par la suite. Le programmeur-maître ne dispose en fait d'aucun programme, et les commandes ordinaires ont pour lui une signification spéciale. Quand il se presente, tout se passe normalement jusqu'au moment où l'on est dans l'état DEBUT. Les commandes autorisées sont toujours table, nommer et fin, mais elles ont une signification différente, que je vais détailler à présent.

6.151 - Table des utilisateurs.

La commande table permet au programmeur-maître de consulter la table des utilisateurs. Le fonctionnement est exactement le même que pour une table de programmes, et les commandes fonctionnent comme il est indiqué au paragraphe 6.123. La table des utilisateurs comprend uniquement les noms des personnes autorisées à travailler avec le système Eulalie, dans l'ordre inverse de leur introduction. Les seules manipulations que peut faire le programmeur-maître sur cette table servent à la consulter.

6.152 - Introduction ou élimination d'un utilisateur.

La commande nommer sert à introduire ou retirer des utilisateurs; le dialogue est le suivant:

U nommer
S NOM de L'UTILISATEUR :
U Il tape ce nom, qui peut avoir 1 a 20 caractères quelconques.
fin de ligne
Si le nom donne n'apparaît pas dans la table:
S UTILISATEUR NOUVEAU
MOT DE PASSE :
U Le programmeur-maître peut taper le mot de passe (1 a 16 caractères quelconques) et envoyer la commande fin de ligne, auquel cas ce nouvel utilisateur est introduit dans la table,

ou bien il peut envoyer la commande fin pour signaler qu'il s'est trompé, auquel cas cas on ne fait rien; dans les deux cas, on revient à l'état DEBUT.

Si le nom donné apparaît déjà dans la table:

S UTILISATEUR EXISTANT

Les commandes fin et supprimer sont autorisées; fin envoie en 1, supprimer en 2.

1 - U fin

S NOUVEAU MOT DE PASSE :

U Il peut taper un nouveau mot de passe et envoyer la commande fin de ligne, auquel cas le mot de passe de cet utilisateur est modifié dans la table; ceci peut être utile au cas ou un mot de passe a été divulgué par erreur. Si le programmeur-maître envoie simplement la commande fin à nouveau, on ne modifie rien; ceci est utile si l'on s'est trompé de nom, ou si l'on veut simplement savoir si un utilisateur figure dans la table sans pour autant consulter celle-ci. On revient à l'état DEBUT.

2 - U supprimer

S Le nom de l'utilisateur est retiré de la table. S'il avait des programmes, on va en 3.

PAS DE PROGRAMMES

On revient à l'état DEBUT.

3 - S SUPPRESSION DE L'UTILISATEUR NNNNNNNN

Ce message reste affiché pendant toute la suite.

FIN POUR TRAITER EN BLOC, FIN DE LIGNE POUR LE DETAIL.

Le traitement en bloc consiste a effacer ou transferer à un autre utilisateur tous les programmes de l'utilisateur que l'on retire; le traitement en détail consiste à considérer séparement le cas de chaque programme.

U fin envoie en 4.

fin de ligne

S PROGRAMME MMMMMMMMM

Il s'agit du nom du premier programme de l'utilisateur que l'on retire.

5 - S NOM DE L'UTILISATEUR :

U Le programmeur-maître tape le nom de l'utilisateur à qui il veut transférer tous les programmes ou le programme dont le nom apparaît sur l'écran, et envoie la commande fin de ligne; s'il veut supprimer au lieu de transférer, il envoie simplement la commande fin.

S On effectue le travail demandé; si l'on travaille en bloc, on revient ensuite au niveau DEBUT; sinon, on fait apparaître le nom du prochain programme à traiter et on revient en 4. Dans ce dernier cas, si l'utilisateur à qui l'on veut transférer le programme est toujours le même, on n'a pas besoin de

frapper a nouveau son nom: il suffit d'envoyer la commande fin de ligne, le nom étant resté sur l'écran. Quand on travaille en détail, le traitement s'arrête après le dernier programme de l'utilisateur que l'on retire, et l'on revient alors à l'état DEBUT.

6.153 - Fin du travail du programmeur-maître.

Seul le programmeur-maître peut modifier les mots de passe des utilisateurs ou les retirer; cette restriction s'applique tout aussi bien à l'utilisateur particulier qu'il constitue lui-même, mais il faut de plus procéder d'une facon spéciale, car le nom du programmeur-maître n'apparaît pas dans la table des utilisateurs. Les modifications sont donc possibles au moment ou le programmeur-maître termine son travail, le dialogue étant le suivant:

U fin
S NOUVEAU NOM :
U On peut taper le nouveau nom à donner au programmeur-maître, et envoyer la commande fin de ligne; si l'on ne veut pas modifier ce nom, on envoie simplement la commande fin. Si l'on donne au programmeur-maître un nom qui figure déjà dans la table des utilisateurs, l'utilisateur en question ne pourra plus travailler, ce qu'il faut bien entendu éviter.
S NOUVEAU MOT DE PASSE :
U On peut taper le nouveau mot de passe du programmeur-maître, et envoyer la commande fin de ligne; si l'on ne veut pas modifier ce mot de passe, on envoie la commande fin. On retourne alors à l'état INITIAL.

La fin du travail du programmeur-maître est donc en général signalée par trois envois consécutifs de la commande fin.

6.2 - PRINCIPES DE CONCEPTION DU SYSTEME EULALIE.

Je n'ai aucun désir d'exposer dans ce paragraphe le détail du fonctionnement des programmes et modules qui composent le système Eulalie: cela ne ferait qu'alourdir un texte déjà volumineux, sans la moindre utilité pratique. Je me contenterai donc d'exposer successivement:
- les points d'appui du système, c'est-à-dire les options fondamentales qui ont été prises au sujet de sa réalisation;
- la structure générale du système et de ses composants, sans jamais descendre au niveau des représentations en mémoire, codifications et autres détails de programmation;
- enfin, les quelques idées d'extensions qui peuvent se présenter.

6.21 - LES POINTS D'APPUI DU SYSTEME.

Les deux points principaux sont les mêmes que ceux du langage Euphémie: il s'agit de l'ensemble de sous-programmes G.S.P. [Ru68, IBM2] et du "langage" Macro-Euphémie [C169]. Plus précisément, le système Eulalie est constitué d'un certain nombre de modules de programmes, écrits dans le langage d'assemblage de l'ordinateur 360; ces programmes doivent utiliser les possibilités du terminal graphique 2250, en tant que moyen de dialogue avec l'usager: tout ce qui concerne ce terminal est écrit au moyens de macros-instructions du langage Macro-Euphémie, qui conduisent finalement à l'appel de sous-programmes de G.S.P.

Il y a donc la un choix délibéré, et dont il ne faut pas se dissimuler les inconvénients:
- lourdeur d'écriture de certaines opérations;
- énormite du temps d'assemblage des programmes;
- encombrement de la mémoire par G.S.P.;
- performances parfois médiocres.

En fait, les avantages de ce choix, même s'ils sont moins évidents, l'emportent largement sur les inconvénients. Ramenons d'abord ces derniers à leur juste valeur: les programmes qui constituent Eulalie n'ont pas à être assemblés fréquemment, sauf pendant la phase de mise au point; l'encombrement de la mémoire par G.S.P. n'est pas gigantesque, et de toutes facons il n'est pas gênant puisque G.S.P. doit être là quand on exécute les programmes écrits en Euphémie; on n'a pas besoin de performances extraordinaires, car on sait que l'on ira toujours plus vite que l'utilisateur, comme c'est le cas dans tout travail conversationnel tant que le système n'a rien à faire de répétitif. Mais la meilleure raison d'utiliser Macro-Euphémie est que l'on retrouve par ce moyen les avantages de l'écriture des programmes en langage évolué, tout en ne perdant que très peu des avantages de l'écriture en langage d'assemblage. En gros, on peut dire que l'on peut grâce à Macro-Euphémie écrire rapidement et facilement des programmes clairs et assez généraux pour pouvoir par la suite être facilement transposés sur un autre matériel.

Les mêmes raisons se sont retrouvées à un autre moment, et elles ont conduit à un choix analogue: comme on le verra par la suite, l'écriture du traducteur et du retraducteur, celle de l'interprète, celle des différents sous-programmes, tout cela s'appuie sur la définition et l'utilisation d'un nombre important de macros-instructions souvent complexes. Là encore, les raisons de cette méthode de travail viennent de la facilité d'écriture, de modification, de compréhension et de transposition, facilité qui n'est pas

amoindrie par une mauvaise optimisation du programme généré, comme ce serait le cas si l'on utilisait un langage évolué.

Un dernier point n'est pas non plus à négliger: pour se définir une sorte de langage de macros-instructions, on doit d'abord bien définir ce qu'on a à faire, réfléchir aux facons de le réaliser, dégager les moyens fondamentaux, jusqu'à arriver à une suite restreinte d'instructions simples, bien définies, et réalisables sous forme de macros-instructions. On doit ensuite s'en tenir au schéma tracé par la définition de ce langage, c'est-à-dire que l'on est obligé de programmer de facon propre et simple, de s'interdire les "astuces" de programmation qui gagnent une micro-seconde ou un octet et rendent le programme illisible, d'éviter les licences d'écriture qui facilitent la tâche d'un programmeur paresseux mais font perdre toute sa généralite au travail. Enfin, comme un programme net et rigoureux ne peut aucunement exprimer des principes flous et mal définis, ces contraintes d'écriture réagissent de façon salutaire sur l'ensemble du travail, et aident à en améliorer la conception même.

6.22 - STRUCTURE GENERALE DU SYSTEME EULALIE.

La figure 6.4 donne un schéma global de la structure modulaire du système Eulalie. Je vais d'abord expliquer rapidement le rôle de chaque module, avant d'en détailler un peu plus le fonctionnement dans les paragraphes suivants.

La fonction maîtresse du système, même si c'est la plus simple, est ce qu'on pourrait appeler la fonction de commande: il s'agit de reconnaître le langage de commande défini au paragraphe 6.1, et de réaliser ou faire réaliser les actions demandées par les commandes. Ce travail est réparti entre trois modules: le module SYSTEME gère toutes les commandes, sauf celles qui découlent de déclarations ou instructions, qui sont dévolues au module DECLINST, et celles qui résultent de exécuter, dont se charge le module EXECUTER.

Les autres fonctions correspondent aux actions à effectuer pour le traitement des commandes; les principales concernent la traduction des déclarations ou instructions des programmes (module COMPILED pour les déclarations, COMPILEI pour les instructions), leur retraduction (module DCOMPILD pour les déclarations, DCOMPILI pour les instructions), le chargement d'un programme en mémoire (module CHARGEUR) et son exécution par interprétation (module INTERPRE). Les commandes table, copier et supprimer sont entièrement traitées

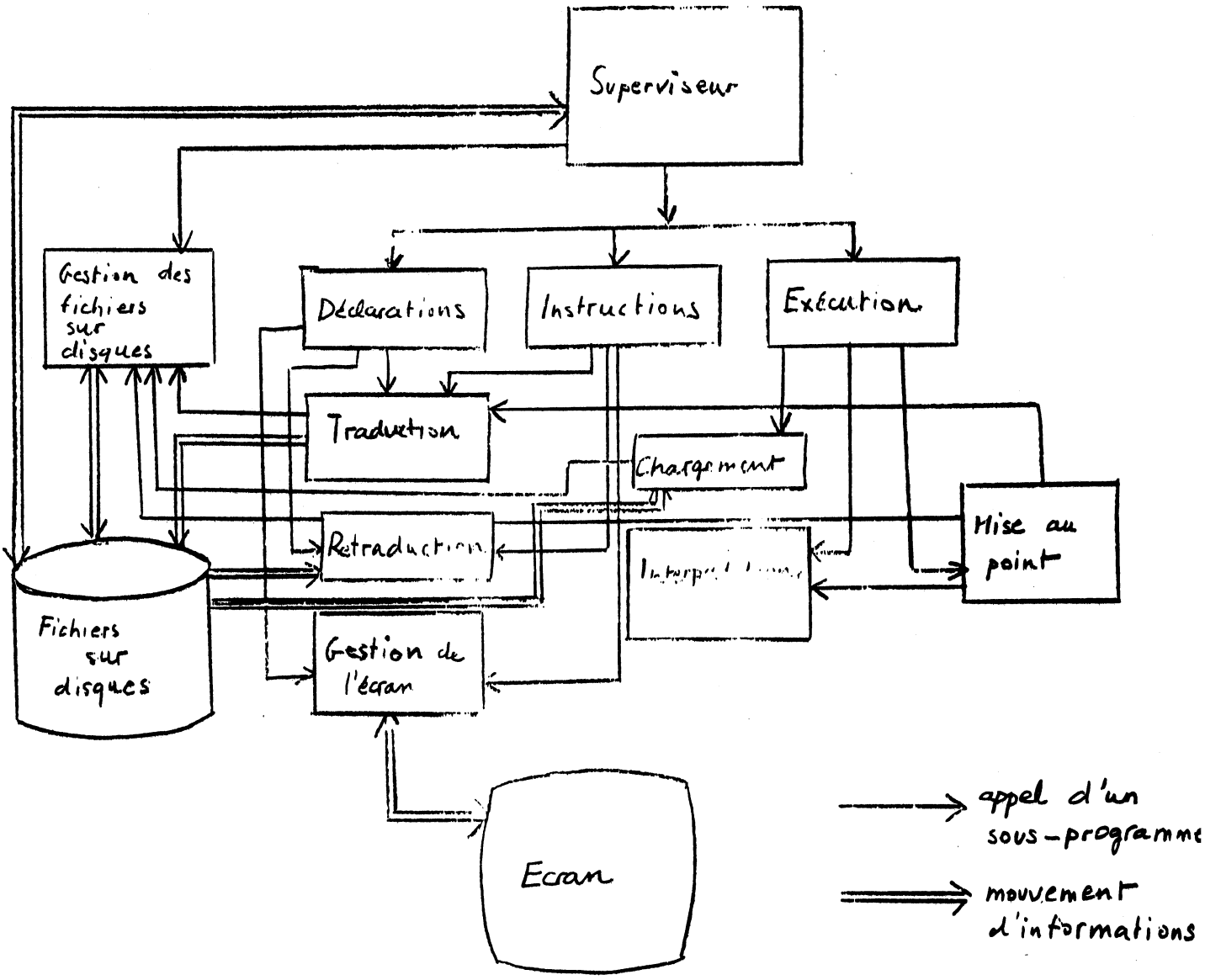


Figure 6.4 - Organisation générale du système Eulalie.

dans le module SYSTEME, et les commandes de mise au point dans le module EXECUTER.

Enfin, trois modules se chargent, par l'intermédiaire de sous-programmes nombreux, de fonctions complexes mais bien délimitées: le module SPDISQUE pour tout ce qui concerne la gestion des fichiers conservés sur disques, le module SPLECTUR pour toutes les fonctions confiées d'ordinaire à la phase d'édition d'un compilateur, et le module SPECRAN pour la gestion de l'écran pendant toute la durée des commandes déclarations et instructions.

6.221 - La gestion des disques.

L'ensemble des informations conservées sur disques, c'est-à-dire principalement les programmes des utilisateurs, est rangé sous forme d'éléments de 8 mots pleins ou 32 octets. Ces éléments sont groupés par 8 en blocs qui occupent 260 octets car chacun comporte en tête un mot qui sert à regrouper les blocs en liste. Sur une pile de disques 2314, on peut mettre 23 blocs par piste, 460 par cylindre et 92000 pour toute la pile, ces chiffres étant respectivement 11, 110 et 22000 pour une pile de disques 2311. A l'initialisation de la zône utilisée par Eulalie (fichier EULALIE FICHIERS en CMS, fichier repéré par la commande //FICHIERS DD ... en OS), tous les blocs sont reliés au moyen de leur premier mot en une liste des blocs libres, dans laquelle on puisera pour trouver des blocs, et dans laquelle on remettra tous les blocs qui ne sont plus utilisés.

Les éléments peuvent eux aussi être mis en liste, auquel cas il ne reste que 7 mots ou 28 octets pour ranger de l'information; on tient également à jour une liste des éléments libres, qui est vide au départ. Le premier élément du premier bloc est l'élément-maître; il contient toutes les informations qui permettent de travailler sur les blocs et les éléments conservés sur disques, c'est-à-dire les pointeurs LISBLOLI, LISELELI et LISINDIV, ainsi que le nom et le mot de passe du programmeur-maître. Ces deux dernières informations sont mises en place lors de l'initialisation de la zône de disques, mais elles peuvent être modifiées par le programmeur-maître (voir le paragraphe 6.153).

On se réfère à un bloc ou à un élément par ce que l'on appelle son numéro; pour un bloc, il s'agit simplement d'un numéro d'enregistrement physique. En règle générale, et en particulier pour les sous-programmes d'accès aux disques (voir le paragraphe 6.223), on peut considérer ce numéro comme une adresse et l'ensemble des disques comme une mémoire de très grande taille. Le numéro d'un élément est formé en ajoutant un octet qui donne le déplacement à l'intérieur d'un bloc aux trois octets qui donnent le numéro de bloc; cette technique permet de donner un numéro distinct à tous les octets conservés sur disques. On dispose donc pour chaque octet

conserve d'une "adresse" d'un mot plein, que l'on peut fournir telle quelle aux sous-programmes qui travaillent sur les blocs ou les éléments.

Comme on doit pouvoir se référer à n'importe quel bloc à tout moment, on utilise pour laisser au système des performances acceptables une technique de pagination semblable semblable à celle de J. Cohen [Co67], et que je vais expliquer brièvement.

Un certain nombre de blocs peuvent se trouver a un moment donné en mémoire; une table donne pour chacun de ces blocs son numéro et son adresse en mémoire. Quand on demande l'accès à un bloc, on regarde d'abord s'il figure dans la table; s'il y est, on trouve dans la table son adresse en mémoire, que l'on fournit comme résultat; s'il n'y est pas, on va le chercher sur disques et on l'amène en mémoire. Il faut pour cela le mettre a une place occupée présentement par un autre bloc; pour optimiser la succession des blocs en mémoire, on ajoute alors deux informations à la table descriptive: l'heure de derniere utilisation de chaque bloc, et un bit indiquant s'il a été modifie.

Quand on doit lire un bloc sur disques et le placer en mémoire, on cherche celui des blocs actuellement en mémoire qui n'a pas été utilisé depuis le plus longtemps. Si le bit de modification n'est pas en fonction, on se contente de mettre en place le bloc demandé, et de corriger le numéro indiqué dans la table. Si le bit de modification est en fonction, il faut recopier le bloc sur les disques pour tenir compte de ce dernier état; on le transfère d'abord dans une zône spéciale, on lit a sa place le bloc demandé, puis on lance l'opération de réécriture, qui pourra se dérouler en simultanéite avec le programme.

Cet algorithme a été longuement étudié par J. Cohen, et donne toutes satisfactions dans le cas ou le choix des blocs se fait de façon à peu pres aléatoire, ce qui est le cas quand on utilise les listes. Les performances sont d'autant meilleures que la place occupée en memoire par la table des blocs est plus grande, mais on est bien sur obligé de rester dans des limites raisonnables.

6.222 - Le superviseur.

Le superviseur (module SYSTEME) est le module qu'appelle le systeme hote a l'initialisation d'Eulalie. L'annexe 4 en donne une description a peu pres complete en langage Euphemie, aussi me contenterai-je de quelques indications sur son fonctionnement.

Le module SYSTEME contient certains paramètres d'intérêt général, qui sont nécessaires au fonctionnement d'ensemble; il s'agit de pointeurs sur les listes des blocs libres (LISBLOLI), des utilisateurs (LISINDIV), des éléments libres sur disques (LISELELI), des programmes de l'utilisateur courant (SESPROGR), des déclarations du programme courant (SESDECLA), de ses instructions (SESINSTR), de ses identificateurs (SESIDENT) et des valeurs de ses identificateurs ou constantes (SESVALEU), et enfin du compteur d'éléments retirés de la liste des éléments libres (NOMBELEM).

Le travail du superviseur commence par un certain nombre d'initialisations destinées à permettre le fonctionnement du module de gestion des disques; il faut en particulier réserver la zône de mémoire nécessaire à la table des blocs; la taille de cette zône peut être fixée par un paramètre fourni au moment de l'appel du système Eulalie. Une autre initialisation importante concerne la lecture sur les disques de l'élément-maître.

Le traitement des commandes est ensuite fait d'une façon simple et systématique, que facilite la définition de quelques macros-instructions, représentées par des procédures dans l'annexe 4. En particulier, la macro-instruction QUESTION permet d'effectuer toutes les opérations qui sont nécessaires pour poser une question à l'utilisateur au moyen d'un message affiché sur l'écran, lire sa réponse, et prendre une décision spéciale s'il a appuyé sur la touche fin au lieu de la touche fin de ligne. A chaque commande descendante est associé un niveau d'interruptions, ce qui rend extrêmement simple l'utilisation de la structure de commandes hiérarchisées décrite au paragraphe 6.121; en particulier, il suffit pour remonter dans la hiérarchie de terminer le niveau d'interruptions en cours: on retrouve intacts le niveau supérieur et l'ensemble de commandes qu'il autorisait.

Un problème délicat est de savoir quand doivent disparaître les messages que l'on affiche sur l'écran: d'une part, une question ne doit pas disparaître dès qu'on y a répondu, sauf cas spéciaux, ni un message d'erreur ou d'explication dès qu'on l'a affiché, mais d'autre part il faut éviter d'encombrer l'écran de messages périmés. L'utilisation de figures spécialisées facilite beaucoup ce travail: la figure BARATINS contient tous les messages du superviseur (questions, informations ou erreurs), la figure REPONSES reçoit les réponses de l'utilisateur, la figure JINFORME contient le message affiché pendant tout le cours de la commande nommer. Ainsi, la macro-instruction REPONSE, qui sert a afficher les messages d'information, peut-elle effacer de l'écran tout ce qui n'est plus

nécessaire dès que l'utilisateur envoie une nouvelle commande, sans pour cela perturber ce qui doit rester intact.

Un dernier travail important du module superviseur est la mise à jour des disques en fin d'utilisation du système, en particulier en ce qui concerne l'élément-maître. Cette mise a jour est un point particulièrement critique, car si une panne se produit avant qu'elle n'ait été faite, les informations qui subsistent sur les disques risquent de ne pas correspondre à la situation courante. Pour eviter autant que possible ce grave ennui, on peut multiplier les mises a jour, mais la véritable solution est de fournir un petit programme autonome et conversationnel d'examen et de correction de ce qui est conservé sur disques; on peut alors revenir à une situation normale sans avoir perdu trop d'informations.

6.223 - Les sous-programmes d'accès aux disques.

Les sous-programmes d'accès aux disques sont tous contenus dans le module SPDISQUE; ils utilisent les paramètres fondamentaux initialisés et conservés dans le module SYSTEME, c'est-à-dire LISBLOLI, LISELELI et NOMBELEM. Tous ces sous-programmes sont appelés depuis à peu près tous les autres modules, par l'intermédiaire de macros-instructions très simples qui servent simplement à harmoniser la nature des paramètres fournis avec ce qu'attendent les sous-programmes.

Je me contenterai de donner la liste des sous-programmes avec leurs paramètres et le rôle qui leur est dévolu, et en plus dans quelques cas des indications sur leur fonctionnement.
- LIREBLOC numéro: lecture du bloc dont on donne le numéro, d'après les principes donnés au paragraphe 6.221; fournit comme résultat l'adresse en mémoire du bloc.
- LIRELEM numero: même chose pour un élément.
- PRENDBLO: acquisition d'un bloc pris dans la liste des blocs libres; fournit comme résultat le numéro et l'adresse de ce bloc.
- PRENDELE: même chose pour un élément; quand la liste des éléments libres est épuisée, ce sous-programme demande un nouveau bloc pour la compléter.
- RENDBLOC numero: restitution à la liste des blocs libres (en tête) du bloc dont on donne le numéro.
- RENDELEM numéro: même chose pour un élément.
- RENDLIBL numéro: restitution d'une liste de blocs enchaînés par leur premier mot, et dont on donne le numéro du premier bloc; la fin d'une liste est marquée par un premier mot nul.
- RENDLIEL numéro: même chose pour une liste d'éléments.
- CHERELEM numéro, condition: recherche du premier élément d'une

liste (dont on donne le numéro du premier élément) qui satisfasse à un certain critère, représenté par une instruction ou un appel de sous-programme; le résultat est le numéro et l'adresse de l'élément demandé, ou l'indication dans le code de condition que la recherche a été infructueuse.
- METRELEM numéro, numéro: adjonction en tête de la liste indiquée par le premier paramètre de l'élément indiqué par le deuxième paramètre; ceci pose un petit problème bien connu des langages de liste [Le66-1], qui nécessite l'échange du contenu de l'élément à ajouter et du premier élément de la liste, et interdit l'usage de ce sous-programme dans le cas d'une liste vide.
- OTERELEM numéro: suppression d'un élément de la liste dans laquelle il se trouve; il faudra fermer la coupure produite, ce qui pose un deuxième problème bien connu, et facile à résoudre sauf dans le cas de l'élément queue de liste.
- COPILIEL numéro: recopie de la liste d'éléments indiquée, le résultat étant le numero du premier élément de la liste produite.
- MODIF: indication de modification du dernier bloc ou élément lu.

L'utilisation d'une section de contrôle fictive permet la référence symbolique à tous les composants des éléments, ce qui facilite énormément le travail; la seule précaution à prendre est de tenir compte que l'on n'a aucun garantie qu'un élément ou un bloc soit encore en mémoire après qu'on en ait utilisé un autre.

6.22⁴ - Gestion de l'écran.

Le module de gestion de l'écran (SPECRAN) contient un certain nombre de sous-programmes appelés par le module DECLINST, et sert à séparer complètement les fonctions d'interprétation des commandes, les fonctions de traduction et de retraduction, et les fonctions d'affichage sur l'écran. Ce module établit la liaison avec les modules de traduction (COMPILED et COMPILEI) et de retraduction (DCOMPILD et DCOMPILI), qui peuvent ainsi travailler sur des chaînes sans se soucier d'où elles viennent ni où elles vont.

Parmi les règles imposées à ce module, on peut citer les suivantes:
- il faut travailler logiquement sur des instructions, mais physiquement sur des lignes;
- il faut pouvoir déplacer les marqueurs devant les instructions, faire avancer ou reculer le texte page par page, ajouter ou retirer des lignes ou des instructions;
- une instruction ne doit pas figurer sur l'écran de facon incomplète, coupée par le haut ou le bas de la zône d'affichage;
- les symboles des instructions ne doivent pas etre coupés lors du

passage d'une ligne a l'autre;
- seule doit pouvoir être modifiable au moyen du clavier alphanumérique l'instruction en cours, et jamais les autres;
- toutes les modifications apportées au texte affiché doivent se refléter sur le programme conservé sur disques en langage intermédiaire.

Les principaux sous-programmes qui composent ce module sont les suivants:
- LAFFICHE fait apparaître sur l'écran deux pages de texte (déclarations ou instructions), après avoir effectué toutes les initialisations nécessaires pour le bon fonctionnement du module;
- LAJOUTER ajoute une ligne blanche modifiable sous la ligne courante, avec le curseur en tête;
- LAVANCER avance d'une page dans le texte;
- LCORRIGE rend modifiable l'instruction courante;
- LDESCEND descend les marqueurs d'une instruction;
- LEFFACE efface tout le texte qui apparaît sur l'écran, et restitue la place utilisée par le module;
- LENLEVER supprime une ou plusieurs instructions (quand l'utilisateur envoie la commande fin sous contrôle de la commande enlever);
- LMONTER remonter les marqueurs d'une instruction;
- LRECULER recule d'une page dans le texte;
- LRETRADU appelle le retraducteur approprié en lui fournissant l'instruction courante, et affiche sur l'écran la chaîne fournie à la place de l'instruction courante;
- LTRADUIT appelle le traducteur approprié en lui fournissant l'instruction courante, et range éventuellement sur disques le fragment de langage intermédiaire fourni comme résultat.

Le module DECLINST, qu'il est inutile de décrire, a simplement à appeler ces sous-programmes suivant la logique des commandes envoyées par l'utilisateur. Les quelques sous-programmes non cités servent aux initialisations et terminaisons pour certaines commandes, et à l'affichage et l'effacement des messages d'erreur.

Chaque modification d'une ligne de l'écran nécessite le réaffichage de tout ce qui apparaît au-dessous. Pour faciliter ce travail, le module SPECRAN utilise deux tables organisées en listes symétriques, l'une pour les instructions et l'autre pour les lignes qui la composent. Des sous-programmes internes effectuent alors des manipulations de ces tables, pour ajouter une ligne a une instruction, supprimer des lignes, insérer une instruction après l'instruction courante, retirer une instruction. D'autres sous-programmes servent à découper en lignes le texte en clair d'une instruction (en respectant les règles exposées au paragraphe 6.131), à placer les

marqueurs en fonction du numéro de la ligne courante, à gérer les différents messages d'information qui apparaissent en bas d'écran, à supprimer une instruction du langage intermédiaire (en effacant également son étiquette si elle en avait une).

Dans ce module encore, un petit jeu de macros-instructions bien définies sert à la réalisation de toutes les fonctions élémentaires, avec tous les avantages déjà indiqués.

6.225 - Traduction et retraduction des programmes.

Les problèmes de traduction et de retraduction sont évoqués plus longuement qu'ils ne le seront ici en référence [Cl69]. Je me contenterai de signaler brièvement quelques points intéressants. Il n'y a en fait rien de révolutionnaire dans cette partie du système Eulalie, le seul côté qui présente quelque originalite étant lié au procédé de retraduction qui est utilisé. Pour que l'utilisateur puisse examiner son programme après l'avoir composé, il faut en efet qu'on lui présente le texte sous une forme aussi proche que possible de celle qu'il a frappée. On pourrait pour cela conserver le programme sous deux formes différentes, le texte source et le texte objet; outre que le texte source serait extrêmement encombrant, ce procédé s'adapte très mal à la compilation conversationnelle et incrémentielle. Dans le système Eulalie, on ne conserve donc les programmes que sous une seule forme, dite langage intermédiaire: ce langage est conçu pour pouvoir d'une part être interprété pendant la phase d'exécution, d'autre part retraduit en langage source pendant la phase de compilation.

Pour préciser un peu plus la situation, un programme est représenté sur disques par quatre listes d'éléments:
- la liste des instructions contient ce qui constitue réellement le langage intermédiaire, c'est-à-dire une représentation condensée et post-fixee (pour être interprétable) des instructions du programme;
- la liste des déclarations contient une représentation condensée du texte des déclarations, qui sert uniquement à en retrouver la représentation en clair; on ne peut pas en effet appliquer ici la même principe que pour les instructions;
- la liste des identificateurs représente la signification des déclarations; elle contient pour chaque identificateur son nom en clair et ses caracteristiques, indiquées par des profils binaires et des pointeurs;
- la liste des valeurs contient toutes les valeurs des constantes utilisées dans le programme, et les valeurs initiales des identificateurs; les noms des constantes apparaissent comme des identificateurs dans la liste précédente.

Les instructions sont représentées dans le langage intermédiaire sous une forme simple et systématique: à chaque instruction correspond une suite d'éléments; on trouve en tête la nature de l'instruction, qui correspondra à l'exécution à un sous-programme particulier, puis la liste des paramètres de ce sous-programme, et enfin la transcription post-fixee des expressions qui peuvent servir à l'évaluation de certains de ces paramètres. Tous les éléments de la liste des instructions sont chaînés entre eux dans les deux sens, pour faciliter les modifications.

Les références aux variables ou constantes sont remplacées dans le langage intermédiaire par des références a la liste des identificateurs; les caractéristiques des variables ou constantes figurent dans les mêmes mots que les références, mais la liste des identificateurs en donne une description plus détaillée. Pendant tout le travail de retraduction, cette liste doit être perpétuellement consultée, et on ne peut pas la classer pour accélérer les recherchés; on utilise donc une méthode d'adressage dispersé ("hash-coding"), qui associe à chaque identificateur un nombre pseudo-aléatoire de six positions binaires; la liste des identificateurs est alors divisée en soixante-quatre parties, à l'intérieur desquelles on effectue une recherche séquentielle qui n'est jamais très longue.

Les traducteurs et retraducteurs sont tous quatre écrits suivant les mêmes principes, c'est-à-dire à l'aide de quelques macros-instructions qui réalisent les traitements élémentaires en appelant des sous-programmes. Le fonctionnement de ces quatre modules est calqué d'assez près sur la structure du langage Euphémie, ce qui ne présente pas d'inconvenients de principe puisque les macros-instructions rendent les modificatons très simples. Les seules parties qui font appel à des méthodes de compilation générales sont celles qui concernent la traduction et la retraduction des expressions arithmetiques, booléennes ou de chaînes: ce travail est fait de façon très classique, avec une pile d'opérateurs, et une pile d'opérandes pour les vérifications de type nécessaires en traduction [BeLe65-2].

Les deux retraducteurs sont beaucoup plus simples que les traducteurs correspondants, car ils n'ont a faire aucune vérification et ne peuvent jamais rencontrer d'erreur.

6.226 - Chargement et exécution des programmes.

La forme intermédiaire sous laquelle les programmes sont conservés sur disques n'est pas directement utilisable par le module

d'interprétation. Elle doit donc être traitée par un module de chargement (CHARGEUR), qui effectue les opérations suivantes:
réservation en mémoire de l'emplacement nécessaire à toutes les variables et constantes, au moyen de la liste des identificateurs;
- initialisation des variables et constantes, au moyen de la liste des valeurs;
- constitution d'une table de correspondance de l'adresse sur disques et de l'adresse en mémoire de chaque variable; cette table est utilisée pendant la suite du chargement et pendant les opérations de mise au point;
- chargement des instructions elles-mêmes, en remplaçant tous les pointeurs sur disques par des pointeurs en mémoire, au moyen de la table précédente, et en remplacant les noms d'instructions par les adresses des sous-programmes correspondants;
- constitution d'une table des étiquettes, qui résoud le problème des références d'une instruction à l'autre, non traité au moment de la traduction.

L'exécution du programme est réalisée de façon interprétative par le module INTERPRE, suivant un schéma très simple:
- recopie de la liste de paramètres de l'instruction dans une zône de travail;
- balayage de cette liste pour y rechercher les expressions non évaluées;
- évaluation des expressions;
- remplacement de chaque référence à une expression par la référence au résultat de l'évaluation de cette dernière;
- appel du sous-programme voulu en lui communiquant la liste des paramètres ainsi constitués.

L'évaluation des expressions est faite suivant une technique de pile bien connue [BeLe65-2, Bo67]; il n'y a aucune vérification à faire à ce niveau, en particulier tous les problèmes de conversion de types sont traités au moment de la traduction; parmi les opérateurs qui figurent dans les expressions, on trouve donc des opérateurs arithmétiques à opérandes entiers distincts des opérateurs arithmétiques à opérandes réels, et des opérateurs de conversion pour les six types possibles.

6.23 - EXTENSIONS POSSIBLES.

Le système Eulalie tel qu'il est défini dans ce qui précède est assez simple, quelquefois presque trop; les extensions qui seraient intéressantes concernent presque toujours des perfectionnements qui doivent faciliter le travail de l'utilisateur. Comme toujours en

pareil cas, c'est surtout l'expérience des usagers qui pourra suggérer les meilleures améliorations, mais on peut déjà cependant en prévoir quelques-unes, que je cite sans ordre particulier:
- possibilite d'accès d'un utilisateur aux programmes des autres ou plutôt à certains programmes de certaines personnes. Ceci réclame une commande de définition du mode d'accès à un programme, que l'on doit pouvoir utiliser à tout moment et non pas seulement lors de la création dudit programme. D'autre part, il faut pouvoir indiquer quand on donne le nom d'un programme qu'il appartient à un autre utilisateur.
- Possibilite de concaténation de plusieurs programmes ou fragments de programmes pour en constituer un nouveau. Ceci nécessiterait une compilation préalable de tout le programme ainsi constitué, comme dans le cas des modifications aux déclarations (voir le paragraphe 6.131). La commande à ajouter devrait se placer immédiatement sous le contrôle de nommer, comme copier et supprimer.
- Possibilite de rangement et de lecture d'un programme sur un milieu extérieur (cartes ou bandes magnétiques), pour récupérer la place occupée sur disques par des programmes peu utilisés, et permettre un traitement en differé (perforation des cartes, lecture, compilation, impression de la liste du programme avec les messages d'erreurs éventuels).
- Possibilité de corrections réelles à un programme pendant son exécution, c'est-à-dire de modifications, adjonctions et suppressions d'instructions. Plus généralement, le système Eulalie devrait permettre l'étude des problèmes de mise au point conversationnelle des programmes a l'aide d'un terminal graphique, que j'ai déjà signalés à la fin du praragraphe 3.4 comme une voie de recherches très ouverte.

Bien d'autres perfectionnements et extensions sont encore possibles; je pense en tout cas qu'un point capital serait de les insérer convenablement dans la structure hiérarchisee du langage de commandes, qui est un des points les plus caractéristiques du système Eulalie.

CHAPITRE 7 - CONCLUSIONS.

> Icy je feray fin à ce livre; la teste me faict un peu de mal, et sens bien que les registres de mon cerveau sont quelque peu brouillez de ceste purée de septembre.
> Rabelais (Pantagruel, XXXIV)

Il est difficile de tirer de véritables conclusions sur ce travail, car il n'est pas achevé: en effet, d'une part le langage Euphémie est insuffisant dans son état actuel, ainsi que le système Eulalie, d'autre part la réalisation de l'ensemble n'est pas encore opérationnelle et l'expérimentation n'a donc pas pu commencer. Je voudrais par conséquent simplement signaler les points que je juge les plus importants et les plus positifs dans mon travail, ceux que j'espère en particulier pouvoir appliquer ailleurs.

La terminologie concernant les traitements graphiques (chapitre 2 et annexe 2) a été établie en commun avec d'autres personnes, en particulier M. Lucas. Elle est cohérente et structurée, et elle s'est déjà montré utilisable; il devra être facile de la compléter dans les domaines où elle est encore incomplète, en particulier du côté des structures [Ca70-2].

Les principes d'organisation donnés au paragraphe 3.1, et qui se fondent sur le niveau des langages de programmation et la puissance des outils que ceux-ci fournissent, représentent une autre tentative de mise en ordre que j'espère intéressante: sans peut-être pouvoir aller toujours jusqu'à la manière ultra-cartésienne de construction d'Algol 68, on doit tout de même à mon avis s'astreindre à une certaine discipline quand on construit un langage de programmation, pour espérer aboutir à quelque chose qui tienne debout.

Au sujet du langage Euphémie, je retiendrai trois points que je juge importants: le premier est l'existence même de ce langage; le deuxième est son aspect et sa structure syntaxiques; le troisième est la façon dont il est employé. Au total, j'espère qu'Euphémie se montrera un langage utilisable, les nombreux exemples que j'ai rédigés n'en étant pas encore une preuve suffisante.

Quant au système Eulalie enfin, deux points me semblent intéressants: la structure hiérarchisée de son langage de commande, que je

pense être applicable à d'autres systèmes et susceptible de notablement les améliorer, et l'utilisation très importante de macros-instructions dans la réalisation même du système, qui s'est montré une méthode de travail très fructueuse.

Je voudrais terminer sur trois points d'une nature assez différente: il s'agit des conditions dans lesquelles ce travail a été fait, de son utilité et de sa poursuite éventuelle.

Tout d'abord, les deux personnes qui ont le plus travaillé à Eulalie et Euphémie, c'est-à-dire E. Cleemann et moi-même, sont toutes deux des enseignants de la Faculté des Sciences de Grenoble, successivement assistants puis maîtres-assistants. Il est certain qu'il ne s'agit pas là de la situation la meilleure pour faire un travail de recherches: l'horaire d'enseignement que doit assurer un assistant ou un maitre-assistant est le triple ou le quadruple de celui d'un maître de conférences ou d'un professeur, les matières enseignées sont encore plus souvent que pour ces derniers à réapprendre complètement en debut d'année scolaire, et les étudiants sont beaucoup plus encombrants. Autant dire que le travail de recherches est fait plus ou moins à temps perdu, avec de longues périodes d'interruptions. Il n'y a donc pas lieu de s'étonner si cette thèse n'a pas été soutenue un an plus tôt, et si le système qui en est l'objet n'est pas encore operationnel.

Je voudrais ensuite insister sur l'utilité de ce travail, car c'est un des points qui me tiennent le plus a coeur. Le système Eulalie et le langage Euphémie ont été concus pour être utilisables et utilisés, même si cela devait se répercuter de façon fâcheuse sur leur élégance et leur généralite. Mon ambition est qu'ils puissent servir dans une première etape aux travaux pratiques d'étudiants, puis à tous les utilisateurs du matériel graphique de l'Institut de Mathématiques Appliquées de Grenoble. Ce système est utilisable et utile si l'on peut réaliser avec tout ce qui actuellement se fait en Fortran avec G.S.P.[Sa70]; il n'a pas la prétention de remplacer le travail en langage d'assemblage [Ad70-2, Ca70-2], ou dans des langages spécialises [Si70-1 et 2]. Je pense en fait que ce système peut être la premiere approche d'un produit commercialisable, et j'espère que l'avenir ne me contredira pas.

Si ce système est utile, il serait absurde que l'on arrête sur lui tout travail après la soutenance de cette these: cela éviterait à celle-ci de courir le risque d'être une démonstration académique et stérile. Je compte pour ma part appliquer les principes que j'ai exposés au matériel dont dispose l'Université de Montréal, et qui est encore plus sous-employé que celui de Grenoble; comme il s'agit

d'un matériel produit par un autre constructeur et avec des caractéristiques très différentes de celui de Grenoble, l'expérience ne peut être que fructueuse. D'autre part j'estime indispensable que la réalisation effective du système Eulalie puisse être menée à bien à Grenoble, ce qui peut être fait à temps pour que des travaux pratiques puissent avoir lieu au deuxième semestre de la présente année scolaire. Il s'agit de la meilleure expérimentation possible pour un système de ce genre, et s'il s'en sort sans trop de mal on peut être assuré qu'il pourra servir à quelque chose.

A.11 - CLASSEMENT ALPHABETIQUE.

Ad70-1 : Michel Adiba
Le langage Snobol4
Note technique IMAG et Séminaire de Programmation
Janvier 1970

Ad70-2 : Michel Adiba
Edition graphique
Colloque sur les traitements graphiques
Avril 1970

BeLe65-1 : Camille Bellissant et Olivier Lecarme
Compilateur Algol sur IBM 1130
Notice d'utilisation
Juin 1965

BeLe65-2 : Camille Bellissant et Olivier Lecarme
Compilateur Algol sur IBM 1130
Notice de fonctionnement
Novembre 1965

Bo67 : Louis Bolliet
Notation et processus de traduction des langages symboliques
Thèse de doctorat d'Etat es Sciences Appliquées
Juin 1967

BU.. : Brown University
BRUIN (Brown University Interpretor)

Co67 : Jacques Cohen
Langages d'écriture de compilateurs
Thèse de doctorat d'Etat ès Sciences Appliquées
Juin 1967

Cl69 : Edouard Cleemann
Un macro-langage de programmation graphique
Thèse de troisieme cycle
Mars 1969

CLL68 : Edouard Cleemann, Olivier Lecarme et Michel Lucas
Les langages de programmation graphique
Revue Francaise d'Informatique et de Recherche Opérationnelle
Décembre 1968

Ca69 : Jean-Michel Cagnat
Structures pour la représentation des données en programmation graphique
Séminaire de programmation
Mai 1969

Ca70-1 : Jean-Michel Cagnat
Composition simultanée d'une figure en forme de graphe et de la structure correspondante au moyen d'un terminal graphique
Colloque sur les traitements graphiques
Avril 1970

Ca70-2 : Jean-Michel Cagnat
(titre non encore determiné)
Thèse de troisième cycle
(date non encore determinée)

ChDo68 : F. C. Chen et R. L. Dougherty
A system for implementing interactive applications
IBM systems journal n° 3-4
Décembre 1968

ChPi65 : Carl Christensen et Elliott N. Pinson
Multifunctions graphics for a large computer system
AFIPS Conference proceedings
1967 Fall Joint Computer Conference

DaEl64 : M. R. Davis et T. O. Ellis
The Rand tablet: a man-machine communication device
AFIPS Conference proceedings
1964 Fall Joint Computer Conference

Gr69 : Michael Griffiths
Analyse deterministe et compilateurs

Thèse de doctorat d'Etat ès sciences appliquées
Novembre 1969

Gu69 : Richard Guedj
Système "Grace". Incidence du software sur la conception d'une console graphique de visualisation
Deuxième journée de visualisation de l'AFCET et Séminaire de programmation de l'IMAG
Octobre 1969

GPP68 : R. E. Griswold, J.F. Poage et I. P. Polonsky
The SNOBOL4 programming language
Prentice-Hall, 1968

Gr61 : A. A. Grau
The structure of an Algol translator
Contract N° W-7405-eng-26, Oak Ridge National Laboratory
Février 1961

HCY67 : A. Hurwitz, J. P. Citron et J. P. Yeaton
GRAF: Graphic additions to Fortran
AFIPS Conference proceedings
1967 Spring Joint Computer Conference

HBB68 : J. L. Haour, S. Brehinier et C. Boksenbaum
LAPREC: Langage de programmation pour écran cathodique
Revue Francaise d'informatique et de recherche opérationnelle
Décembre 1968

Ha69 : Jean-Louis Haour
LAPREC
Deuxième journée de visualisation de l'AFCET
Octobre 1969

Ha70 : Jean-Louis Haour
LAPREC et sa compilation
Colloque sur les traitements conversationnels
Avril 1970

HaLe70 : Claude Hans et Olivier Lecarme
Support graphique pour terminal 2250 sous CMS
Note technique IMAG

IBM1 : INERNATIONAL Business Machines Corporation
GPAK - An on-line system/360 graphic data processing

subroutine package with real time 2250 input and display
System/360 general program library 360D-3.4.005

IBM2 : International Business Machines Corporation
IBM System/360 operating system graphic programming services for Fortran IV
System/360 reference library, C27-6932

IBM3 : International Business Machines Corporation
IBM 2250 display unit model 1
IBM System/360 component description, A27-2701

IBM4 : International Business Machines Corporation
OS/360 concepts and facilities
System/360 reference library, C28-6535

IBM5 : IBM Cambridge scientific center
CP/CMS user's guide
Report 320-2015

IBM6 : International Business Machines Corporation
RAX

IBM7 : International Business Machines Corporation
OS/360 graphic programming services - Basic
System/360 reference library, C27-6912

IBM8 : International Business Machines Corporation
PL/I language specifications
System/360 reference library, C28-6571

IBM9 : International Business Machines Corporation
TEXT360
System/360 general program library, 360D.29.4.001

IBM10 : International Business Machines Corporation
Problem language analyser (PLAN)
System/360 application program 360A-CX-27X

IBM11 : International Business Machines Corporation
PLAN graphics support for the IBM 2250
System/360 application program 360A-CX-34X

IBM12 : International Business Machines Corporation
IBM 1130/2250 graphic subroutine package
C27-6934

IBM13 : International Business Machines Corporation
IBM System/360 operating system assembler language
System/360 reference library, C28-6514

IBM14 : International Business Machines Corporation
IBM System/360 operating system
Graphic programming services for Fortran IV
Program logic manual, Y27-7152

IBM15 : International Business Machines Corporation
IBM System/360 time sharing system concepts and facilities
System/360 reference library, C28-2003

Jo68 : Craig I. Johnson
Ab experimental PL/I extension for graphic programming
IBM Cambridge scientific center
Report 320-2025

Jo63 : T. E. Johnson
Sketchpad III - 3D graphical communication with a digital computer
AFIPS conference proceedings
1963 Spring Joint Computer Conference

Ku68 : H.E. Kulsrud
A general purpose graphic language
Communications of the ACM
Avril 1968

Ko68 : P. S. Kopel
Interactive computer graphics
Bell laboratories record
Juin 1968

Le64 : Olivier Lecarme
Etude comparative de cinq langages utilisant les listes
Séminaire de programmation de l'IMAG
Novembre 1964

Le65-1 : Olivier Lecarme
Etude comparative des principaux langages de

programmation
Thèse de troisieme cycle
Juin 1965

Le65-2 : Olivier Lecarme
Cours de langage machine IBM 7040/44, tome I, deuxième partie (les autres parties sont de MM. Auroux, Mermet et Veyrunes)
Novembre 1965

Le66-1 : Olivier Lecarme
Langage IPL-5: Manuel de références pour programmeurs
Janvier 1966

Le66-2 : Olivier Lecarme
Ecriture en Algol de système de programmation
Cinquième congres de l'AFIRO
Juin 1966

Le67 : Olivier Lecarme
Interface du compilateur COMIT II pour fonctionnement sous controle du système IBSYS du 7040/44
Programme distribué par l'organisation SHARE
Septembre 1967

Le68 : Olivier Lecarme
Un système de programmation graphique conversationnelle
Colloque sur les systèmes conversationnels
Novembre 1968 (Monographie d'Informatique N° 6, Dunod)

Le69-1 : Olivier Lecarme
Le langage AED
Séminaire de programmation de l'IMAG
Avril 1969

Le69-2 : Olivier Lecarme
An online graphic programming system
Propose a l'Annual Review of Computer Programming

Le69-3 : Olivier Lecarme
Niveau des langages et puissance des outils en programmation graphique
Deuxième journée de visualisation de l'AFCET et note technique IMAG
Octobre 1969

Le69-4 : Olivier Lecarme
Les langages de programmation spécialisés
Traité pratique d'informatique (Techniques de l'ingenieur, volume H1)
Novembre 1969

Le70-1 : Olivier Lecarme
Les systèmes d'exploitation des ordinateurs
Cours de l'Institut de Programmation
Avril 1970

Le70-2 : Olivier Lecarme
Niveau des langages et puissance des outils en programmation graphique
Revue francaise d'informatique et de recherche opérationnelle
Juin 1970

Le70-3 : Olivier Lecarme
Programmation et dialogue avec le système Eulalie et le langage Euphémie
Congrès d'informatique de l'AFCET
Septembre 1970

LeCl68 : Olivier Lecarme et Edouard Cleemann
Système d'utilisation rationnelle d'un terminal graphique évolué
Séminaire de programmation de l'IMAG
Janvier 1968

LeCl70 : Olivier Lecarme et Edouard Cleemann
Système de programmation graphique conversationnelle
Colloque sur les traitements graphiques
Avril 1970

LeLu68 : Olivier Lecarme et Michel Lucas
LAGROL: un langage pour l'utilisation d'un terminal graphique
Note technique IMAG
Janvier 1968

LeSa70 : Olivier Lecarme et Jean-Claude Saillard
Manuel d'utilisation de G.s.p.
Note technique IMAG

LTD69 : F. Lustman, B. Teiling et M. Duguet
C.I.C. - A conversational system for Fortran programming
SEAS meeting
Novembre 1969

Lu68-1 : Michel Lucas
Expériences sur un terminal graphique élémentaire
Séminaire de programmation de l'IMAG
Janvier 1968

Lu68-2 : Michel Lucas
Techniques de programmation et d'utilisation en mode conversationnel des terminaux graphiques
Thèse de troisieme cycle
Juin 1968

Lu68-3 : Michel Lucas
Un système conversationnel pour la construction et la manipulation d'images
Colloque sur les systèmes conversationnels
Novembre 1968 (Monographie d'informatique N° 6, Dunod)

Lu70 : Michel Lucas
Production de dessins animés à l'aide d'un terminal graphique
Colloque sur les traitements graphiques
Avril 1970

Mo67 : R. A. Morrison
Graphic language translation with a language-independent processor
AFIPS Conference proceedings
1967 Fall Joint Conference Conference

Ni65 : W. H. Ninke
Graphic I - A remote display console system
AFIPS conference proceedings
1965 Fall Joint Computer Conference

Pa68 : R. J. Pankhurst
GULP - A compiler-compiler for verbal and graphic languages
Proceedings of 1968 ACM National Conference

Po68 : R. J. Popplestone et R. M. Burstall
POP-2 reference manual
Oliver et Boyd, 1968

Ro64 : D. T. Ross et autres
AED kit
Massachusetts Institute of Technology

Ri67 : H. J. Ridinger
The Grafacon man-machine interface
Datamation
Juillet 1967

RHC65 : D. E. RIPPY, D. E. Humphries et J. A. Cunningham
MAGIC: Machine for automatic interface to a computer
AFIPS Conference proceedings
1965 Fall Joint Computer Conference

Ru68 : A. D. Rully
A subroutine package for Fortran
IBM systems journal N° 3-4
Décembre 1968

Sa70 : Jean-Claude Saillard
Utilisation d'un terminal graphique pour la conception des masques d'une plaquette de circuit integre
Colloque sur les traitements graphiques
Avril 1970

SBV68 : L. Siret, M. Bellot et J. P. Verjus
Diamag 2, système conversationnel à accès multiple
Revue Francaise d'Informatique et de Recherche Opérationnelle
Septembre 1968

Si70-1 : Yvon Siret
Méthode d'affichage des formules
Colloque sur les traitements graphiques
Avril 1970

Si70-2 : Yvon Siret
Contribution au calcul formel par ordinateur
Thèse de doctorat d'Etat ès sciences appliquées
Juin 1970

Sy68 : A. J. Symonds
Auxiliary-storage associative data structure for PL/I
IBM systems journal N° 3-4
Décembre 1968

St65 : T. G. Stockham, Jr.
Some method of graphical debugging
Proceedings of the IBM scientific computing symposium on man-machine communication, 1965

Su63 : T. E. Sutherland
Sketchpad - A man-machine graphical communication system
AFIPS Conference proceedings
1963 Spring Joint Computer Conference

Su65 : I. E. Sutherland
Three kinds of graphic processing
1965 IFIPS Congress

Su66 : I. E. Sutherland
Computer graphics: Ten unsolved problems
Datamation
Mai 1966

TeSa68 : J. R. Teixeira et R. P. Sallen
The Sylvania data tablet: a new approach to graphic data input
AFIPS Conference proceedings
1968 Spring Joint Computer Conference

Ve68 : J. P. Verjus
Etude et réalisation d'un système Algol conversationnel
Thèse d'ingenieur-docteur
Juillet 1968

A.12 - CLASSEMENT ANALYTIQUE.

A.121 - TRAVAUX PERSONNELS.

Le64 - Le65-1 - Le65-2 - Le66-1 - Le66-2 - BeLe67-1 - Le67 - BeLe67-2 - LeLu68 - LeCl68 - Le68 - CLL68 - Le69-1 - Le69-2 - Le69-3 - Le69-4 - Le70-1 - HaLe70 - LeSa70 - LeCl70 - Le70-2 - Le70-3

A.122 - TRAVAUX DE L'IMAG.

Ad70-1 - Ad70-2 - Bo67 - Co67 - Cl69 - Ca69 - Ca70-1 - Ca70-2 - Gr69 - Lu68-1 - Lu68-2 - Lu68-3 - Lu70 - Sa70 - SBV68 - Si69 - Si70-1 - Si70-2 - Ve68 - Si70-2

A.123 - LANGAGES DE PROGRAMMATION GRAPHIQUES.

Cl69 - CLL68 - Ca69 - ChDo68 - ChPi65 - Gu69 - HCY67 - HBB68 - Ha69 - Ha70 - HaLe70 - IBM1 - IBM2 - IBM7 - IBM11 - IBM12 - Jo68 - Ku68 - Ko68 - Le69-2 - Le69-3 - Le70-2 - Le70-3 - LeCl68 - LeCl70 - LeLu68 - LeSa70 - Lu68-1 - Lu68-2 - Mo67 - Ni65 - Pas68 - Ru68 - Sy68 - Su70-1 - Si70-2

A.124 - SYSTEMES GRAPHIQUES CONVERSATIONNELS.

Ad70-2 - Ca70-1 - Ca70-2 - ChDo68 - ChPi65 - Jo63 - Ko68 - Le68 - Le69-2 - Le70-3 - LeCl68 - LeCl70 - LTD69 - Lu68-2 - Lu68-3 - Lu70 - Ni65 - Pa68 - Sa70 - Si69 - Si70-1 - Si70-2 - St65 - Su63

ANNEXE 2 - GLOSSAIRE.

Asynchrone : Deux ensembles d'operations successives sont dits asynchrones si le deroulement des operations d'un ensemble n'est pas conditionne par le deroulement des operations de l'autre, et vice-versa; on peut dire aussi que l'on n'a aucun moyen, de l'interieur d'un des ensembles, de connaitre et de prevoir le deroulement de l'autre. Ainsi, l'execution d'un programme dans l'ordinateur et les actions de l'utilisateur devant son terminal sont naturellement asynchrones; c'est le role du traitement des interruptions° que de synchroniser ces deux processus.

Boule roulante (rolling ball, crystal ball) : dispositif de communication°, forme d'une sphere affleurant une table, que l'on peut faire pivoter dans toutes les directions et a la vitesse que l'on desire; le resultat de ce mouvement, par le jeu de potentiometres et d'un convertisseur analogique-digital, est de produire une information de direction et une information de vitesse sous la forme d'un vecteur dont on fournit les composantes en x et y. On interprete le plus souvent cette information comme l'indication d'un deplacement° a faire subir a un objet° ou une figure° de l'image affichee; a la difference du manche a balai°, l'information cesse d'exister si l'on interrompt le mouvement.

Caractere: Un des elements° fondamentaux affichables par la plupart des terminaux graphiques°; sur les modes de production des caracteres par les generateurs associes, voir [Lu68-2].

Clavier alphanumerique (alpha(nu)meric keyboard): Clavier semblable a celui d'une machine a ecrire, permettant la production de chaines de caracteres alphabetiques, numeriques ou divers (ponctuation et signes mathematiques principalement). La transmission effective a l'ordinateur de cette chaine de caracteres se fait par interruptions, soit a chaque caractere frappe si l'entretien° de l'image est fait directement par l'ordinateur, soit sur la frappe d'une touche speciale de "fin de ligne" dans le cas contraire.

Clavier de fonctions (function keyboard, pushbuttons): Clavier de touches muettes dont chacune envoie a l'ordinateur une interruption° reconnaissable par un numero quand on l'enfonce; on peut associer n'importe quelle signification a ces interruptions. On ajoute souvent a chaque touche un indicateur lumineux permettant de

signaler à tout moment les touches dont l'utilisation est permise. Pour se souvenir de la signification de chaque touche, on écrit à côté ce que l'on désire, en particulier en utilisant un cache amovible qui porte lui-même un numéro lisible par l'ordinateur; ceci permet d'avoir plusieurs caches associés chacun à un jeu de possibilités, et de ne pas être limité par le nombre de touches du clavier, forcément réduit.

Communication: Les outils de communication sont ceux qui servent à l'envoi d'informations du terminal vers l'ordinateur, permettant ainsi l'utilisation conversationnelle de l'ensemble. Les principaux outils technologiques sont le pointeur optique°, les outils d'envoi de coordonnées, le clavier alphanumérique° et le clavier de fonctions°; l'outil programme est le niveau° d'interruptions.

Console de visualisation (display, CRT terminal): Dispositif périphérique connectable à un ordinateur et servant à la production d'information visuelle sur un écran cathodique; suivant ses possibilités d'affichage, c'est un terminal alphanumérique° ou un terminal graphique°.

Curseur: Symbole de forme quelconque placé sur l'écran du terminal pour indiquer la position où sera placé le prochain caractère frappé sur le clavier alphanumérique°. Il peut être matérialisé dans la mémoire d'entretien° par un indicateur lié au caractère correspondant; des touches spéciales du clavier alphanumérique permettent de le déplacer sans modifier les caractères existants.

Définition: Les outils de définition servent à donner le nom et les caractéristiques des variables graphiques à utiliser dans un programme, à un niveau élevé; cette notion n'a de sens qu'à l'étage des structures°, et recouvre les déclarations et les outils de génération° des étages inférieurs.

Déformation: Manipulation d'une figure° ou d'un objet° qui en conserve les propriétés topologiques mais non pas les propriétés géométriques; elle peut être obtenue par le déplacement° d'un objet de la figure ou d'un élément° de l'objet, ou par une modification globale telle qu'une affinité ou l'élévation au carré des coordonnées.

Déplacement: Manipulation d'une figure° ou d'un objet° qui en conserve les propriétés géometriques; elle peut être obtenue par une translation, une rotation, ou la composition des deux, et s'exprime facilement par un changement de système d'axes qui n'introduit pas

d'homothétie: $x = x' \cos a - y' \sin a + x^0$; $y = x' \sin a + y' \cos a + y^0$. Si la translation est réalisée de façon très simple par la plupart des terminaux, il n'en est pas de même de la rotation, qui nécessite le plus souvent une réfection complète de la liste d'affichage°.

Elément: Ce qui sert à composer tout ce que peut afficher le terminal (atome technologique); dans la plupart des cas, les éléments sont le point°, le vecteur° et le caractère°. On associe aux éléments le premier étage de la pyrmide des langages (voir 3.12).

Entretien de l'image (refreshing): Régénération continuelle de l'image par retour au début de la liste d'affichage dès qu'on a fini de la balayer ou de l'exécuter. L'entretien de l'image doit se faire à un rythme compris entre une limite inférieure au-dessous de laquelle l'image clignote, et une limite supérieure au-dessus de laquelle l'écran risque d'être détérioré; il est dévolu soit à l'ordinateur, soit au terminal lui-même si celui-ci dispose d'une mémoire destinée à cet usage.

Figure (graphic data set): Ensemble d'objets° disposant de certaines caractéristiques communes et manipulable comme un tout. Ces propriétés communes sont: le nom de la figure; le système de coordonnées; la portion d'écran affectée; les modes de coordonnées, des éléments générés, des caractères alphanumériques, etc. On associe aux figures le troisième étage de la pyramide des langages (voir 3.14).

Génération: Les outils de génération permettent la construction des objets° qui composent l'image° à afficher, par composition de la liste d'affichage°; au niveau des structures, les outils de génération se fondent avec les déclarations pour donner les outils de définition°.

Image: Ensemble de ce qui apparaît sur l'écran à un instant donné; l'image peut être formée de plusieurs figures°, ou d'une seule, ou être vide.

Interruption (attention): Arrêt technologique du déroulement d'un programme, sans que celui-ci y soit pour quelque chose; ceci permet de signaler un évènement qui s'est produit alors que le programmme était en train de fonctionner, donc de synchroniser deux processus asynchrones° d'une façon moins coûteuse que par la consultation périodique d'indicateurs.

Langage de commande (control language): Langage servant au dialogue avec le système°; il peut être ou non conversationnel suivant que le système l'est ou ne l'est pas.

Langage de communication (procedure-oriented language): Langage servant au dialogue avec le programme que l'on a fabriqué, et servant donc à indiquer ce que l'on veut faire mais non pas la façon de le faire; il n'est pratiquement intéressant que s'il est conversationnel.

Langage de programmation (programming language, problem-oriented language): Langage servant à construire des programmes, en indiquant comment doivent être effectuées les actions; il ne peut mormalement pas être conversationnel.

Ligne: Objet° fabriqué avec des éléments° de type vecteur°; l'origine d'une ligne est toujours implicite, c'est-a-dire qu'elle est donnée par le dernier élément généré: point°, caractère° ou extrémité de vecteur°. L'objet "ligne" est donc l'expression exacte de l'élément "vecteur".

Liste d'affichage (display file): Suite de données, ou d'ordres, ou des deux mélanges, dont le traitement par exécution ou consultation permet l'affichage d'une image°. S'il y a une mémoire d'entretien°, la liste d'affichage y est placée, sinon elle est dans la mémoire de l'ordinateur.

Manche à balai (joystick): Dispositif de communication°, forme d'un levier vertical que l'on peut incliner dans toutes les directions, et qui sert à fournir des informations de direction et d'inclinaison au moyen des composantes d'un vecteur. On utilise généralement ceci pour indiquer le déplacement° d'un objet° ou d'une figure° de l'image° affichée, l'inclinaison servant à représenter la vitesse de ce déplacement. A la difference de la boule roulante°, l'information peut exister alors que le manche a balai est immobile, aussi ajoute-t-on souvent un bouton poussoir au sommet du levier, sur lequel on appuie quand on veut envoyer l'information à l'ordinateur.

Mémoire d'entretien (refreshing memory, display buffer): Mémoire dans laquelle l'ordinateur place la liste d'affichage° et où les circuits de commande du terminal trouvent les ordres à exécutet et les données qui les concernent. Ceci permet l'entretien de l'image sans intervention de l'ordinateur.

Menu: Liste de noms de commandes, fonctions, options, etc., affichée sur l'écran; le pointage d'un de ces noms (à l'aide du pointeur optique°) équivaut fonctionnellement à l'envoi à l'ordinateur de ce mot. Le processus tout entier doit évidemment être programmé. Si le menu est long, on le divise en pages, et l'un des noms affichés permet de changer de page.

Niveau (d'interruptions) (attention level): Ensemble de files d'attente, à raison d'une par source d'interruptions°, où l'on place les informations concernant les interruptions qui se produisent; des operations spécialisées permettent principalement l'ouverture et la fermeture des portes d'entrée de ces files d'attente, et la consultation des informations qui se trouvent à la sortie. C'est le principal outil programmé de communication°, il a pour effet important de synchroniser des traitements naturellement asynchrones°.

Objet: Ce qui sert à composer tout ce que l'on peut afficher au moyen du langage de programmation° (atome programmé); les objets sont fabriqués avec des éléments°, et un objet peut être formé d'une suite de composants de même type. Les objets les plus courants sont le point°, la ligne°, le segment°, la position° et le texte°. On associe aux objets le deuxième étage de la pyramide des langages (voir 3.13).

Point: Elément fondamental de tout terminal graphique°, formé par l'impact sur l'écran du faisceau d'électrons. L'objet° "point" est formé d'un ou plusieurs éléments° "points".

Pointeur optique (light pen): Dispositif de communication° formé d'une cellule photo-électrique occultable qui envoie une interruption° à l'ordinateur quand elle détecte de la lumière. On l'utilise comme moyen de désignation d'un objet° affiché sur l'écran, car l'interruption arrête l'entretien° de l'image, ce qui permet de connaître l'adresse du dernier ordre exécuté ainsi que les valeurs des registres de coordonnées. La cellule est souvent placée non pas dans le pointeur lui-même, mais au bout d'un conducteur optique en fibre de verre. Les autres termes proposés en français pour désigner ce dispositif sont par exemple crayon lumineux, crayon ou pointeur photosensible, crayon optique, photostyle, photocapteur, piège à photons, etc.; le terme proposé ici me semble être le seul qui indique sans pédanterie et sans contre-sens la fonction et le fonctionnement de ce dispositif.

Position: Objet° formé d'un élément° "point" éteint, qui permet de placer le faisceau lumineux en un endroit précis, avant l'affi-

chage d'une ligne° ou d'un texte°, ou le passage à des coordonnees relatives.

(Caractères) Protégés: Des caractères° affichés dans le mode protégé ne peuvent pas être modifiés au moyen du clavier alphanumérique°, même si le curseur° vient se placer dessus.

Segment: Objet fabrique avec des éléments de type vecteur° et point°: on indique pour chaque segment les coordonnées de ses deux extrémités, ce qui fait qu'un objet "segment" multiple est formé de vecteurs non contigus.

Séquence: Regroupement d'objets° d'une même figure° générés consécutivement, et auquel on donne un nom par lequel on le désigne pour lui faire subir en bloc certaines manipulations simples.

Souris (mouse): Dispositif de communication° formé d'un objet (ressemblant vaguement à une souris) que l'on peut déplacer sur une surface plane quelconque (par exemple une table ou l'écran lui-même); une sphère affleure sous la partie de l'objet qui repose sur la surface plane, et les mouvements qu'on lui communique produisent finalement la même information qu'avec la boule roulante°. La seule différence avec ce dernier dispositif est qu'on relie mentalement plus facilement les mouvements que l'on communique à la souris aux déplacements° des objets° ou figures° sur l'écran qu'avec la boule roulante, qui est fixe et que l'on ne peut pas manipuler de facon continue.

Structure: Outil d'expression des propriétés géométriques, topologiques, logiques et autres des figures° et des objets° qui les composent. On associe aux structures le quatrième étage de la pyramide des langages (voir 3.15).

Systeme: Emsemble constitué d'une part de l'ordinateur, des dispositifs périphériques et des organes de commande et de liaison qui les relient (partie technologique ou "hardware"), d'autre part des nombreux programmes et sous-programmes qui forment le superviseur (partie programmée ou "software"). Cet ensemble a pour but de permettre la fabrication et le fonctionnement de programmes conçus par les utilisateurs. Un programme écrit par un utilisateur et mis à la disposition des autres peut venir augmenter le volume et les possibilités du système, de même que l'adjonction d'un nouveau dispositif périphérique.

Tablette Rand: Dispositif de communication° permettant l'envoi de coordonnées à l'ordinateur, et formé d'une tablette carrée où

sont entrecroisées deux nappes perpendiculaires de fils électriques, de telle façon qu'en posant la pointe d'un stylet métallique en un point du carré on envoie du courant électrique dans les deux fils qui se coupent en cet endroit. Cet outil ne produit pas normalement d'interruption, il constitue donc en fait un dispositif que l'on doit "lire" pour connaître la position actuelle du stylet. La différence avec la tablette Sylvania° est que la tablette Rand° est opaque et doit être placée à côté de l'écran, généralement à plat devant lui; le fait d'agir quelque part et de regarder ailleurs l'effet de cette action n'a rien de difficile, comme le montre la conduite d'une automobile.

Tablette Sylvania: Dispositif de communication° analogue à la tablette Rand°, mais transparent, ce qui permet de le placer devant l'écran et de pointer le stylet directement sur l'image° affichée.

Terminal alphanumérique: Console de visualisation° capable uniquement d'afficher des caractères°, généralement en des positions fixes de l'écran. Le mode de balayage ressemble d'ordinaire à celui d'un récepteur de télévision. Certaines commandes spéciales du clavier alphanumérique° fournissent des possibilités de modification du texte affiché avant son envoi à l'ordinateur; c'est à peu près la seule différence fonctionnelle d'un terminal alphanumérique avec une machine à écrire, à part son silence et sa rapidité.

Terminal graphique: Console de visualisation° capable d'afficher au moins des points°, le plus souvent des vecteurs° et des caractères°, parfois des arcs de cercles ou d'ellipses. Le balayage est le plus souvent cavalier, ce qui interdit l'affichage de surfaces pleines et les demi-teintes; certains terminaux graphiques produisent des dessins de plusieurs couleurs, quoi qu'on n'en ait encore guère vu l'utilité.

Texte: Objet formé avec des éléments° de type caractère°; si l'on associe des coordonnées de départ à un texte, il faut en plus un élément de type point° (éteint) pour le construire.

Transmission: Les outils de transmission permettent de commander le terminal par l'envoi d'informations: rangement de la liste d'affichage° dans la mémoire d'entretien°, lancement ou arrêt de la régénération, modification directe de la liste d'affichage; ils permettent en outre à l'ordinateur de "lire" certaines informations, par exemple le contenu de la mémoire d'entretien ou celui des registres du terminal. Au niveau des structures°, on préfère s'abstraire de la façon dont est fait le travail, et parler d'outils d'utilisation°.

Utilisation: Les outils d'utilisation permettent d'afficher, éteindre, rallumer et modifier (pour des déformations° ou des déplacements°) les figures° et les objets° qui composent l'image°. On les emploie au niveau des structures°, préférant aux niveaux inférieurs voir en détail les problèmes de transmission° qu'ils recouvrent.

Vecteur: Elément fondamental existant sur la plupart des terminaux, et seul composant ou presque de la grande majorité des dessins que l'on fabrique. L'origine d'un vecteur est toujours implicite, c'est l'endroit ou vient d'arriver le faisceau lumineux; on ne représente donc un vecteur que par les coordonnées de son extrémité (mode absolu) ou par ses composantes en x et y (mode relatif).

ANNEXE 3 - DESCRIPTION SYNTAXIQUE DU LANGAGE EUPHEMIE.

```
******** FONCTIONS
*
*       OPT EST UN MODELE QUI COINCIDE AVEC SON PARAMETRE OU RIEN
*
        DEFINE('OPT(MODELE)')     :(E1)
OPT     OPT = NUL | MODELE  :(RETURN)
*
*       REP EST UN MODELE QUI COINCIDE AVEC UN NOMBRE INDEFINI DE FOIS
*       SON PARAMETRE
*
E1      DEFINE('REP(MODELE)')     :(E2)
REP     REP = MODELE OPT(*REP(MODELE))      :(RETURN)
*
E2
******** DECLARATIONS DE VARIABLES, TABLEAUX ET CHAINES
*
*       IDENTIFICATEURS ET CONSTANTES
*
        LETTRES = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        LETTRE = ANY(LETTRES)
        CHIFFRES = "0123456789"
        CHIFFRE = ANY(CHIFFRES)
        IDENTIFICATEUR = LETTRE ARBNO(LETTRE | CHIFFRE)
        ENTIER_SANS_SIGNE = REP(CHIFFRE)
        SIGNE = "+" | "-"
        ENTIER = OPT(SIGNE) ENTIER_SANS_SIGNE
        NOMBRE_DECIMAL = ENTIER_SANS_SIGNE |
+               OPT(ENTIER_SANS_SIGNE) "." ENTIER_SANS_SIGNE
        NOMBRE_SANS_SIGNE = NOMBRE_DECIMAL |
+               OPT(NOMBRE_DECIMAL) "    " ENTIER
        NOMBRE = OPT(SIGNE) NOMBRE_SANS_SIGNE
        ALPHABET = &ALPHABET
        ALPHABET '"' =
        CHAINE = '"' SPAN(ALPHABET) '"'
*
*       DECLARATIONS DE VARIABLES
*
        B = SPAN(" ")
*       B COINCIDE AVEC UN OU PLUSIEURS BLANCS
```

```
        E = OPT(B)
*       E COINCIDE AVEC ZERO, UN OU PLUSIEURS BLANCS
        DECL_REELS = "'REEL'" E IDENTIFICATEUR
+                   OPT(E ":=" E NOMBRE)
                    ARBNO(E "," E IDENTIFICATEUR OPT(E ":=" E NOMBRE))
+                   E ";"
        DECL_ENTIERS = "'ENTIER'" E IDENTIFICATEUR
+                   OPT(E ":=" E ENTIER)
+                   ARBNO(E "," E IDENTIFICATEUR OPT(E ":=" E ENTIER))
+                   E ";"
*
*       DECLARATION DE TABLEAUX
*
        ELEM_TABLEAU = ("'REEL'" | "'ENTIER'") E IDENTIFICATEUR
+                   ARBNO(E "," E IDENTIFICATEUR)
+                   E "(" E ENTIER_SANS_SIGNE E ")"
        DECL_TABLEAUX = "'TABLEAU'" E ELEM_TABLEAU
+                   ARBNO(E "," E ELEM_TABLEAU)
+                   E ";"
*
*       DECLARATION DE CHAINES
*
        ELEM_CHAINE = ENTIER_SANS_SIGNE E "'CARACTERES'" E
+                   IDENTIFICATEUR OPT(E ":=" E CHAINE)
+                   ARBNO(  E "," E IDENTIFICATEUR OPT(E ":=" E CHAINE))
        DECL_CHAINES = "'CHAINE'" E ELEM_CHAINE
+             ARBNO(E "," E ELEM_CHAINE)
+                   E ";"
*
******** EXPRESSIONS
*
*       EXPRESSION ARITHMETIQUE
*
        VARIABLE = IDENTIFICATEUR OPT(E "(" E
+                   *EXPRESSION_ARITHMETIQUE E ")")
        OPADD = ANY("+-")
        OPMUL = ANY("*/")
        PRIMAIRE = NOMBRE_SANS_SIGNE | *VARIABLE |
+                   "(" E *EXPRESSION_ARITHMETIQUE E ")"
        FACTEUR = *PRIMAIRE | *FACTEUR E "**" E *PRIMAIRE
        TERME = *FACTEUR | *TERME E OPMUL E *FACTEUR
        EXPRESSION_ARITHMETIQUE = *TERME | OPADD E *TERME |
+                   *EXPRESSION_ARITHMETIQUE E OPADD E *TERME
*
*       EXPRESSION DE TEXTE
*
```

```
          ELEM_TEXTE = CHAINE | IDENTIFICATEUR
          SELECTION = "'SELECTION'" E ELEM_TEXTE OPT(E "'DEPUIS'" E
+                   *EXPRESSION_ARITHMETIQUE) OPT(E "'LONGUEUR'" E
+                   *EXPRESSION_ARITHMETIQUE)
          EXPRESSION_TEXTE = (ELEM_TEXTE | E *SELECTION)
+                   OPT(E "||" E *EXPRESSION_TEXTE)

*
*         EXPRESSION BOOLEENNE
*
          OPREL = "<" | "<=" | "=" | ">=" | ">" | "¬="
          EXPRESSION_BOOLEENNE = (*EXPRESSION_ARITHMETIQUE E OPREL E
+                   *EXPRESSION_ARITHMETIQUE) |
+                   (*EXPRESSION_TEXTE E ("=" | "¬=") E
+                   *EXPRESSION_TEXTE)
*
******** INSTRUCTIONS NON GRAPHIQUES
*
*         AFFECTATIONS
*
          AFFECT_ARITH = *VARIABLE E ":=" E *EXPRESSION_ARITHMETIQUE
          AFFECT_TEXTE = (IDENTIFICATEUR | *SELECTION) E ":=" E
+                   *EXPRESSION_TEXTE
*
*         INSTRUCTIONS DIVERSES
*
          ALLER_A = "'ALLERA'" E IDENTIFICATEUR
          POUR = "'POUR'" E *VARIABLE E ":=" E *EXPRESSION_ARITHMETIQUE
+                   OPT(E "'PAS'" E *EXPRESSION_ARITHMETIQUE)
+                   E "'A'" E *EXPRESSION_ARITHMETIQUE
          SI = "'SI'" E *EXPRESSION_BOOLEENNE
*
*         TOUTES LES INSTRUCTIONS NON GRAPHIQUES
*
          INST_NON_GRAPHIQUE = *AFFECT_ARITH | *AFFECT_TEXTE |
+                   ALLER_A | *POUR | *SI | "'FINPOUR'" | "'SINON'" |
+                   "'FINSI'"
          INSTRUCTION = OPT(IDENTIFICATEUR E ":")
+                   (*INST_NON_GRAPHIQUE | *INST_GRAPHIQUE) E ";"
*
******** DECLARATIONS GRAPHIQUES
*
*         NOMS
*
          NOM_NIVEAU = IDENTIFICATEUR
          NOM_FIGURE = IDENTIFICATEUR
```

```
          NOM_LIGNE = IDENTIFICATEUR
          NOM_POINT = IDENTIFICATEUR
          NOM_SEGMENT = IDENTIFICATEUR
          NOM_TEXTE = IDENTIFICATEUR
          NOM_POSITION = IDENTIFICATEUR
          NOM_SEQUENCE = IDENTIFICATEUR
          NOM_ARRET = IDENTIFICATEUR
          NOM_OBJET = NOM_LIGNE | NOM_POINT | NOM_SEGMENT | NOM_TEXTE |
     +              NOM_POSITION
     *
     *    DECLARATION DE NIVEAU
     *
          DECL_NIVEAU = "'NIVEAU'" E NOM_NIVEAU
     +              ARBNO(E "," E NOM_NIVEAU)
     *
     *    DECLARATION DE FIGURE
     *
          BORNE = *VARIABLE | NOMBRE
          DECL_FIGURE = "'FIGURE'" E NOM_FIGURE
     +              OPT(E "'ENTREE'" E ("'ABSREEL'" | "'ABSENT'" |
     +              "'RELREEL'" | "'RELENT'") OPT(E "," E ("'ABSREEL'"
     +              | "'ABSENT'" | "'RELREEL'" | "'RELENT'")))
     +              OPT(E "'COUPAGE'" E ("'EC'" | "'EA'" | "'FC'" |
     +              "'FA'" | "'LIBRE'"))
     +              OPT(E "'SORTIE'" E ("'OPTIMISE'" | "'ABSOLU'" |
     +              "'INCREMENT'"))
     +              OPT(E "'CARACTERES'" E ("'PETITS'" | "'GRANDS'")
     +              OPT(E "'PROTEGES'"))
     +              OPT(E "'ZONE'" E *BORNE E "," E *BORNE E "," E
     +              *BORNE E "," E *BORNE OPT(E "'ECRAN'" E *BORNE E ","
     +              E *BORNE E "," E *BORNE E "," E *BORNE))
     +              OPT(E "'DONNEES'" E *BORNE E "," E *BORNE E "," E
     +              *BORNE E "," E *BORNE)
     +              OPT(E "'TAILLE'" E ENTIER_SANS_SIGNE)
     +              OPT(E "'RENOMME'" E NOM_FIGURE OPT(REP(E "," E
     +              NOM_FIGURE)))
     *
     *    DECLARATIONS D'OBJETS
     *
          DECL_OBJETS = ("'LIGNE'" | "'POINT'" | "'SEGMENT'" | "'TEXTE'"
     +              | "'POSITION'" | "'SEQUENCE'" | "'ARRET'") E
     +              IDENTIFICATEUR
     +              ARBNO(E "," E IDENTIFICATEUR)
     +              OPT(E "'FIGURE'" E NOM_FIGURE)
     *
     ******** INSTRUCTIONS GRAPHIQUES
```

```
*
*       INSTRUCTIONS D'INITIALISATION
*
        DEBUT_FIN = ("'DEBUT'" | "'FIN'") E
+                   ("'FIGURE'" E NOM_FIGURE OPT(REP( E "," E
+                   NOM_FIGURE)))
+                   | ("'SEQUENCE'" E NOM_SEQUENCE)
+                   | ("'NIVEAU'" E NOM_NIVEAU OPT(E ("'LIBRE'" |
+                   "'GARDE'")))
        CRAYON = "'CRAYON'" E NOM_FIGURE E ("'TRAITE'" | "'IGNORE'")
        OPTIONS = "'OPTIONS'" E NOM_FIGURE
+                   OPT(E "'ENTREE'" E ("'ABSREEL'" | "'ABSENT'" |
+                   "'RELREEL'" | "'RELENT'") OPT(E "," E ("'ABSREEL'"
+                   | "'ABSENT'" | "'RELREEL'" | "'RELENT'")))
+                   OPT(E "'COUPAGE'" E ("'EC'" | "'EA'" | "'FC'" |
+                   "'FA'" | "'LIBRE'"))
+                   OPT(E "'SORTIE'" E ("'OPTIMISE'" | "'ABSOLU'" |
+                   "'INCREMENT'"))
+                   OPT(E "'CARACTERES'" E ("'PETITS'" | "'GRANDS'")
+                   OPT(E "'PROTEGES'"))
+                   OPT(E "'ZONE'" E *BORNE E "," E *BORNE E "," E
+                   *BORNE E "," E *BORNE OPT(E "'ECRAN'" E *BORNE E ","
+                   E *BORNE E "," E *BORNE E "," E *BORNE))
+                   OPT(E "'DONNEES'" E *BORNE E "," E *BORNE E "," E
+                   *BORNE E "," E *BORNE)
*
*       TRAITEMENT DES INTERRUPTIONS
*
        TRAITER_IGNORER = ("'TRAITER'" | "'IGNORER'") E NOM_NIVEAU
+                   E "(" E *EXPRESSION_ARITHMETIQUE
+                     ARBNO(E "," E *EXPRESSION_ARITHMETIQUE)  E ")"
        TOUCHES = "'TOUCHES'" E OPT(NOM_NIVEAU) E
+                   ("'DEFAUT'" | "'ETEINT'" | "'NORMAL'" |
+                   ("(" E *EXPRESSION_ARITHMETIQUE OPT(REP(E "," E
+                   *EXPRESSION_ARITHMETIQUE)) E ")"))
        MODIFIER = "'MODIFIER'" E NOM_NIVEAU  E "'DE'" E
+                   *EXPRESSION_ARITHMETIQUE OPT(E ("'ENHAUT'" |
+                   "'ENBAS'"))
        ETAT = "'ETAT'" E NOM_NIVEAU E "(" E *EXPRESSION_ARITHMETIQUE
+                     ARBNO(E "," E *EXPRESSION_ARITHMETIQUE)  E ")"
+                   OPT(E "'REPONSE'" E *VARIABLE)
+                   OPT(E "'TABLEAU'" E NOM_TABLEAU)
+                   OPT(E ("'ATTENTE'" | "'RETOUR'"))
*
*       GENERATION DES OBJETS
*
```

```
EA = *EXPRESSION_ARITHMETIQUE
MODE = ANY("=?")
SYMB_GEN = "<" MODE MODE
SYMB_MAJ = "<" MODE MODE ">"
DES_NUMERO = NOM_FIGURE E "'NUMERO'" E *EA
GEN_P_L_P = (NOM_POINT ! NOM_LIGNE ! NOM_POSITION !
+            ("'POINT'" ! "'LIGNE'" ! "'POSITION'") E NOM_FIGURE) E
+            SYMB_GEN E *EA E "," E *EA
+            OPT(E "'NUMERO'" E *EA)
+            OPT(E ("'INCLUS'" ! "'OMIS'"))
GEN_SEG = (NOM_SEGMENT ! ("'SEGMENT'" E NOM_FIGURE)) E SYMB_GEN
+            E *EA E "," E *EA E "," E *EA E "," E *EA
+            OPT(E "'NUMERO'" E *EA)
+            OPT(E ("'INCLUS'" ! "'OMIS'"))
GEN_TEX = (NOM_TEXTE ! ("'TEXTE'" E NOM_FIGURE)) E SYMB_GEN E
+            EXPRESSION_TEXTE OPT(E "'EN'" E *EA E "," E *EA)
+            OPT(E "'NUMERO'" E *EA)
+            OPT(E ("'INCLUS'" ! "'OMIS'"))
GENMUL_P_L = (NOM_POINT ! NOM_LIGNE ! ("'POINT'" ! "'LIGNE'") E
+            NOM_FIGURE) E SYMB_GEN E *EA E "'FOIS'" E *COOR E ","
+            E *COOR OPT(E "'NUMERO'" E *EA)
+            OPT(E ("'INCLUS'" ! "'OMIS'"))
GENMUL_SEG = (NOM_SEGMENT ! ("'SEGMENT'" E NOM_FIGURE)) E
+            SYMB_GEN E *EA E "'FOIS'" E *COOR E "," E *COOR E ","
+            E *COOR E "," E *COOR
+            OPT(E ("'INCLUS'" ! "'OMIS'"))
+            OPT(E "'NUMERO'" *EA)
*
*       MISE A JOUR DES OBJETS
*
MAJ_P_L_P = (NOM_POINT ! NOM_LIGNE ! NOM_POSITION ! ("'POINT'"
+            ! "'LIGNE'" ! "'POSITION'") E NOM_FIGURE) E SYMB_MAJ
+            E *EA E "," E *EA
+            OPT(E "'NUMERO'" *EA)
MAJ_SEG = (NOM_SEGMENT ! "'SEGMENT'" E NOM_FIGURE)
+            E SYMB_MAJ E *EA E "," E *EA E "," E *EA E "," E *EA
+            OPT(E "'NUMERO'" *EA)
MAJ_TEX = (NOM_TEXTE ! "'TEXTE'" E NOM_FIGURE) E SYMB_MAJ E
+            *EXPRESSION_TEXTE OPT(E "'EN'" E *EA E "," E *EA)
+            OPT(E "'NUMERO'" *EA)
MAJMUL_P_L = (NOM_POINT ! NOM_LIGNE ! ("'POINT'" ! "'LIGNE'")
+            E NOM_FIGURE) E SYMB_MAJ E *EA E "'FOIS'" E *COOR E
+            "," E *COOR
+            OPT(E "'NUMERO'" *EA)
MAJMUL_SEG = (NOM_SEGMENT ! "'SEGMENT'" E NOM_FIGURE) E
+            SYMB_MAJ E *EA E "'FOIS'" E *COOR E "," E *COOR E ","
```

```
+                E *COOR E "," E *COOR
+                OPT(E "'NUMERO'" *EA)
*
*       INSTRUCTIONS "PLACER" ET "situer"
*
        PLACER = "'PLACER'" E NOM_FIGURE E "'EN'" E *EA E "," E *EA
        situer = "'REPRISE'" E NOM_FIGURE E "'EN'" E *EA E "," E *EA
*
*       TRAITEMENT DES FIGURES
*
        METTRE_ARRET = "'METTRE'" E "'ARRET'" E (NOM_ARRET |
+               DES_NUMERO) OPT(E ("'INCLUS'" | "'OMIS'"))
        INCLURE_OMETTRE = ("'INCLURE'" | "'OMETTRE'") E (NOM_FIGURE |
+               NOM_OBJET | NOM_SEQUENCE | NOM_ARRET | DES_NUMERO)
        AFFICHER = "'AFFICHER'" E NOM_FIGURE
+               ARBNO(E "," E NOM_FIGURE)
        ORDONNER = "'ORDONNER'" E NOM_FIGURE
+               ARBNO(E "," E NOM_FIGURE)
        ANNULER = "'ANNULER'" E (NOM_FIGURE | NOM_OBJET | NOM_SEQUENCE
+               | NOM_ARRET | DES_NUMERO)
*
*       INSTRUCTIONS DIVERSES
*
        METTRE_CURSEUR = "'METTRE'" E "'CURSEUR'" E (NOM_TEXTE |
+               NOM_SEQUENCE | DES_NUMERO) OPT(E "'EN'" E *EA)
        ENLEVER_CURSEUR = "'ENLEVER'" E "'CURSEUR'" E NOM_FIGURE
        LIRE = "'LIRE'" E *EA E ("'DONNEES'" | "'CARACTERES'") E
+               "'DANS'" E NOM_TABLEAU OPT(E "'JUSQUA'" E ("'FIN'" |
+               "'CURSEUR'"))
+               OPT(E "'DEPUIS'" E (NOM_OBJET | NOM_ARRET | *EA))
+               OPT(E "'A'" E (NOM_OBJET | NOM_ARRET | *EA))
+               OPT(E "'FIGURE'" E NOM_FIGURE)
+               OPT(E "'REPONSE'" E *VARIABLE)
        LIRECRAYON = "'LIRECRAYON'" E NOM_FIGURE E "'DANS'" E
+               *VARIABLE E "," E *VARIABLE
        ALARME = "'ALARME'"
        FINGRAPHIQUE = "'FINGRAPHIQUE'"
*
*       TOUTES LES INSTRUCTIONS GRAPHIQUES
*
        INST_GRAPHIQUE = DEBUT_FIN | CRAYON | OPTIONS | TRAITER_IGNORE
+               | TOUCHES | MODIFIER | ETAT | GEN_P_L_P | GEN_SEG |
+               GEN_TEX | GENMUL_P_L | GENMUL_SEG | MAJ_P_L_P |
+               MAJ_SEG | MAJMUL_P_L | MAJMUL_SEG | PLACER | situer
+               METTRE_ARRET | INCLURE_OMETTRE | AFFICHER | ORDONNER
+               ANNULER | METTRE_CURSEUR | ENLEVER_CURSEUR | LIRE |
```

```
*                    LIRECRAYON ! ALARME ! FINGRAPHIQUE
          &FULLSCAN = 1
LIREI     LIGNE = TRIM(INPUT)          :F(END)
          OUTPUT = '      ' LIGNE
LIREI1    LIGNE ';'           :S(TRAITER)
          CARTE = TRIM(INPUT)      :F(END)
          OUTPUT = '      ' CARTE
          CARTE '     ' =
          LIGNE = LIGNE CARTE          :(LIREI1)
TRAITER   &ANCHOR = 1
          LIGNE INSTRUCTION =                :F(ERREURI)
          &ANCHOR = 0
          DIFFER(LIGNE)           :F(LIREI)
ERREURI OUTPUT = '********** E R R E U R **********'
          &ANCHOR = 0
          OUTPUT =          :(LIREI)
END
```

```
commentaire déclarations graphiques;
figure BARATINS entrée entabs écran 1, 1, 74, 52 zône 3, 2, 72,
    48 données 1, 48, 70, 1;
texte QUESTION, CONFIRME, INFIRME figure BARATINS;
figure TABLEAUX entrée entabs écran 1, 1, 74, 52 zône 3, 1, 72,
    48 données 1, 1, 70, 48;
figure JINFORME entrée entabs écran 1, 1, 74, 52 zône 1, 1, 74,
    4 données 1, 4, 74, 1;
texte  ONAJOUTE,  ONENLEVE,  ONCORRIJ,  DESDECLA,  DESINSTR,
    AUPROGRA, QUEJINDI, MESSERRE figure JINFORME;
figure REPONSES entrée entabs caractères petits écran 1, 1, 74,
    52 zône 3, 1, 72, 48 données 1, 48, 70, 1;
texte repONSE figure REPONSES;
entier TDEBUT := 1, TFIN := 2, TEXECUTE  := 3,  TTABLE  := 6,
    TNOMMER  := 7, TCOPIER := 8, TSUPPRIM := 9, TRECULER := 11,
    TAVANCER := 12, TDECLARA := 13,TINSTRUC := 14, FINLIGNE  :=
    32;
niveau NIVDEFIN, NIVDEBUT, NIVNOMME, NIVTABLE;
commentaire déclarations arithmetiques et de chaînes;
entier  TOTO,  MONCACHE  :=  0,  MOIMEME,  LAPERSON,  DECLIN,
    SESPROGR, SONPROGR, ADRPROGR, SESDECLA, SESINSTR, SESIDENT,
    SESVALEU, CREEDATE, TUTU, INDIC, TRUC, NUMERO, I, J;
tableau entier INFOETAT (10);
chaîne 32 caractères BLANCS  :=  "  ",  TAMPON,  20  caractères
    NOMINDIV,  MACHIN,  NOMHABIL  :=  "EULALIE",  16 caractères
    MOTDEPAS;
commentaire déclaration des procédures;
procédure ALLERSI entier EN étiquette;
si TOTO = ALLERSI;
allerà EN finisi;
finprocédure;
procédure PQUESTIO chaîne  EN  entier  LONGUEUR  entier  NIVEAU
    niveau FIN étiquette RESULTAT chaîne;
QUESTION <== PQUESTIO en 1, EN;
REPONSE  <==  sélection  BLANCS  longueur  LONGUEUR en longueur
    PQUESTIO + 1;
afficher BARATINS;
afficher REPONSES;
mettre curseur REPONSE;
état NIVEAU (TFIN, FINLIGNE) réponse TOTO;
```

```
enlever curseur REPONSES;
allersi TFIN en FIN;
lire LONGUEUR caractères de REPONSE dans RESULTAT;
finprocédure;
procédure PREPONSE chaîne EN entier NIVEAU niveau FIN étiquette
    SUITE emtier ALARME chaîne := "OUI";
INFIRME <== PREPONSE en 1, EN;
afficher BARATINS;
si ALARME = "OUI";
alarme;
finsi;
état NIVEAU (TFIN, SUITE) réponse TOTO:
annuler BARATINS;
allersi TFIN en FIN;
finprocédure;
commentaire fin des déclarations;


commentaire début du programme;
commentaire niveau DEBUT-FIN - commandes "début" et "fin";
DEFIN :  touches normal;
début niveau NIVDEFIN;
traiter NIVDEFIN (TDEBUT, TFIN, finLIGNE);
début figure BARATINS, REPONSES;
SUIDEFIN :  état NIVDEFIN (TDEbut, TFIN) réponse TOTO  résultat
    INFOETAT;
allersi TFIN en FINDEFIN;
si INFOETAT (1) ¬= MONCACHE;
réponse  "VEUILLEZ  PLACER  LE  BON  CACHE"  en 18 fin RENOMPRO
    niveau NIVDEFIN suite TDEBUT;
finsi;
RENOMPRO :  question "DONNEZ VOTRE NOM" en 18 longueur  20  fin
    FINDEFIN résultat NOMINDIV niveau NIVDEFIN;
MOIMEME := 1;
si NOMHABIL ¬= NOMINDIV;
cherelem LISINDIV avec NOMINDIV = individu réponse TOTO;
si TOTO = 0;
CONFIRME   <==  "COMMANDE  DEBUT  POUR  RECOMMENCER,  FIN  POUR
    TERMINER" en 1, 30;
réponse "JE NE VOUS CONNAIS PAS"  en  28  fin  FINDEFIN  niveau
    NIVDEbut suite TDEBUT;
annuler REPONSES;
allerà RENOMPRO;
finsi;
LAPERSON := numéro;
MOIMEME := 0;
```

```
finsi;
QUESTION <== "BBBBBBBBBBBBBBB" en 28, 23;
QUESTION <== "MMMMMMMMMMMMMMM" en 28, 23;
QUESTION <== "WWWWWWWWWWWWWWW" en 28, 23;
question "DONNEZ VOTRE MOT DE PASSE" en 23 longueur 16 niveau
    NIVDEFIN fin FINDEFIN résultat MOTDEPAS;
si MOTDEPAS ¬= motpasse;
réponse "MOT DE PASSE INCORRECT" en 33 niveau NIVDEFIN fin
    FINDEFIN suite TDEBUT; annuler REPOINSES;
allerà RENOMPRO finsi;
SESPROGR := linompro;
annuler BARATINS;
annuler REPONSES;
commentaire niveau DEBUT - commandes "TABLE", "NOMMER" et
    "FIN";
DEBUT : début niveau NIVDEBUT;
traiter NIVDEBUT (TTABLE, TNOMMER, TFIN, FINLIGNE);
SUIDEBUT : état NIVDEBUT (TTABLE, TNOMMER, TFIN) réponse TOTO;
annuler BARATINS;
annuler REPONSES;
allersi TTABLE en TABLE;
allersi TNOMMER en NOMMER;
commentaire fin du niveau DEBUT;
FINDEBUT : si MOIMEME = 1;
question "NOUVEAU NOM" en 18 longueur 20 niveau NIVDEBUT fin
    CHANMOPA résultat NOMHABIL;
CHANMOPA : question "NOUVEAU MOT DE PASSE" en 23 longueur 16
    niveau NIVDEBUT fin PAChanGE résultat MOTHABIL;
PACHANGE : annuler BARATINS;
annuler REPONSES;
sinon;
lirelem LAPERSON;
linompro := SESPROGR;
finsi;
fin niveau NIVDEBUT;
allerà SUIDEBUT;
commentaire niveau NOMMER;
NOMMER : début niveau NIVNOMME;
si MOIMEME = 1;
allerà PASDEPRO finsi;
traiter NIVNOMME (TNOMMER, TFIN, TFINLIGNE);
question "NOMMEZ VOTRE prograMME" en 18 longueur 10 niveau
    NIVNOMME fin FINNOMME résultat SONPROGR;
cherelem SESPROGR avec SONPROGR = nomprog réponse TOTO;
si TOTO = 0;
CONFIRME <== "CE PROGRAMME N'EXISTE PAS ENCORE" en 1, 23;
```

```
NOMBELEM := 0;
ADRPROGR := 0;
SESDECLA := 0;
SESINSTR := 0;
SESVALEU := 0;
CREEDATE := date;
pour J := 1 à 8;
INFOETAT (I) := prendele;
finpour;
SESIDENT := prendele;
adrsuite := INFOETAT;
J := prendele;
délares := SESDECLA;
K := prendele;
nomprog := SONPROGR;
datecréé := CREEDATE;
programe := J;
metrelem SESPROGR, K;
traiter NIVNOMME (TCOPIER);
sinon;
ADRPROGR := numéro;
NOMBELEM := encombre;
CREEDATE := datecréé;
lirelem programe;
SESDECLA := déclares;
CONFIRME  <==  "CE  PROGRAMME A ETE CREE LE " || datecréé || ",
    MODIFIE LE " || datemodi en 1, 23;
CONFIRME <== "IL OCCUPE " || encombre || " ELEMENTS" en 1, 25;
traiter NIVNOMME (TSUPPRIME, TEXECUTE);
finsi;
afficher BARATINS;
début figure JINFORME;
ONAJOUTE <== "ON AJOUTE DES " en 1, 4 omis;
ONENLEVE <== "ON ENLEVE DES " en 1, 4 omis;
ONCORRIJ <== "ON CORRIGE DES " en 1, 4 omis;
DESCDECLA <== "DECLARATIONS."  omis;
DESINSTR <== "INSTRUCTIONS."  omis en 16, 4;
AUPROGRA <== " - PROGRAMME ";
QUEJINDI <== SONPROGR;
afficher JINFORME;
traiter NIVNOMME (TDECLARA, TINSTRUC);
ignorer NIVNOMME (TNOMMER);
SUINOMME  :  etat  NIVNOMME  (TDECLARA,  TINSTRUC,  TCOPIER,
    TSUPPRIM, TEXECUTE, TFIN) réponse TOTO;
annuler BARATINS;
annuler REPONSES;
```

```
allersi TDECLARA en DECLARAT;
allersi TINSTRUC en INSTRUCT;
allersi TCOPIER en COPIER;
allersi TSUPPRIM en SUPPRIME;
allersi TEXECUTE en EXECUTER;
FINNOMME :  fin niveau NIVNOMME;
fin figure JINFORME;
lirelem ADRPROGR;
encombre := NOMBELEM;
datecréé := CREEDATE;
datemodi := date;
lirelem ADRPROGR;
déclares := SESDECLA;
allerà SUIDEBUT;
DECLARAT:  inclure DESDECLA;
DECLIN := 0;
appel DECLINST;
omettre DESDECLA;
allerà SUINOMME;
INSTRUCT :  inclure DESINSTR;
DECLIN := 1;
appel DECLINST;
omettre DESINSTR;
allerà SUINOMME;
PASDEPRO:  traiter NIVNOMME (TFIN, FINLIGNE);
question  "NOM  DE  L'UTILISATEUR"  en  18  longueur 20 réponse
    NOMINDIV fin FINPROMA niveau NIVNOMME;
cherelem LISINDIV avec individu = NOMINDIV réponse TOTO;
si TOTO ¬= 0;
allerà PANOUVO finsi;
CONfirME <== "PROGRAMMEUR NOUVEAU" en 1, 20;
afficher BARATINS;
question "mot DE PASSE" en 22 longueur 16 réponse MOTDEPAS  fin
    FINPROMA niveau NIVNOMME;
I := prendele;
individu := NOMINDIV;
metrelem I en LISINDIV;
allerà FINPROMA;
PANOUVO:  I := numero;
ADRPROGR := linompro;
traiter nivNOMME (TSUPPRIM);
reponse  "UTILISATEUR  EXISTANT"  en  20  fin  CHANMOPA  niveau
    NIVNOMME suite TSUPPRIM;
ignorer NIVNOMME (TSUPPRIM);
oterelem I;
si ADRPROGR = 0;
```

```
CONFIRME <== "PAS DE PROGRAMMES" en 1, 20;
afficher BARATINS;
allerà FINPROMA finsi;
texte BARATINS <== "SUPPRESSION DE L'UTILISATEUR" en 1, 20;
MOTDEPAS := 0;
I := 0;
J := 0;
réponse "FIN POUR TRAITER EN BLOC, FIN DE LIGNE POUR LE DETAIL"
    en 22 fin SUPENBLO niveau NIVNOMME suite FINLIGNE;
I := 1;
lirelem ADRPROGR;
SONPROGR := nomprog;
texte BARATINS <== SONPROGR en 1, 30 numéro 1;
SUPENBLO :  QUESTION <== "NOM DE L'UTILISATEUR" en 1, 22;
REPONSE <== "                          " en 22, 22;
afficher BARATINS;
afficher REPONSES;
ENBLOSUP :  mettre curseur REPONSE;
état NIVNOMME (TFIN, FINLIGNE) réponse TOTO;
enlever curseur REPONSES;
allersi TFIN en APERSONE;
lire 20 caractères de REPONSE dans NOMINDIV;
si NOMINDIV =" MOTDEPAS; allerà SELEMEME finsi;
K := 0;
cherelem LISINDIV avec individu = NOMINDIV réponse TOTO;
si TOTO = 0;
INFIRME <== "UTILISATEUR INCONNU" en 1, 24;
afficher BARATINS;
alarme;
état NIVNOMME (TFIN, FINLIGNE);
annuler INFIRME;
allerà ENBLOSUP finsi;
L := numéro;
si MOTDEPAS = 0;
liremem J;
linompro := K;
finsi;
J := L;
MOTDEPAS := NOMINDIV;
SELEMEME :  lirelem ADRPROGR;
L := adrsuite;
si J = 0;
J := ADRPROGR;
sinon;
metrelem ADRPROGR dans J;
finsi;
```

```
ADRPROGR := L;
si L = 0;
allerà FINSUPMA finsi;
si I = 0;
allerà SELEMEME finsi;
BLOSUPEN :  lirelem ADRPROGR;
SONPROGR := nomprog;
texte BARATINS <==> SONPROGR en 1, 30 numéro 1;
allerà ENBLOSUP;
FINSUPMA :  si MOTDEPAS ¬= 0;
lirelem J;
linompro := k;
finsi;
FINPROMA :  fin niveau NIVNOMME;
allerà SUIDEBUT;
APERSONE :  lirelem ADRPROGR;
J := adrsuite;
lirelem programe;
SESDECLA := déclares;
annuler CONFIRME;
allerà SUPPRIME;
SUMAITRE :  ADRPROGR := J;
si J = 0;
allerà FINSUPMA finsi;
si I = 0;
allerà APERSONE finsi;
allerà BLOSUPEN;
.
.
.
```

VU

Grenoble, le

Le Président de la Thèse

VU

Grenoble, le

Le Doyen de la Faculté des Sciences

VU, et permis d'imprimer

Le Recteur de l'Académie de GRENOBLE