Editeur de texte interactif: réalisation et évaluation

Yolande Aronovitz

N° D'ORDRE : 357

Année 1974

THESE

présentée

DEVANT L'UNIVERSITE CLAUDE BERNARD - LYON

pour obtenir

LE DIPLOME DE DOCTEUR DE SPECIALITE (3ème CYCLE)

DE MATHEMATIQUES (INFORMATIQUE)

par

Yolande ARONOVITZ

EDITEUR DE TEXTE INTERACTIF
REALISATION ET EVALUATION

Soutenue le 4 juillet 1974 devant la Commission d'Examen

MM. BOUCHE — Président

BELLISSANT, KOULOUMDJIAN, MAHL — Examinateurs

N° D'ORDRE : 357

Année 1974

THESE

présentée

DEVANT L'UNIVERSITE CLAUDE BERNARD - LYON

pour obtenir

LE DIPLOME DE DOCTEUR DE SPECIALITE (3ème CYCLE)

DE MATHEMATIQUES (INFORMATIQUE)

par

Yolande ARONOVITZ

EDITEUR DE TEXTE INTERACTIF
REALISATION ET EVALUATION

Soutenue le 4 juillet 1974 devant la Commission d'Examen

MM. BOUCHE — Président

BELLISSANT, KOULOUMDJIAN, MAHL — Examinateurs

UNIVERSITE CLAUDE-BERNARD - LYON 1

Président : Mr le Pr J. BOIDIN

Premier Vice-Président : Mr le Pr R. TOURAINE

Deuxième Vice-Président : Mr P. PONCET, Maître-assistant

Troisième Vice-Président : Mr C. BADOR, étudiant

Secrétaire Général de l'Université : Mr J. RAMBAUD, administrateur civil

Unités d'Enseignement et de Recherche (U.E.R)

U.E.R médicale GRANGE-BLANCHE : Mr le Pr D. GERMAIN

U.E.R médicale ALEXIS-CARREL : Mr le Pr C. GIROD

U.E.R médicale LYON-NORD : Mr le Pr J.P GARIN

U.E.R médicale SUD-OUEST : Mr le Pr J. ROBERT

U.E.R des Sciences Pharmaceutiques : Mr le Pr M. CARRAZ

U.E.R des Techniques de Réadaptation : Mr A. MORGON, Maître de Conférenc:

U.E.R de Biologie Humaine : Mr le Pr J.C. CZYBA

U.E.R. d'Education Physique et Sportive : Mr A. MILLON, Professeur d'E.P.S

U.E.R des Sciences Odontologiques : Mr le Dr R. VINCENT

U.E.R de Mathématiques : Mr le Pr G. MAURY

U.E.R de Physique : Mr le Pr R. UZAN

U.E.R de Chimie et Biochimie : Mr le Pr J. HUET

U.E.R des Sciences de la Nature : Mr le Pr J. BRUN

U.E.R de Sciences Physiologiques : Mr R. FONTANGES, Maître de Conférences

U.E.R de Physique nucléaire : Mr le Pr A. SARAZIN

U.E.R de Mécanique : Mr le Pr J. MATHIEU

Observatoire de LYON : Mr le Pr J.H. BIGAY

I.U.T 1 : Mr le Pr B. POUYET

I.U.T 2 : Mr J. GALLET, Directeur E.N.S.A.M.

Je désire exprimer ma grande reconnaissance à Monsieur le Professeur BOUCHÉ, dont l'aide m'a été précieuse tout au long de la réalisation de cette thèse, et qui a bien voulu me faire l'honneur de présider ce jury.

Monsieur BELLISSANT, Maître assistant à l'Université de Grenoble, et Monsieur KOULOUMDJIAN, Maître de Conférences à l'IUT1 de Lyon, m'ont conseillé pour la rédaction de ces pages ; qu'ils trouvent ici l'expression de mes remerciements.

Je tiens à assurer tout particulièrement de ma profonde gratitude Monsieur Robert MAHL, Professeur à l'Ecole Nationale Supérieure des Mines de Saint-Etienne, qui a dirigé mon travail. Sans ses encouragements, ses conseils, et son dynamisme, ce projet n'aurait jamais été réalisé.

Enfin, je remercie tous ceux qui m'ont apporté leur aide dans divers domaines :

- les membres du département Informatique - Gestion de l'Ecole des Mines, qui ont bien voulu tester YETI, aux risques et périls de leurs fichiers.
- Mesdames BONNEFOY et MONTMARTIN, qui ont tapé cette thèse, et dont le travail ne mérite que des éloges.
- Monsieur BROSSARD, qui a dessiné les histogrammes.
- Monsieur BRENOUX, de l'UER de Sciences de Saint-Etienne, qui a réalisé en un temps record le tirage de cet ouvrage.
- et surtout mon mari, qui, en tant qu'informaticien, a participé aux essais de YETI, et qui, d'autre part, m'a supportée pendant tout le temps de la réalisation et de la rédaction de ce travail.

TABLE DES MATIERES

- INTRODUCTION -

La fabrication de textes en général - livres, notices techniques, cours, circulaires administratives, etc... - a toujours été un problème désagréable à résoudre ; il faut écrire, raturer et recopier un certain nombre de feuilles de papier avant d'obtenir un manuscrit à peu près présentable ; une fois ce manuscrit tapé à la machine, il faut corriger les erreurs dont il s'est orné au passage, retaper, relire, etc ce sans compter les additifs ou les modifications auxquels on pense à la dernière minute. De plus, bon nombre de textes - notices techniques, cours.. demandent après fabrication une mise à jour régulière qui n'est pas non plus une opération plaisante.

La confection de programmes ne valait guère mieux dans les tout débuts de l'informatique : paquets de cartes ou manuscrits, perforeuses ou machines à écrire, les inconvénients sont un peu différents mais tout aussi nombreux. Avec le stockage sur disques ou sur bandes est né un type de programme utilitaire qui résoud d'une façon bien meilleure - au point de vue agrément et au point de vue rapidité - les problèmes de production et de mise à jour de programmes, et même de toute littérature : l'éditeur de texte . Un éditeur de texte permet d'enregistrer un texte, de le modifier ensuite "sans avoir à le sortir", et éventuellement de l'éditer.

Il faut peut-être, ici, ouvrir une parenthèse au sujet de cette appellation . Le terme anglais est "text editor" - terme approprié puisque "edit" signifie à la fois préparer un texte et l'éditer - ; le vocable français est le résultat d'une simple transposition, consacré maintenant par l'usage ; aussi a-t-il été conservé dans ces pages.

Un éditeur de texte a son mini langage personnel : un petit nombre de commandes (entre quinze et vingt habituellement), écrites d'une façon particulière, chacune ayant une fonction fixe : par exemple, supprimer une partie du texte, ou insérer un nouveau morceau.

L'implantation de tels utilitaires , pour mettre au point des programmes, sur une configuration classique avec lecteur de cartes et imprimante est maintenant chose tout à fait courante.

Mais on utilise de plus en plus des terminaux en communication directe avec un ordinateur - machines à écrire, télétypes, et plus récemment, écrans -. Ces terminaux remplacent plus qu'avantageusement le traditionnel circuit lecteur de cartes - machine - imprimante ; leurs réponses sont beaucoup plus rapides - surtout pour les écrans -, et ils permettent d'éviter au maximum les sorties sur papier d'étapes intermédiaires sans intérêt - D'où l'idée, de plus en plus répandue, de produire des éditeurs de texte interactifs, utilisables à partir de tels terminaux.Ces éditeurs, s'ils ont les mêmes fonctions que leurs homologues traditionnels, sont cependant conçus d'une façon tout à fait différente ; une part importante est faite au dialogue avec l'usager : messages nombreux temps de réponse réduits au minimum, etc...

Le chapitre I présente la structure et les caractéristiques externes habituellement rencontrées parmi eux. ([3],[4]).
Afin d'illustrer cette description, le chapitre II donne un panorama des principaux éditeurs de texte existants et de leurs particularités ([1] , [2] , [3] , [4] , [6]).

L'évolution générale, du paquet de cartes au programme enregistré, des périphériques classiques à l'interactif, est actuellement reproduite en raccourci dans beaucoup de centres de recherche. C'est ce qui s'est produit à l'Ecole des Mines de Saint-Etienne. Parti d'un petit ordinateur, le Centre de Calcul de l'Ecole possède actuellement un Philips 1175 qui gère quatre unités de disques, deux dérouleurs de bandes et quatre écrans alphanumériques, et un Télémécanique 1600 doté de deux terminaux interactifs : un télétype et un écran graphique. Le système normal du P1175 comprend un éditeur de texte classique, UPDATE ([18]) ; mais, à l'arrivée des écrans, il parut intéressant d'en posséder un qui soit intéractif. D'autre part, une liaison est en cours d'implantation entre le T1600 et le P1175

(le T1600 doit servir d'intermédiaire entre le P1175 et le réseau Cyclades) ; comme l'ordinateur Télémécanique a peu de possibilités pour stocker les programmes, ses utilisateurs placent leurs textes sur les disques du P1175 ; cela crée une demande assez importante pour un éditeur conversationnel.

La première idée était simplement de produire rapidement quelque chose qui marche et qui soit pratique. On commença par étudier les éditeurs de texte existants, et par faire une petite enquête auprès des utilisateurs éventuels.

Mais, pour produire un outil utile et efficace, il faut :
1°) étudier ce qu'il est possible et souhaitable de faire (enquêter sur les besoins des personnes, classer ce qui a plus ou moins d'intérêt, recenser les moyens dont on dispose, etc...)
2°) construire l'outil
3°) s'occuper a postériori de son évolution, pour le modifier et l'améliorer.

Ce sont là choses connues depuis longtemps, surtout dans le domaine des applications pratiques ; ainsi entre autre tout ce qui relève du hardware, dans les ordinateurs, est traité de cette façon.

Pendant longtemps, cette conception ne s'était pas appliquée au software ; on construisait quelque chose qui fonctionnait, c'était déjà beaucoup. Mais l'on s'aperçut vite que le temps de l'artisanat était passé, et l'on commença à s'intéresser aux mesures de système. ([18] , [19] , [21]).

Cette idée se développe de plus en plus : le premier volume de l'AFIPS 73 consacre toute une série d'articles à ce sujet ([16] , [17]). Il y est proposé, soit des idées générales, soit des techniques précises pour évaluer les performances d'un certain nombre de produits du software : systèmes, moniteurs, gestion des fichiers, compilateurs, etc...

Cette démarche semble naturelle lorsqu'on a dépassé le stade de la première expérimentation, car elle est la seule qui permet de faire un travail dont l'utilité n'est pas inversement proportionnelle au cube de l'âge (au moins).

D'autre part, elle se rattache à une méthode appliquée avec succès dans d'autres domaines de l'informatique ; en clair dans tous les domaines qui nécessitent la saisie et l'analyse d'un membre important de données.

Dans ces cas là, l'étude des données et la construction progressive d'une idée a postériori sont beaucoup plus efficaces qu'une opinion forgée d'avance ; d'autre part, comme on a souvent affaire à des problèmes qui se ressemblent, il est intéressant de systématiser la chose et de construire un véritable modèle .

Ainsi les mesures faites sur un système permettent elles de se faire une idée de son allure ; cette idée dirige les modifications à apporter ; de nouvelles mesures viennent corriger la première opinion, etc... et l'on obtient petit à petit la représentation d'une sorte d'idéal - peut-être pas facile à reproduire exactement, mais qui permet au moins de déceler quels défauts sont graves et quels autres peuvent demeurer sans trop d'inconvénients.

Toutefois, afin d'être efficaces, ces mesures ne doivent pas être des "pièces rapportées" au travail ; il vaut mieux prévoir le "recyclage" du produit dans sa conception même.

C'est dans cet ordre d'idées que YETI a été conçu.

On était déjà loin de l'idée de départ d'une simple construction utilitaire. On s'en éloigna encore plus en mettant au point l'interface avec l'usager. Cette partie avait été étudiée, au départ, sous l'angle pratique : temps de réponse minimum et maximum souhaitables, nature des messages, disposition sur l'écran.

Mais le principal avantage d'un programme interactif : le dialogue établi avec l'utilisateur, est singulièrement amenuisé si ce dialogue est, pour ainsi dire, standard. Bien sûr, il reste l'avantage de la rapidité, de l'action immédiate, qui est très important ; mais l'impression de "conversation" qu'a l'usager débutant disparaît rapidement.

D'autre part, un utilisateur est, par certains côtés, "mesurable ". Ce problème fut abordé pour YETI, aussitôt après avoir choisi les premières mesures sur les performances de l'éditeur ; en effet, beaucoup de résultats pouvaient être interprétés différemment suivant l'usager avec lequel ils avaient été obtenus ; il était donc indispensable d'étudier cela de plus près et, en quelque sorte, de mesurer l'utilisateur.

En confrontant ces deux faits, il apparut qu'on pouvait, non seulement tenir compte des mesures générales sur tous les utilisateurs pour faire à l'éditeur une modification définitive, mais aussi se servir des mesures faites sur un usager donné pour adapter l'éditeur à cet usager, et faire varier ses réponses en fonction de la personne à qui elles s'adressaient.

On vient de décrire ici brièvement la conception de YETI telle qu'elle s'est chronologiquement déroulée. Le chapitre III ([7],[4]) en donne une version détaillée, mais reconstruite afin d'avoir une vue d'ensemble des influences diverses, et de mettre en lumière les points importants.

Dans le chapitre IV ([14],[15]), YETI est présenté "vu de l'intérieur" : structure interne et interface avec l'utilisateur, tandis que le chapitre V est consacré à une description externe du "produit fini" tel qu'il apparait à l'usager.

Enfin, le dernier chapitre regroupe tout ce qui concerne les mesures. Tout d'abord, les techniques employées pour interpréter les résultats. En effet, les premières choses à mesurer étaient simples : temps d'exécution d'une commande donnée, temps de réponse de la machine ou de l'utilisateur, etc... Les premières modifications apportées furent également très classiques : optimisation de sous programmes, au sens gain de temps.

Mais la deuxième série de mesures a porté sur des phénomènes plus complexes ; pour citer un exemple entièrement traité par la suite : mesure des préférences des utilisateurs pour certaines commandes, suivant qu'ils sont ou non habitués à utiliser YETI, et en fonction du langage dans lequel leurs textes sont écrits. Pour traiter ce genre de choses, on a du faire appel à des techniques d'analyse de données, en particulier l'analyse des correspondances.

On trouve également au chapitre VI la description des mesures proprement dites - faites, en cours, ou prévues -, les premiers résultats significatifs obtenus, et leur interprétation.

Enfin, la conclusion, après avoir fait le bilan de l'état actuel de YETI, décrit les extensions possibles et les nouvelles orientations que l'on peut lui donner.

CHAPITRE I

EDITEUR DE TEXTE
DEFINITIONS ET CARACTERISTIQUES

§ 1 - STRUCTURE TYPIQUE D'UN EDITEUR DE TEXTE INTERACTIF - DIFFERENCES DE BASE

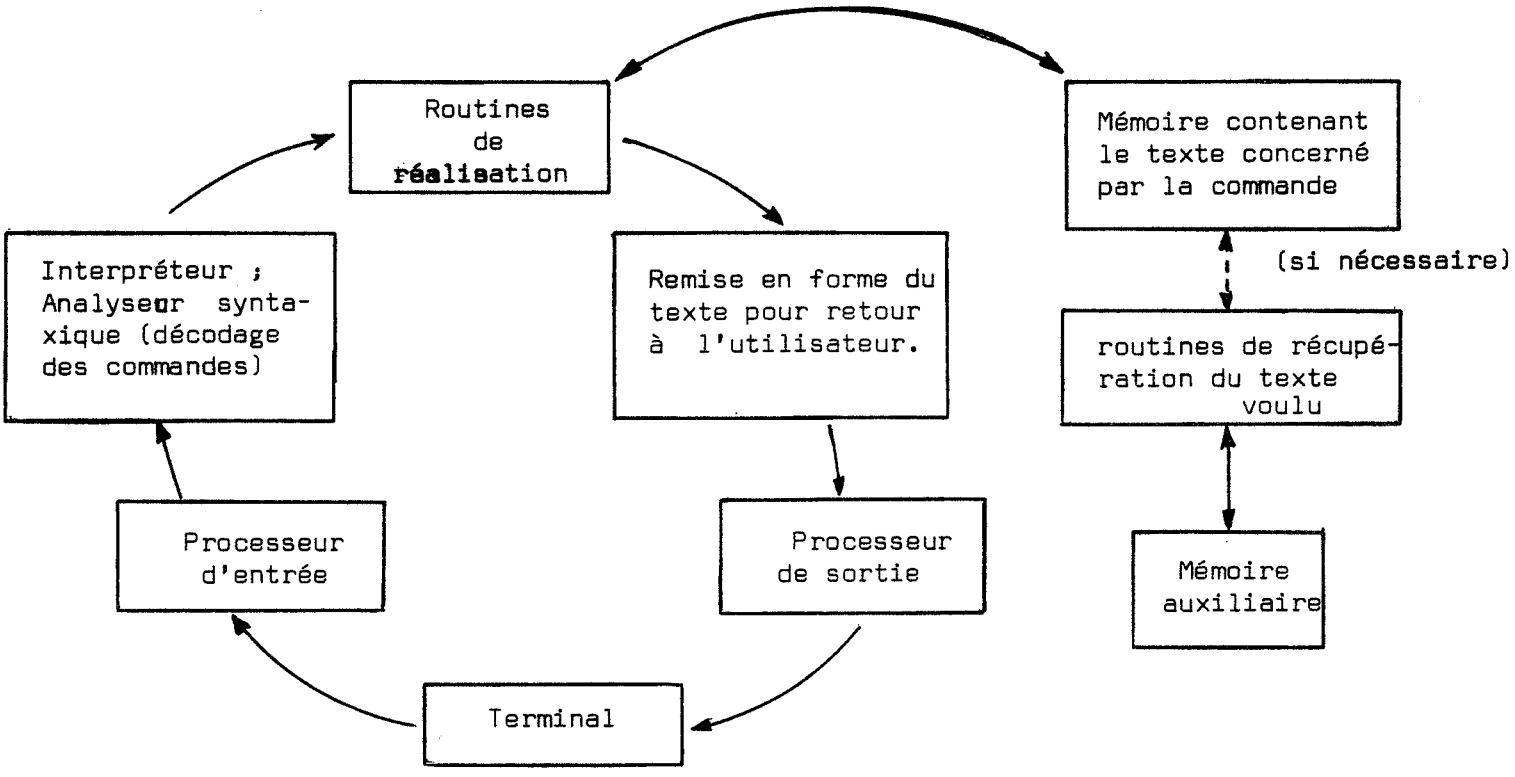


Figure 1

La figure 1 représente la forme la plus générale d'un éditeur de texte. L'utilisateur est installé au terminal, sur lequel il inscrit ses commandes. Le processeur d'entrée transmet ces commandes à l'analyseur syntaxique qui les décode, puis à l'interpréteur qui localise, dans la forme interne du texte, la partie concernée, et appelle les routines de réalisation ; ces routines effectuent l'ordre demandé ; à ce niveau là - à partir de la localisation de l'interpréteur -, le travail se fait sur la forme interne ; il est donc nécessaire de procéder à une remise en forme du texte, avant de le restituer à l'utilisateur à l'aide du processeur de sortie.

Lorsqu'on a à travailler sur des textes de grande taille, on utilise une mémoire auxiliaire (disque, etc...) ; il faut alors prévoir des routines pour mettre en mémoire centrale la portion de texte nécessaire.

Parmi les éditeurs interactifs, plus que parmi les éditeurs classiques, on peut distinguer deux types :

- l'éditeur de programmes
- l'éditeur de texte libre ("texte libre" signifie : tout texte courant, manuscrits de livres, notices techniques, circulaires, etc...)

Cette distinction est un peu artificielle dans la mesure où un éditeur peut presque toujours remplir les deux fonctions ; cependant, il est nécessairement plus adapté à l'une des deux.

D'autre part, cette séparation est accentuée par la différence entre les utilisateurs :

- l'éditeur de programmes est fait pour des informaticiens ; il a de nombreuses capacités, est assez complexe, et par là même doit souvent être implanté sur une configuration assez importante ; en fait,on le trouve surtout dans les centres de calcul et de recherche.
- l'éditeur de texte libre peut en principe être utilisé par n'importe qui, n'ayant aucune notion de programmation, pour modifier un texte quelconque. Pour être utile, il doit être facilement accessible à qui que ce soit. Cela suppose de pouvoir l'implanter à un grand nombre d'endroits (de préférence plus ouverts au profane qu'un centre de calcul), et une éventuelle commercialisation.

Les petites configurations étant bien plus nombreuses que les grosses, on en vient à réduire au minimum sa taille ; pour le moment, cette réduction s'accompagne d'une diminution notable de ses capacités.

On se trouve donc - actuellement - en face de deux types d'éditeurs de texte libre : ceux qui sont en fait des éditeurs de programmés utilisés aussi pour le texte libre, implantés dans de grands centres, ayant beaucoup de possibilités mais peu d'utilisateurs, et ceux qui sont exclusivement éditeurs de texte libre, implantés sur de petites machines et dotés de possibilités très rudimentaires.

Cette situation étant provisoire et nullement provoquée par des motifs scientifiques, les éditeurs spécialisés dans le texte libre sont cités seulement pour mémoire dans cet exposé. Pour les autres, les différences entre le mode texte libre et le mode programme seront évoquées au fur et à mesure.

Dans le schéma général présenté figure 1, la différence principale est la plus grande complexité de la remise en forme du texte : un programme est composé de lignes de longueur fixe (images-cartes), alors qu'un texte libre doit être formatté de façon précise : marges, paragraphes, etc... (cela est en général réalisé à l'aide de caractères spéciaux introduits au moment du passage à la forme interne, chaque caractère indiquant une particularité : aller à la ligne, sauter une ligne, une page, laisser une marge à gauche ou à droite, etc...)

Un autre motif de distinction entre les éditeurs interactifs est le genre de terminal pour lequel ils sont prévus : terminal lent - télétype, machine à écrire) ou terminal rapide - écran -.

Cette différence de vitesse - énorme : un télétype fait de 100 à 300 mots/minute, un écran de visualisation alphanumérique peut en faire 9600 - influe beaucoup sur l'interface avec l'utilisateur (sur un terminal lent, les messages sont plus brefs, les commandes plus concises, etc... De son côté, l'usager hésite à demander, par exemple, des impressions trop longues), donc sur la conception même de l'éditeur.

Ce sont là les deux motifs "matériels" essentiels de différence. D'autres en découlent directement,par exemple, la façon dont un texte est imprimé : les terminaux lents donnent une ligne à la fois, les écrans envoient toute une portion de texte.

Mais avant de s'étendre sur les ressemblances et différences dues au matériel, il convient d'étudier de plus près ce qui caractérise un éditeur de texte conversationnel.

§ 2 - *CARACTERISTIQUES D'UN EDITEUR DE TEXTE INTERACTIF.*

2.1. - LE POINT DE VUE DE L'UTILISATEUR :

Un éditeur a un rôle bien défini ; il doit pouvoir exécuter certaines fonctions de base, à savoir :

- création d'un texte nouveau
- insertion de morceaux supplémentaires dans un texte existant.
- suppression de parties d'un texte
- impression ou visualisation de tout ou partie d'un texte.

(c'est à dessein que l'on utilise ici des termes vagues tels que "parties", "morceaux" ; ces notions diffèrent suivant les éditeurs - voir § 3.2. de ce chapitre et d'une façon générale le chapitre 2).

A côté de ce minimum, il peut possèder diverses facilités et possibilités : remplacement d'un morceau par un autre, vérification de syntaxe pour un ordinateur orienté sur un langage de programmation particulier (cf [3] , [4] et [1] , éditeurs IPSS et EMILY).

Mais il ne faut pas oublier qu'un éditeur de texte est un moyen et non une fin ; quels que soient les usages particuliers qu'on désire en faire, et les renseignements qu'on espère en tirer (cf. chapitre "Evaluations et mesures"), sa fonction primitive est celle d'un simple outil.

Aussi :

1°) Le langage de commande doit être facile à retenir et concis. Il ne faut pas qu'il soit un obstacle pour l'usager, quelque chose qui s'interpose entre ce qu'il veut faire et la réalisation de ses désirs ; c'est le langage qui doit s'adapter au mode de pensée de l'utilisateur, et non l'inverse. ([2] ; cf cependant exception importante : [4])

2°) Les commandes doivent avoir de larges possibilités, sans trop de restrictions ou d'exceptions. Idéalement, on doit pouvoir agir comme on le ferait naturellement sur un paquet de cartes perforées, ou avec un crayon sur un texte écrit.

3°) Il faut utiliser toutes les capacités disponibles, ie celles d'un ordinateur (le plus souvent assez gros). Et cela, non seulement pour améliorer la technique (vitesse d'exécution, fiabilité du résultat), mais surtout pour aider l'utilisateur. Ainsi que le dit W.J. Hausen, à propos de conversationnel en général ([4]), "ne pas oublier que l'usager est un être humain ; et que, en tant que tel, il a deux particularités : il oublie et se trompe". Cet aspect de la question sera repris et développé ultérieurement (chapitres "conception de YETI" et "Evaluations et mesures").

2.2. - LE POINT DE VUE DU FABRICANT.

Si, dans leur grande majorité, les éditeurs de texte présentent à l'usager un visage assez semblable, il en va un peu différemment de leur constitution interne ; en fait, ce qu'ils ont en commun dans ce domaine, ce sont surtout les mêmes difficultés à résoudre. Ce sont ces difficultés qu'on va essayer de recenser ici, accompagnées de l'allure générale de leurs solutions.

Lorsqu'on veut produire un éditeur de texte, le premier problème auquel on se retrouve confronté est un problème de stockage : que faire du texte en cours de modification ? Le laisser en mémoire centrale est une solution rapide,mais encombrante et dangereuse (si un incident système se produit alors que l'éditeur fonctionne, le texte entier risque d'être détruit) ; le déposer dans une mémoire auxiliaire implique une récupération - plus ou moins longue, donc plus ou moins gênante. La solution la plus généralement adoptée est un compromis entre ces deux extrêmes : stockage du texte entier, cependant que l'on garde en mémoire centrale la portion sur laquelle l'utilisateur est en train de travailler.

Le deuxième problème est la localisation rapide d'une partie d'un texte. Il importe, lorsque l'usager veur avoir accès à tel endroit précis de son texte, qu'il puisse désigner cet endroit de façon non ambigüe ; il importe d'autre part, lorsque l'usager cherche quelque chose, qu'on puisse lui fournir l'emplacement de ce quelque chose avec le minimum d'indications de sa part.

Cela nécessite une certaine organisation du texte à l'intérieur de la machine. A quelques rares exceptions près (cf [3] : éditeur TECO), tous les éditeurs stockent le texte par lignes, de longueur fixe ([1] : IPSS, [3] : CMS, TVEDIT) ou variable ([6] : WYLBUR, [2] : QED).

Il suffit dès lors de numéroter ces lignes pour avoir des points de repère précis; cependant, si la numérotation à l'introduction du texte n'est pas un problème, les complications commencent dès la première insertion ou suppression.
Deux solutions sont possibles :

- ou l'on attend la fin d'une session pour procéder à une mise à jour des numéros; l'usager est alors tenu de travailler sur son texte par numéros de ligne croissants, sans retour en arrière. Cette solution, habituelle pour un éditeur non interactif, est de moins en moins employée, en raison de sa rigidité, peu compatible avec du conversationnel.

- ou l'on pratique une numérotation dynamique, refaite à chaque modification du texte. Le danger est évident : après quelques manoeuvres, l'utilisateur ne sait plus où il en est. Il existe deux méthodes courantes pour pallier ce risque :

* La numérotation décimale (cf [11] et [6]) : les lignes du texte d'origine sont numérotées 1, 2, 3, les lignes introduites après ont des numéros décimaux ; par exemple les lignes insérées après la ligne n seront étiquetées n.001, n.002, etc... . Le nombre de chiffres possibles varie suivant les éditeurs, le record revenant probablement à WYLBUR ([6]) : 4 chiffres pour numéroter les lignes à l'entrée, plus 3 chiffres décimaux possibles ; deux chiffres décimaux représentent un minimum pour ne pas limiter le nombre de lignes insérables à un endroit donné.

  Une bonne méthode (cf [11]) consiste à avoir une commande de remise à jour des numéros, qui, à la demande de l'utilisateur, renumérote normalement, sans décimales, l'ensemble du texte.

* Une renumérotation complète à chaque fois, mais assortie de nombreuses facilités pour désigner une ligne autrement que par son numéro : par exemple, en donnant une chaîne de caractères qu'elle contient. Cette forme d'adressage par contexte est de plus en plus répandue car elle est la plus naturelle à l'utilisateur.

  Autre exemple, le caractère spécial pour représenter la dernière ligne du texte, ou la prochaine ligne à lire.

Un additif utile peut consister en la présence d'une commande qui donne explicitement le numéro d'une ligne désignée par l'une de ces méthodes.

A la notion de ligne, un certain nombre d'éditeurs ajoutent la notion de page, qui représente un certain nombre, fixé en général, de lignes. Cela est surtout utilisé actuellement par les éditeurs qui n'ont pas d'adressage par contexte. Les commandes de type "page suivante" permettent alors de faire défiler plus rapidement le texte. (pour les éditeurs plus anciens, à numérotation statique, ce type de notion avait beaucoup plus d'importance).

Cette organisation en lignes, qui résoud le deuxième problème de fabrication, est malheureusement à l'origine d'une troisième difficulté ; le numéro de ligne étant le repère, les commandes portent en général sur des lignes entières : insérer telle ligne après telle autre, supprimer telle ligne. Et l'usager qui a oublié une virgule ou fait une faute de frappe se voit dans l'obligation de réécrire sa ligne. Si cela est inévitable lorsqu'il s'agit de cartes perforées, c'est beaucoup moins normal lorsqu'on travaille à un terminal en ligne directe.

Il faut donc penser à conserver au texte, dans sa forme interne, une structure suffisamment souple, afin d'avoir facilement accès au caractère dans la ligne.

On peut alors prévoir une commande permettant de remplacer, à l'intérieur d'une ligne, une chaîne de caractères par une autre (cette commande peut même être étendue à plusieurs lignes : remplacer, dans toutes les lignes où elle apparaît, la chaîne x par la chaîne y ; il convient cependant de manier ce genre de généralisation avec prudence).

Dans certains cas, ce service peut être rendu directement par le terminal (cf [14]).

Le dernier problème le plus généralement soulevé est la conservation des différents états pris par le texte ([5]). S'il est peu intéressant de retrouver une version d'un programme contenant des erreurs à la compilation, il est par contre agréable de pouvoir conserver plusieurs versions d'un programme, correspondant à divers cas ; dans le cas d'un texte libre, il peut être utile de comparer plusieurs rédactions.

Il n'y a pas jusqu'à maintenant de méthode générale de conservation.

Il resterait à énumérer quelques problèmes plus particuliers qui sortent du cadre d'un éditeur de texte proprement dit, mais qui se posent de plus en plus souvent : citons surtout, car c'est là le principal, la possibilité de lancer l'éxécution d'un job à partir de l'éditeur de texte, et une éventuelle récupération des résultats sur le terminal.

C'est une préoccupation qui apparaît d'elle même, dès qu'un éditeur a un peu servi ; l'usager qui vient de modifier un programme apprécie de pouvoir l'essayer immédiatement et contrôler ce qu'il a produit, là bien évidemment, la méthode de résolution dépend entièrement du système sur lequel est implanté l'éditeur, et il n'y a pas de solution générale.

CHAPITRE II

EDITEURS DE TEXTE EXISTANTS

On va, dans ce chapitre, passer en revue les éditeurs de texte les plus connus. Cela permet de se faire une idée des diverses façons de résoudre les problèmes internes, et surtout des différentes présentations vis-à-vis de l'utilisateur.

§ 1 - CONVERSATIONAL CONTEXT-DIRECTED EDITOR (IBM Cambridge Scientific Center)

([3] , [29]).

Cet éditeur est un éditeur de texte typique ; il fonctionne sur télétype ou machine à écrire. Sous le nom de EDIT, il permet de gérer des programmes. Une version à peu près identique a été associée à un imprimeur qui se charge de la mise en page - marges, interlignes, justification à gauche ou à droite, type des caractères, etc... - ; l'ensemble, baptisé SCRIPT, permet de traiter des textes libres. Il existe à Grenoble un exemplaire d'EDIT et un de SCRIPT.

La façon dont le texte est chargé diffère suivant sa nature ; cependant, il s'agit toujours d'une organisation en lignes : lignes de longueur fixe 80 caractères pour les programmes, enregistrements de longueur variable de 0 à 130 caractères, pour le texte libre.
Cette différence de stockage disparait en cours de traitement : les lignes du texte mis en mémoire centrale sont complétées à 130 caractères par adjonction de blancs, qu'elles proviennent d'un programme ou d'un texte libre.

Un petit inconvénient : les lignes de programme sont enregistrées telles quelles, sans compactage des blancs, ce qui prend beaucoup de place.

EDIT est activé par la commande :

EDIT <nom de fichier> <type de fichier>

Il fonctionne en deux modes :

- le mode "input", pendant lequel on introduit continuellement, ligne par ligne, du texte nouveau (création ou insertion de plusieurs lignes).
- le mode "edit", pendant lequel on peut faire les opérations habituelles d'examen et de modification de fichiers:

Le texte étant organisé en lignes, il existe une notion de "ligne courante" (on la retrouvera dans pas mal d'éditeurs) ; ici, c'est la prochaine ligne à créer (mode input) ou à modifier (mode edit).

Le mode "input" est automatiquement pris au début si le nom de fichier donné ne correspond pas à un fichier déjà existant. Dans ce mode, chaque ligne tapée par l'utilisateur est considérée comme une donnée et rangée dans le fichier. Pour écrire le texte à introduire, on dispose des facilités habituelles d'un télétype - caractères spéciaux pour annuler le caractère précédent, ou pour effacer tout ce qui précède sur la ligne -, et d'un certain nombre d'avantages supplémentaires : par exemple, une tabulation implicite suivant le type du fichier (cette tabulation peut être redéfinie par l'utilisateur s'il le désire), ou la possibilité d'introduire des caractères qui ne sont pas au clavier en attribuant leur code à un caractère existant.
On quitte le mode "input" en envoyant une ligne vide.

Le mode "édit" est automatiquement adopté au début si le fichier demandé existe. C'est également le mode dans lequel on passe si, étant en "input", on envoie une ligne vide.
Une ligne tapée par l'utilisateur est considérée comme une requête à exécuter. On peut insérer une ligne (pour en insérer plusieurs, il vaut mieux passer en mode input), supprimer ou faire lister une ou plusieurs lignes. Les changements à l'intérieur d'une ligne sont obtenus par l'une des trois commandes : "change", "overlay", ou "blank" ;

"change" permet de changer une chaîne de caractères donnés en une autre ; "overlay" permet de mettre des caractères non blancs à des emplacements précis sur une ligne, "blank" de mettre des blancs de la même façon. Enfin, on a la possibilité de conserver une copie d'un certain nombre de lignes, en les effaçant ou non (requête "putd" ou "put") ; on replace ces lignes ailleurs par la requête "get" ; comme il n'y a aucune numérotation externe apparente, on se repère dans le texte par rapport à la ligne courante. Toutes les requêtes s'y rapportent : "change", "overlay", "blank" agissent sur la ligne courante ; une suppression, un listage se font à partir de cette ligne, sur un nombre de lignes donné ; une insertion se fait nécessairement après la ligne courante. Afin de ne pas obliger l'usager à un traitement séquentiel peu intéressant, EDIT permet de modifier la ligne courante par de simples requêtes de déplacement : "next n" fait avancer de n lignes, "up n" fait reculer de n lignes ; "top" et "bottom" envoient respectivement au début et à la fin du fichier. Enfin, pour se positionner à une ligne, on peut indiquer une chaîne de caractères qu'elle contient : commandes "locate" (recherche sur toute la ligne) et "find" (recherche sur une zone donnée de la ligne).
On quitte le mode "edit" pour revenir à "input" en tapant "input" ou "save" ; l'action est identique, mais "save" fait en plus une sauvegarde sur disque du fichier. On termine une session par "QUIT" ou "FILE" (même différence qu'entre "input" et "save").

Une dernière remarque : EDIT est implanté sur un ordinateur à mémoire virtuelle. Cela augmente notablement les temps de réponse - tout le texte peut être considéré comme résident en mémoire - mais augmente les risques de destruction du texte en cas d'incident système. C'est à l'utilisateur de penser à sauvegarder son texte sur disque le plus souvent possible.

§ 2 - *WYLBUR (University Computation Center)*. ([3],[6]).

. Implanté à l'université de Stanford, WYLBUR donne une bonne idée de ce qu'on peut faire avec un éditeur de texte lorsqu'on a de gros moyens autour. C'est en fait, non seulement un éditeur, mais un système qui permet de lancer des jobs et de récupérer leurs résultats. Il travaille en association avec un moniteur particulier, MILIEN, et un système time-sharing pour l'exécution intéractive des jobs : ORVYL.

.Il est prévu pour un terminal de type machine à écrire, et il permet de travailler indifféremment sur texte libre ou programme.
l'interface avec l'utilisation a été particulièrement soigné :
- les commandes sont très proches de l'anglais courant. En fait, il est inutile d'avoir la moindre notion de programmation pour utiliser WYLBUR (ce qui est important car la modification de texte libre intéresse d'autres personnes que des informaticiens). Cela permet également de relire facilement une session.
- les corrections sont très faciles à faire (retaper un caractère faux)
- les messages d'erreur sont rapides et nombreux
- la gestion de texte libre n'est pas oubliée : il existe un certain nombre de commandes de mise en page, d'écriture en majuscules, etc...

. Il est organisé en lignes, de longueur variable de 0 à 133 caractères, mais les commandes opèrent en général sur un groupe de lignes, désigné de diverses façons :

- soit explicite : lignes 5, 7, 13, 20
  lignes 10 à 30
- soit associative : toutes les lignes contenant le mot "ALPHA"
- soit les deux à la fois : toutes les lignes comprises entre 10 et 30 et contenant le mot "ALPHA"

Il en est ainsi, entre autre, pour les suppressions, déplacements, impressions ; évidemment le groupe de lignes peut être réduit à 1.

La modification à l'intérieur d'une ligne existe ; elle peut avoir deux formes :

- action directe au terminal s'il s'agit d'une seule ligne : pour supprimer une chaîne, taper un D en-dessous du premier et du dernier caractère , pour insérer, taper I sous le caractère précédant l'insertion, suivi des caractères à insérer, pour remplacer une chaîne par une autre de même longueur, taper R suivi de la nouvelle chaîne.

- commande si la modification touche plusieurs lignes : par exemple changer "ALPHA" en "BETA" dans les lignes 1 à 10,ou partout où il apparait.

. De même que EDIT, il fonctionne sous plusieurs "modes": mode "collect" pour création de texte, mode "command" pour les modifications.

. La numérotation des lignes est une numérotation décimale (cf chapitre 1, § 3.2). Elle a l'avantage d'être calculée et visualisée par Wylbur.

Exemples : 1) Soit un texte numéroté 1, 2, 3, 4, 5, etc...On désire insérer une ligne après la ligne 3. Pour annoncer qu'il attend l'insertion, WYLBUR écrit : 3.001 ?

2) On insère un texte ; cela se fait ligne par ligne ; à chaque ligne insérée, WYLBUR demande la ligne suivante sous la forme : n? n'étant le numéro affecté à la ligne.

Cette impression du numéro, fort utile en général, peut être supprimée dans certains cas : par exemple pour demander un listing définitif d'un texte libre.

. On a aussi :

- la possibilité de récupérer une partie d'un texte que l'on mettra éventuellement ailleurs (commande "copy").
- une tabulation assurée directement par le terminal.

. Enfin, en plus des possibilités habituelles, on a trois commandes gérant l'exécution des jobs :

- "Run" pour lancer un job.
- "Print" pour récupérer un résultat et l'imprimer sur le terminal.
- "Fetch" pour récupérer un résultat et le stocker sur disque.

. Au point de vue structure interne, WYLBUR a une particularité, peu utilisée par les autres éditeurs : le texte est stocké sous forme compactée.

Exactement, une ligne est chargée, précédée d'un numéro qui indique sa longueur ; elle est ensuite découpée en "segments", contenant chacun au maximum 15 caractères blancs et 15 non blancs, et tous les blancs non significatifs sont ensuite compressés.

Ce type de stockage représente un gain de place important pour les programmes, les lignes de programme comportant généralement beaucoup de blancs. Cependant, toute action doit être précédée d'un décompactage, d'où un ralentissement qui doit être compensé par ailleurs.

Le texte définitif (sur disque) est conservé sous cette forme compactée ; cela permet également de conserver plusieurs versions d'un texte sans occupation excessive de place.

. Enfin, il existe intérieurement un système de pages, mais il est totalement transparent à l'utilisateur.

§ 3 - *QED. (Quick Editor) Com-Share, et Université de Berkeley.* ([2], [3])

Conçu à l'origine à l'Université de Berkeley, c'est un des exemples les plus connus d'éditeur sur télétype utilisant des lignes à longueur variable.
Du point de vue de l'utilisateur, il a l'avantage de posséder un langage particulièrement simple, et un accès au texte pratiquement indépendant de la ligne.
Les commandes sont conçues de façon à ce que :

1) l'utilisateur occasionnel ou débutant puisse s'en tirer avec trois d'entre elles : - insert, delete et print.
2) l'utilisateur habituel ait à sa disposition des possibilités nombreuses et variées ; par exemple :
   - modifier des caractères dans une ligne.
   - substituer une ligne à une autre.
   - définir une tabulation.
   - stocker dans un buffer particulier un texte souvent utilisé afin de pouvoir le récupérer rapidement (il y a 36 buffers internes de longueur fixe) ; à ce propos, il est à noter qu'une séquence de commandes de QED peut être enregistrée, stockée et modifiée comme si c'était un texte quelconque ; il est possible ensuite de faire exécuter cette séquence autant de fois que l'on veut sans avoir à la réécrire.

Un seul problème pour l'utilisateur débutant : bien se familiariser avec le mode d'adressage ; la numérotation est refaite entièrement à chaque modification (voir pourquoi ci-dessous, dans la description de la structure interne), et la façon la plus courante de désigner une ligne est l'adressage par contexte ou le caractère spécial (. pour la ligne courante, $ pour la dernière ligne) . L'adressage par contexte est semblable à celui de EDIT.

. Du point de vue structure interne, QED est assez particulier : il considère le texte introduit comme une seule chaîne de caractères (le "main text buffer"). Les lignes sont simplement délimitées par un marqueur interne ; elles peuvent avoir jusqu'à 500 caractères de long. (l'impression se fait de toute façon 80 caractères par 80 caractères au télétype).

La ligne n°n est celle qui est comprise entre le marqueur (n-1) et le marqueur n. (ce qui explique les modifications continuelles de numéros de ligne).
La conservation de plusieurs versions d'un texte n'est pas faite directement par QED, mais il est possible de la simuler, grâce au fait qu'une séquence de commandes de QED est assimilée à un texte normal : on garde, en plus d'une version d'un texte, la séquence de commandes nécessaires pour en faire une autre version ; il suffit des lors de faire exécuter cette séquence lorsqu'on a besoin de la version en question.

§ 4 - TECO (*Text Editor and Corrector*). *Massachussetts Institute of Technology, et Digital Equipement Corporation*. ([3] , [30]).

Un autre éditeur de texte sur télétype bien connu est TECO. C'est un modèle nettement plus primitif que ceux que l'on vient de voir ; seul parmi tous ceux cités, il ne possède pas de notion de ligne.La notion de base est la page.
Ces pages, de longueur variable, sont délimitées par des caractères spéciaux ; lorsqu'on désire modifier un texte, il faut le lire, page par page, depuis le début, et modifier à chaque fois la page que l'on est en train de lire.
Un retour en arrière est impossible (il faut reprendre au commencement).
Dès qu'on a fini de lire une page, elle est automatiquement sauvegardée sur disque.
La position où l'on se trouve, dans la page que l'on est en train de modifier, est repérée par un pointeur, qui pointe entre deux caractères ; on positionne ce pointeur, soit en lui indiquant le numéro du caractère où il doit se rendre, soit en lui précisant un déplacement de n caractères en avant ou en arrière.

A côté des commandes de base : lire, insérer, détruire, etc..., on a une série de commandes plus évoluées.
D'une part, on peut utiliser un adressage par contexte : se déplacer jusqu'à ce que l'on trouve la chaine x. On peut préciser s'il faut limiter la recherche à la page en cours, ou continuer jusqu'au bout du texte.
D'autre part, il existe, comme dans QED, 36 buffers internes, qui permettent de stocker du texte, réutilisable par la suite.
Egalement comme dans QED, ce texte peut être une séquence de commandes de l'éditeur. Il existe une macro-commande pour demander l'exécution de cette série.
Enfin, on peut indiquer un facteur répétitif devant une commande, ou demander l'exécution conditionnelle d'une certain nombre de commandes - en particulier dans les séquences stockées. Cela amène à ce qui fait le principal intérêt de TECO : ses commandes sont fort simples, primitives même, mais il est facile de les combiner pour faire des choses plus complexes. En bref, un utilisateur averti en arrive à composer de véritables petits programmes de commandes.

§ 5 - *TVEDIT* - *Stanford University* : ([3],[31]).

. Avec TVEDIT commence la série des éditeurs de texte appelables à partir d'un écran : c'est un des premiers réalisés (1960), aussi d'autres plus complexes ont-ils été faits depuis ; il est conçu, surtout, pour utiliser les caractéristiques de l'écran.

Il a deux modes de fonctionnement : le mode ligne et le mode caractère. Ces deux modes sont signalés sur l'écran par des pointeurs différents (une flèche en face de la ligne pour le premier, un rectangle au-dessus du caractère pour le deuxième).
Dans les deux cas, l'écran, qui est une "fenêtre" déplaçable sur le texte, est automatiquement cadré de façon à ce que le pointeur soit visible.
. Les commandes sont représentées par une série de clés alphanumérique, modifiables à l'aide d'un bouton placé sur le terminal.
. L'emplacement d'une modification est repéré par un pointeur lumineux sur l'écran ; elle se fait en écrivant ce que l'on veut à cet endroit là, pour une insertion ou un remplacement, ou en tapant un caractère de contrôle spécial, pour les effacements.

Les mêmes commandes ont une signification différente suivant le mode ; par exemple, "N K" efface N lignes à partir de la ligne courante en mode ligne, et N caractères à partir du caractère courant en mode caractère.
. Pour faciliter la lecture du texte, l'usage peut découper son texte en pages à l'aide de caractères spéciaux. Il a alors deux commandes à sa disposition : aller à la page suivante, aller à la page n (également introduites simplement en appuyant sur une touche). TVEDIT n'ayant pas d'adressage par contexte, ce système peut y suppléer (on visualise une page, et on cherche des yeux ce que l'on veut).

Les caractères spéciaux "limite de page" peuvent être détruits de la même façon que les autres.
. Un seul inconvénient : une ligne de texte peut avoir jusqu'à 128 caractères ; l'écran ne peut en donner que 50 ; les lignes ayant plus de 50 caractères sont donc tronquées. (Un caractère spécial est imprimé pour indiquer cette troncature).

Bien entendu, on peut déplacer la "fenêtre" de l'écran vers la droite pour voir la fin des lignes. Mais on ne peut les voir entières.

§ 6 - *IPSS (Interactive Programming Support System) - System Development Corporation* ([1],[3])

C'est un éditeur sur écran alphanumérique IBM2260, et il est fait un usage maximum des capacités de ce terminal.
Il possède toutes les commandes habituelles : insertion, suppression, impression, applicables à une ou plusieurs lignes ; la modification à l'intérieur d'une ligne est limitée au remplacement d'une chaîne de caractères par une autre de même longueur. La numérotation est décimale, avec possibilité d'une renumérotation périodique par commande. L'originalité de cet éditeur vient du fait qu'il est prévu pour créer et modifier des programmes écrits dans un langage particulier (JOVIAL). A ce titre, il possède un système qui vérifie la syntaxe de chaque nouvelle ligne introduite, ce qui raccourcit notablement la première mise au point d'un programme.

La vérification de syntaxe peut être une partie importante d'un éditeur; cependant, les éditeurs qui le font sont généralement limités car la syntaxe du langage est décrite à l'intérieur du système, et non à l'aide de tables qui permettent d'appeler, suivant la nécessité, la syntaxe de tel ou tel langage.
Il faut cependant remarquer qu'avec le développement des compilateurs interactifs, il n'est peut être pas indispensable d'avoir de telles propriétés sur un éditeur de texte, dont ce n'est pas le but principal. (Sauf dans certains cas : voir le paragraphe suivant).

§ 7 - *EMILY - Argonne National Laboratory.* ([4],[3])

Si l'on doit classer EMILY quelque part, c'est parmi les éditeurs de texte ; il serait probablement meilleur de ne pas la classer du tout. Si elle est citée ici, c'est non pour ses caractéristiques d'éditeur, mais pour la conception qu'y est appliquée de l'interaction homme-machine.
Dans tous les éditeurs précédemment cités, un texte est appréhendé comme une chaîne de caractères, éventuellement débitée en tranches - lignes, pages etc... - et les modifications relèvent essentiellement de la topographie : trouver le lieu et la place.
Pour EMILY, la démarche est totalement différente : un texte a une structure ; c'est cette structure qui est construite tout d'abord. Ensuite on détaille, progressivement, de sous-structure en sous-structure, jusqu'aux mots.

Cette façon de faire correspond beaucoup plus à une "création" réelle que celle d'un éditeur normal ; en fait, c'est celle que l'on adopte lorsqu'on fabrique un programme - ou un texte quelconque - sur le papier : d'abord un plan général, puis un organigramme détaillé, enfin le texte en langage de programmation. Rares sont ceux qui, aux prises avec un problème complexe, se mettent immédiatement à le programmer.
Or, lorsqu'on compose un programme à un terminal en ligne directe,on a le choix entre deux solutions : avoir déjà fait sur papier les étapes intermédiaires. - plan, organigramme - ou programmer directement. Les auteurs d'EMILY ont voulu en offrir une troisième.
Bien sûr, on ne peut pas utiliser n'importe quel langage ; le système possède la description syntaxique d'un certain nombre d'entre eux, et il faut choisir parmi ceux là.
L'idée de base est qu'un certain nombre de langages peuvent être décrits comme un ensemble de règles de remplacement de symboles (cf métalangues, type métalangue de Backus-Naur). Sont appelés non-terminaux les symboles qui peuvent être remplacés et terminaux ceux qui ne le peuvent pas.

Exemple : le IF logique du fortran peut se représenter par :

IF (<comparaison>) <action>

où l'on peut décomposer <comparaison> en :

<comparaison> : : = <expression><symbole de comparaison><expression>

n'est pas un symbole terminal, non plus que <expression> où <symbole de comparaison>.
Mais dans :

<symbole de comparaison> : : = ⌊.LT.|.GT.|.LE.|.GE.|.EQ.|.NE.⌋

l'un quelconque des choix du membre de droite est un symbole terminal.

Partant de là, on demande à l'usager d'introduire le "squelette" de son texte ; ensuite, EMILY prend le texte depuis le début et s'arrête à chaque symbole non terminal. Pour chacun, elle propose un choix de possibilités ; ces diverses possibilités sont inscrites sur l'écran. (l'écran est divisé en trois parties : celle du haut contient le texte en formation, celle du milieu permet l'entrée des identificateurs et la sortie des messages d'erreur, et sur celle du bas s'inscrivent les choix pour le symbole en cours de remplacement).
L'usager désigne avec un crayon lumineux l'option qu'il choisit, et le remplacement est fait ; si le résultat obtenu est non terminal, il contient d'autres symboles à remplacer ; un nouveau choix est alors proposé, etc... - Lorsqu'on est parvenu à un symbole terminal, on passe au remplacement suivant, ce jusqu'à ce que tout le texte soit parcouru. Si l'on rencontre, en cours de remplacement, des éléments au choix du programmeur (variables, identificateurs, etc...), celui-ci peut les introduire dans la partie médiane de l'écran.

Cela permet de construire un programme à partir d'une idée initiale ; mais l'utilisateur peut, à tout moment, changer cette idée ; certaines parties de ce qu'il a écrit lui paraissent à repenser.
Il peut alors demander une vue du "squelette" de son programme, à un endroit déterminé, ou dans son ensemble. (le mode de stockage permet de conserver une trace des remplacements).
Cette propriété est également utilisable pour les modifications : insertions, remplacements, suppressions. Elles peuvent être faites sur le texte détaillé ou sur le "squelette".
Il serait fastidieux de décrire une par une toutes les autres capacités d'EMILY ; disons simplement qu'elles concourent toutes au même but : permettre à l'utilisateur de composer vraiment son texte à l'écran, et lui fournir pour cela toute l'aide possible.

Un effort particulier est fait pour minimiser le travail de mémorisation de l'usager ; un ordinateur a une grande mémoire, autant s'en servir. Ainsi, les diverses possibilités pour un remplacement sont présentées, et l'utilisateur n'a qu'a choisir (cela limite aussi les erreurs,impossible de faire un remplacement incorrect) ; ou, entre autres : les morceaux de texte supprimés sont placés dans un fragment spécial appelé DUMP. Si le nouveau texte ne va pas, on peut retrouver l'ancien sans avoir à le reconstituer.
Bien sûr, des problèmes demeurent : par exemple le choix des identificateurs (encore que EMILY fournisse la liste des identificateurs existants, ce qui permet d'en retrouver un, ou de ne pas nommer de la même façon deux entités différentes) ; mais le fait de soulever des problèmes à ce niveau montre déjà à quel point on en est arrivé.
Finalement, le principal obstacle est l'adaptation de l'utilisateur ; il doit (cf W.J. Hansen [4] "changer des habitudes de pensée vieilles de beaucoup d'années" ; mais les auteurs d'EMILY ont fait tout ce qu'il fallait pour faciliter cette adaptation. Une telle expérience aura certainement beaucoup d'intérêt dans les années à venir, lorsque l'utilisation d'un écran sera devenue chose tout à fait commune.

CHAPITRE III

CONCEPTION DE YETI
INFLUENCES PRELIMINAIRES

§ 1 - L'IDEE DIRECTRICE :

La première et principale influence préliminaire sur la conception de YETI fut l'idée qui se dégagea, rapidement, du contact entre une nécessité matérielle précise : construire un éditeur de texte interactif, et des études théoriques : les motifs qui conduisent à mesurer des systèmes et les méthodes pour le faire. Ainsi qu'on l'a vu dans l'introduction, la notion de modèle a été reconnue comme indispensable si l'on voulait produire un travail sérieux, ce d'autant plus que - cf introduction et chapitre 6 - ce travail, une fois réalisé était destiné à servir à une autre série de mesures, cette fois sur ses utilisateurs.

On a donc été guidé, tout au long des choix à faire, par deux idées :

1°- L'éditeur devait être aisément mesurable ; c'est dans cet ordre d'idées que YETI a été doté d'un certain nombre de petites routines, qui sont pour ainsi dire ses instruments de mesure. 2° - Il devait être également aisément modifiable ; pour cela, il a été conçu avec une structure modulaire : modularité au niveau des commandes - on peut adjoindre ou enlever un nombre quelconque de commandes en changeant deux lignes par commande dans le programme principal, et en ajoutant ou enlevant des sous programmes entiers -, mais aussi à l'intérieur de l'éxécution d'une commande : le déchiffrage et la vérification

de syntaxe sont toujours indépendants de l'éxécution proprement dite de la commande, ce qui permet de modifier une partie sans toucher à l'autre (certaines commandes, impression, insertion sont même beaucoup plus découpées. Voir ch. 6)

- les mesures sur les utilisateurs n'auraient un sens que si "l'instrument de mesure" était suffisamment objectif. YETI devait donc avoir ce qu'on peut appeler une attitude neutre vis-à-vis de l'usager. Cela créait quelques difficultés, dans la mesure où l'on désirait également tenir compte, de façon importante, de cet usager. On a finalement résolu le problème en se bornant à faire, à priori, les suppositions communes à tous travaux conversationnels. On verra ci-dessous - § 3 - qu'elles sont plus contraignantes pour l'auteur du programme interactif que pour ses utilisateurs.

§ 2 - *LE MATERIEL UTILISE - CONTRAINTES DIVERSES* .

(2-1) Multiprogrammation :

Le plus gros problème posé par le matériel fut le fait que le moniteur du P1175 fonctionne en multi-programmation et non en temps partagé Il faut un certain temps pour qu'un travail soit pris en compte, temps qui dépend, entre autres, de la place qu'il va occuper. D'où la nécessité de minimiser l'encombrement du programme ; cette minimisation a été réalisée, d'une part dans la conception et la programmation même, d'autre part en utilisant des techniques de recouvrement sur le programme achevé - cela a été facilité par la constitution modulaire adoptée -. L'occupation actuelle est de 35 K-octets, sur les 512 mis à la disposition des programmes courants par le P1175 (il reste 96 K-octets, en mémoire rapide, mais qui ne sont pas accessibles à l'utilisateur moyen). Pour le moment, cela suffit largement, et il n'a pas été nécessaire de mettre en place le système initialement prévu de réservation horaire (priorité aux utilisateurs de YETI à certaines heures de la journée).

(2-2) Utilisation des écrans :

Ils ont demandé deux précautions :

- premièrement, adapter les messages à leur rapidité ; cela signifie que l'on laisse à l'usager la liberté de choisir le temps pendant lequel il peut regarder ce qu'a écrit l'éditeur : en pratique, un message n'est effacé que si l'utilisateur le demande expressément (soit en réclamant la suite du message s'il y a lieu, soit en composant une nouvelle commande).

Cela signifie également - ce qui est cette fois un agrément - que l'on peut envoyer des messages relativement longs (limite : la taille de l'écran) - ce contrairement à un télétype, où les réponses n'ont en général qu'une ligne, à cause de la lenteur d'écriture.

- deuxièmement, prévoir une sauvegarde fréquente du texte modifié. En effet, en cas d'incident-système, quel qu'il soit, le texte en mémoire centrale est perdu ; il ne reste que ce qui se trouve sur support externe, disque ou bande, et le contenu de la mémoire du terminal, s'il en a une. Or, les écrans en usage à l'Ecole des Mines de St-Etienne ne possèdent pas de mémoire propre. Comme on le verra au chapitre 4 - "stockage et organisation interne" - cette sauvegarde est faite de façon automatique, et totalement transparente à l'utilisateur - afin de ne pas occuper son esprit avec des problèmes qui ne sont pas les siens -.

Une autre contrainte a été imposée par la constitution des bibliothèques disque du P1175, puisque YETI est prévu pour gérer des programmes enregistrés ou à enregistrer sur ces bibliothèques. Ainsi qu'on pourra le constater - également au chapitre 4 - cela a beaucoup orienté la structure interne de YETI, mais la contrainte s'est en fait révélée plutôt fructueuse (stockage compacté, etc...).

§ 3 - *LE POINT DE VUE HUMAIN* :

Avant tout projet, une enquête rapide a été menée auprès des utilisateurs éventuels de YETI, afin de savoir, d'une part, ce qui leur plaisait et leur déplaisait dans les éditeurs de texte qu'ils avaient eu l'occasion d'utiliser ou de connaître, d'autre part, ce qu'ils souhaitaient trouver dans un éditeur interactif. Une étude des principaux éditeurs de texte intéractifs existants (voir chapitre 2) a été conduite en parallèle. Les résultats de ces opérations ont permis de cerner à peu près les commandes intéressantes.

D'autre part, afin de construire un "interlocuteur possible" pour un être humain normal, il était nécessaire d'étudier quelques particularités et temps de réponse se rattachant aux simples activités humaines. Il peut être utile de les rappeler ici :

Tout d'abord, le temps moyen de réponse, dans les rapports humains, est d'environ deux secondes. Une certaine quantité de littérature fait même de ces deux secondes une limite extrême, dont le dépassement implique une gêne certaine, que ce dépassement soit produit par une absence de réponse - exemple : les "anges qui passent" dans les conversations - ou par l'introduction d'éléments étrangers entre la phrase initiale et la réplique attendue - conversation interrompue par un élément extérieur -

Il semble qu'il faudrait nuancer ce jugement : les activités humaines ne sont pas quelque chose de continu ; il y a des moments où l'on peut attendre et des moments où l'on ne peut pas.

En fait, on peut se représenter chaque activité définie sous forme de tranches, chaque tranche formant un bloc dans lequel une attente est perçue comme pénible, alors qu'elle est parfaitement supportable entre deux tranches. Pour reprendre un exemple connu (cf [7]), soit une personne qui désire téléphoner. Elle a trois choses à faire :

- recherche du numéro
- appel
- communication

Premier stade : recherche. Toute interruption est un facteur d'agacement, et même d'oubli (si elle dure trop, on ne sait plus ce que l'on voulait chercher).
Même chose pendant l'appel : un événement extérieur qui se produit pendant la composition du numéro oblige presque toujours à le recommencer, une attente un peu longue des diverses sonneries énerve, etc...
Et il en est de même au cours de la conversation ; par contre, ces trois actions peuvent fort bien être relativement éloignées l'une de l'autre, dans la mesure où l'on sait que l'intervalle sera long (exemple : attendre au téléphone, lorsqu'on a obtenu le numéro, est très supportable si l'on sait que la personne intéressée est prévenue et arrive).
Cette capacité ou incapacité d'attendre, suivant le moment, est liée à la notion de "présent psychologique" : ce que l'on conçoit mentalement comme présent à un instant donné est ce qui se situe dans un intervalle de 2,5 à 3,5 secondes autour de cet instant. Ce qui correspond à une "tranche" d'activité doit être réalisable ou concevable dans cet intervalle.

Tout cela, qui a été mesuré sur des activités humaines courantes, doit être pris en considération lorsqu'on veut fabriquer des systèmes intéractifs : pour paraître naturels, les délais de réponse doivent respecter les temps habituels des rapports humains.

D'une façon générale, il faut éviter les temps trop longs - à plus de 15 secondes, il est inutile de faire conversationnel - ou trop courts : l'utilisateur doit avoir le temps de réaliser globalement ce qui se passe. Plus précisément ([7]) :

- dès l'appel, le système doit réagir : allumage de l'écran, ou, s'il est déjà allumé, message immédiat, pour prouver qu'on a bien entendu. Temps moyen : 0,1 à 0,2 secondes.
- une réponse à une question du Type "système, m'entendez-vous ? " ne doit pas excéder 5 secondes.
- une réponse à une demande de travail ou d'information peut varier de 2 à 5 secondes, suivant la difficulté de ce qui est demandé.
-pour une réponse à "Système, me comprenez-vous ?", il est préférable d'attendre 2 secondes (cf ci-dessus : laisser à l'utilisateur le temps de réaliser).

Les temps de réponse de YETI ont été mesurés, et ajustés de façon à respecter au maximum l'ensemble des limites citées ci-dessus.

Mais les délais de réponse, qui suffisent à tout détruire s'ils sont incorrects, ne suffisent pas pour autant à réussir une intéraction homme-machine. Il reste quelques règles à respecter :

§ 4 - *PRINCIPES GENERAUX DU CONVERSATIONNEL* :

On peut en dénombrer cinq, un qui est primordial et quatre autres, secondaires, qui en découlent plus ou moins. - La liste ci-dessous est essentiellement due à W.J.Hansen. ([4]). :

1. Connaitre l'utilisateur :

Son éducation, son expérience, le temps dont il dispose, sa dextérité manuelle, sa patience, ses réactions au système, sont autant de facteurs qui influent nettement sur son comportement, parfois de façon contradictoire. Ces différents aspects n'ont pu, évidemment, être retenus avec toutes leurs subtilités pour une première construction. Dans sa forme actuelle, YETI s'adresse à un utilisateur-type, auquel on s'est contenté de supposer le moins possible de connaissances et de capacités (ce dans les limites du raisonnable). En particulier, on a évité les a priori du type : "quelqu'un qui connait la programmation doit être capable de ..." ; - chose tentante car les utilisateurs actuellement prévisibles la connaissent tous plus ou moins -, puisqu'aucune notion de programmation n'est réellement nécessaire pour se servir de YETI. Cet état de choses sera modifié lorsque les mesures prévues - cf chapitre 6 - auront donné des résultats appréciables ; en effet, elles sont destinées principalement à permettre de différencier au maximum les utilisateurs, suivant leurs caractéristiques réelles. Face à quelqu'un qui concait déjà trois ou quatre langages de programmation, on peut raisonnablement poser l'à priori ci-dessus, et on le fera ; on l'évitera par contre soigneusement si l'on à affaire à la personne - actuellement rare, mais on ne peut prévoir à long terme sur ce plan-là - qui veut travailler sur un texte libre et n'a pas de connaissances informatiques particulières.

Une chose, par contre, qui a pu être en compte tout de suite, c'est que l'utilisateur est, avant tout humain. On oublie trop vite cette évidence lorsqu'on travaille sur un ordinateur ; on prend l'habitude du fait que l'homme adapte son langage et sa pensée aux besoins de la machine(chose relativement facile puisque, pour se faire comprendre d'un ordinateur, il suffit de suivre des règles précises) et on est fort surpris de trouver la nécessité d'accomplir le chemin inverse - bien moins agréable car l'ordinateur actuel est peu enclin aux subtilités psychologiques ... - . L'usager, donc est humain ; et il faut savoir, et admettre, qu'il oubliera :

1) de faire ce qu'il veut
2) ce qu'il y a dans son texte
3) et s'il est interrompu : ce qu'il voulait faire.

Les quatre principes qui suivent ont surtout pour objet de pallier ces oublis.

2. Minimiser la mémorisation :

Il s'agit, bien entendu, de celle qui incombe à l'utilisateur. Un ordinateur possède une mémoire importante, et-sauf incident électronique - sans défaut, c'est-à-dire exacte. Un homme, s'il a d'indéniables qualités dans d'autres domaines, ne peut cependant prétendre à autre chose qu'à une mémoire approximative, parfois longue à réagir, et dont la plus grande part est inconsciente. Un raisonnement élémentaire sur l'utilisation des compétences conduit donc directement à confier l'essentiel de la mémorisation à l'ordinateur. L'usager, ayant moins de choses à retenir, en aura moins à oublier, et commettra d'autant moins d'erreurs. Par exemple, il est normal de donner, dans un manuel de référence écrit, la syntaxe des diverses commandes d'un éditeur de texte. Mais il est plus simple et plus efficace de la mettre également dans la machine - où l'utilisateur peut la demander quand il le désire -, plutôt que de compter exclusivement sur la mémoire de l'usager ou sur la présence à proximité d'un éventuel manuel de référence. Bien sûr, on peut rétorquer qu'il faut encore que l'usager se rappelle la façon de demander cette syntaxe - mais il faut bien tout de même lui supposer un minimum de possiblilités. On peut d'ailleurs l'aider en choisissant quelque chose de particulièrement "parlant" à la mémoire - dans YETI, il suffit d'écrire "?" sur l'écran, ce qui a paru être largement à la portée de tous les utilisateurs.

3. Optimiser les opérations :

Cela est directement en rapport avec le § 3 précédent, en ce qui concerne les temps de réponse. Un utilisateur qui s'ennuie devant un écran inerte est tout préparé à commettre une collection d'erreurs et d'oublis lorsqu'il aura à nouveau des initiatives à prendre. Il est donc important que les actions de l'éditeur soient suffisamment rapides ; si certaines choses sont vraiment longues à réaliser, il est toujours possible de les découper en apparence, en envoyant sur l'écran des messages destinés à faire patienter l'utilisateur. L'effet psychologique compensera largement le léger temps perdu en transmissions. Dans cet ordre d'idées - effet psychologique -, il ne faut pas sous estimer la valeur d'un message plus ou moins humoristique ; cela peut paraitre déplacé dans un contexte sérieux, mais il faut penser que l'usager qui attend est le plus souvent tendu,

contracté, et que tout ce qui peut le détendre aura un effet bénéfique sur sa clarté d'esprit et ses capacités mentales de tous ordres.

Rappelons d'autre part un écueil dont il a déjà été question précédemment : l'excès de vitesse. Celui-là a l'avantage d'être facile à traiter à partir du moment où l'on en a pris conscience.

En résumé, la notion d'optimisation doit-être entendue dans un sens extrêmement strict : est optimal non ce qui va le plus vite, mais ce qui est le plus synchrone avec l'utilisateur.

4. Soigner le traitement des erreurs :

Cela recouvre plusieurs actions :

- Tout d'abord, prévenir au maximum les erreurs possibles : utiliser des commandes simples, sans ambigüité, pouvoir facilement revenir en arrière, etc... Les trois points cités précédemment jouent un rôle important dans ce traitement préventif.

- Ensuite, envoyer des messages d'erreur précis : il ne faut pas que l'usager passe plus de temps à chercher l'erreur qu'à la corriger. Outre le temps inutilement perdu, cela crée chez l'utilisateur un sentiment profond de désagrément, il conçoit une certaine rancune à l'endroit de cette machine qui lui dit "vous vous êtes trompé ...", qui ne l'aide pas, et qui a l'air de se moquer de lui. Cela ne peut que le conduire à d'autres erreurs ou à un abandon pur et simple de ce qu'il voulait faire.

- Enfin, si l'on constate qu'une erreur se produit fréquemment, cela vient certainement de l'éditeur : il y a quelque chose de peu pratique à écrire, ou de trop compliqué à comprendre, ou de trop difficile à réaliser. Cela peut recouvrir des motifs très variés :

- par exemple, on a oublié de faire une commande spéciale pour une action courante, et elle doit être éxécutée à l'aide d'une suite de manoeuvres peu pratiques. (C'est heureusement assez rare, et YETI a échappé à ce genre d'ennuis - ce qui fut fort agréable).

- ou bien, une opération simple pour qui connaît la structure interne de l'éditeur devient quelque chose de tout à fait arbitraire pour l'usager qui voit les choses de l'extérieur. Cet inconvénient était à craindre tout ce qui se rattache à la notion d' "espace de travail" dans YETI (voir chapitre 4). Vu la situation actuelle, il semble que ces craintes étaient vaines : certes, les usagers ont besoin de lire attentivement la partie du manuel de référence qui décrit cet espace et sa raison d'être ; mais ceux qui passent, pour ainsi dire, aux "travaux pratiques" sur écran n'ont pas de problèmes à ce sujet.

- ou encore, une suite de caractères à taper dans une commande est "mal placée" sur le clavier ; par exemple, un caractère sur deux doit être tapé avec la touche "majuscule" enfoncée, ou des caractères consécutifs se trouvent à des extrémités différentes etc... Ces petits problèmes de topographie du clavier peuvent acquérir une grande importance si la suite de caractères en cause doit être reproduite souvent.

Dans tous les cas, la seule méthode consiste à rechercher la cause d'erreur et à la supprimer.

5. Etudier la sécurité d'utilisation :

Un système de traitement des erreurs ne peut être parfait, surtout lors des premiers essais. De toute façon, il existe toujours, parmi les erreurs extérieures - système, machine - certaines dont on ne peut venir à bout. Un système conversationnel agissant vite, les dégâts peuvent être faits avant même que l'utilisateur ait réalisé ce qui se passe. Il importe donc d'organiser l'éditeur de façon à préserver le plus possible les textes en cours de traitement contre tous les types d'erreur, ce sans que l'usager ait à intervenir trop souvent. On verra chapitre 4 la notion "d'espace de travail" de YETI ; elle est apparue comme la solution la plus simple pour garantir le texte en cours de modification à la fois contre la machine et contre l'utilisateur.

CHAPITRE IV

CONCEPTION ET REALISATION
STOCKAGE DU TEXTE ET ORGANISATION INTERNE
INTERFACE AVEC L'UTILISATEUR

§ 1 - STOCKAGE ET ORGANISATION INTERNE :

(1-1) Généralités : L'organisation interne de YETI est d'abord basée sur la sécurité. La protection à réaliser, ainsi qu'on vient de le voir (fin du chapitre précédent) était double : protection de l'usage contre les incidents extérieurs d'une part, contre lui-même d'autre part.

Elle a été obtenue en intercalant entre l'utilisateur et son propre texte un espace de travail. Plus précisément : les textes - en général des programmes traités par YETI se trouvent ou sont destinés à se trouver sur des bibliothèques-disques (voir ci-dessous 1-2). La première opération que l'utilisateur doit accomplir pour pouvoir modifier un texte est de l'introduire dans l'"espace de travail" (voir ci-dessous 1-3) de YETI (par copie si le texte existe, sinon par une introduction directe à l'écran. Il existe deux commandes dans l'éditeur qui exécute cela). Ensuite, il peut agir à sa guise, supprimer, insérer, modifier, etc... Tout cela n'affecte que le texte situé en espace de travail.

Lorsqu'il parvient enfin à une nouvelle forme du texte qu'il désire conserver, il a à sa disposition une commande qui lui permet de faire une copie sur sa propre bibliothèque. Ainsi l'usager peut faire des essais sans risques pour son propre texte - s'il détruit tout en espace de travail, il n'a qu'à prendre une nouvelle copie et recommencer.

D'autre part, chose totalement transparente à l'utilisateur, l'espace de travail est lui-même situé sur disque. Le texte est donc constamment sauvegardé, à l'abri des éventuels incidents système, la seule partie en mémoire centrale à un moment donné étant la ligne sur laquelle on travaille.

(1-2) Les bibliothèques-disque :

Elles jouent dans la structure de YETI un rôle important, il a donc paru utile d'en donner un bref aperçu pour clarifier l'exposé.
Vues de l'extérieur, ces bibliothèques se présentent comme des espaces disque que l'on peut réserver, afin d'y entreposer des textes. Lorsqu'un texte est enregistré (par la commande W de YETI, ou à l'aide de l'éditeur non conversationnel UPDATE), il constitue un "membre" de la bibliothèque, doté d'un nom (celui que l'on a choisi en l'enregistrant).

La liste de tous les noms des membres, stockée dans la bibliothèque et connue sous le nom de "directory", peut être fournie à l'usager à tout moment (commande D de YETI).

Vues de l'intérieur, les bibliothèques-disque sont des fichiers, formés de blocs de longueur uniforme 512 octets. Le nom choisi pour le texte détermine - à l'aide d'une fonction de hashcode - son propre emplacement dans le directory (le directory est un morceau de disque réservé d'avance, et l'écriture à l'intérieur n'est pas séquentielle), et par suite l'endroit où l'on va commencer à écrire le texte - cet endroit se situe toujours au début d'un bloc.

Lorsque le premier bloc choisi est plein, on recherche séquentiellement le prochain bloc vide, etc... Les différents blocs qui constituent un "membre" sont chaînés entre eux par leurs quatre derniers octets - qui contiennent l'adresse du bloc suivant s'il en existe un, sinon 0000 -.

Le remplissage des blocs n'est pas effectué directement, en recopiant le texte introduit tel quel ; chaque ligne-écran (égale à une image-carte, ie 80 octets) ou chaque carte est d'abord compactée sous le format suivant :

- Chaque chaîne de deux blancs ou plus est remplacée par un octet indiquant sa longueur (en octets).
- Chaque chaîne composée de caractères non blancs et de blancs isolés est gardée telle quelle et précédée d'un octet contenant sa longueur (toujours en octets).
- Enfin l'enregistrement compacté complet est précédé de cinq octets ; le premier indique sa longueur d'origine (avant compression ; elle est fixée à 80 dans YETI, mais ce n'est pas une obligation) ; le deuxième est utilisé par le système, les troisième et quatrième donnent les numéros de versions d'entrée et de sortie de l'enregistrement (voir ci-dessous paragraphe 1-5), et le cinquième donne sa longueur sous forme compactée (ce cinquième octet compris).

Exemple :

Soit la ligne ("b" désigne un blanc)

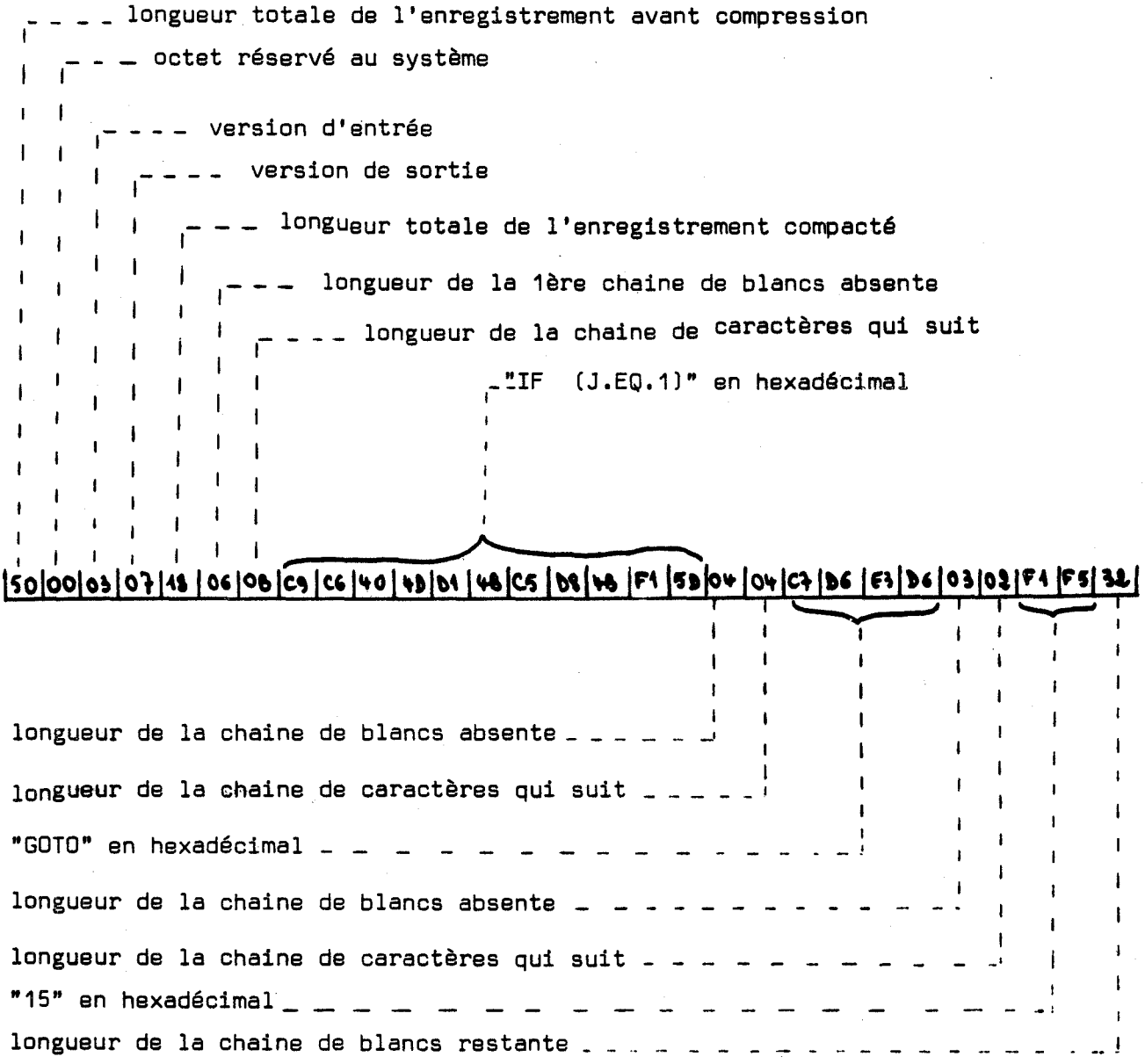
bbbbbbIFb(J.EQ.1)bbbbGOTObbb15bb.....b

blancs jusqu'à la fin

(soit 50)

de numéro de version d'entrée 3 et de numéro de version de sortie 7.

Elle devient : (sous forme hexadécimale):



Ces enregistrements compactés sont écrits séquentiellement, mais ne peuvent être à cheval sur deux blocs (s'il ne reste pas assez de place dans le bloc en cours pour écrire l'enregistrement à venir, on passe au bloc suivant).

Le mode d'accès aux bibliothèques-disque est dit partitionné, car on va directement à un membre (son nom est recherché dans le directory à l'aide de la fonction de hashcode), mais, à l'intérieur du membre, l'accès est séquentiel.

(1-3) Structure de l'espace de travail de YETI :

Cet espace de travail, BITRAV, a été placé sur disque, comme dit plus haut, pour raisons de sécurité. On avait le choix entre un certain nombre de structures ; il a finalement été constitué comme une bibliothèque-disque. En effet, il était nécessaire de le stocker sous forme compactée - pour d'évidentes raisons d'encombrement -. Or, d'une part, le compactage type bibliothèque est très avantageux, surtout lorsque les lignes comportent beaucoup de blancs - ce qui est le cas pour l'essentiel des textes par YETI.
D'autre part, ces textes à traiter doivent se trouver sur des bibliothèques disque donc déjà compactés, ou à compacter. Les recopiages variés sont donc beaucoup plus rapides si le texte en espace de travail a déjà la forme voulue.
Cette structure est totalement transparente à l'utilisateur, pour que BITRAV est un endroit où il dépose son texte, sans plus de précisions.

(1-4) Fonction de l'espace de travail Organisation du texte à l'intérieur :

Le texte est tout naturellement organisé en lignes correspondant aux lignes de programmes. Ces lignes sont numérotées lors de la mise en espace de travail du texte à traiter. La numérotation est dynamique; elle est refaite à chaque modification. Ainsi qu'on l'a vu au chapitre 2, ceci est à la fois un avantage et un inconvénient :

Un avantage, car les numéros portés par les lignes du texte en espace de travail correspondent effectivement à leur emplacement dans ce texte.

Un inconvénient, car l'usager peut avoir des difficultés à suivre une telle numérotation.

Pour pallier cet ennui, il a été prévu deux choses :

1) un ligne peut être désignée de diverses manières : soit par son numéro réel, soit par des symboles spéciaux (. pour la prochaine ligne à lire, $ pour la dernière ligne du texte), soit par une chaîne de caractères qu'elle contient.

2) Si l'on désire connaître l'emplacement d'une ligne, sans plus, la commande N permet de demander son numéro.

Afin de ne pas rendre trop rigide l'organisation en lignes du texte, il existe une commande, M, qui permet d'agir à l'intérieur d'une ligne : modifier une chaîne de caractères existante. - l'effacer, la remplacer par une autre non nécessairement de même longueur - ou insérer une nouvelle chaîne à un endroit donné.

Cette ligne modifiée, et celles dont on demande l'insertion et qui ne sont pas encore insérées, sont les seules qui risquent d'être perdues en cas d'incident quelconque, puisque ce sont les seules qui existent en mémoire centrale sans avoir de copie sur disque.

(1-5) Conservation définitive du texte - Notion de version :

Il est souvent intéressant, pour un programme ou un texte quelconque, de pouvoir conserver diverses formes, soit en cours de mise au point - on peut ainsi abandonner une modification qui ne convient pas -, soit après - on peut alors conserver différentes versions également opérationnelles, pour traiter des problèmes voisins.

Il existe diverses méthodes pour cela, ici, en fait, pour garder trace des modifications apportées au texte initial ([5]) . Cela peut être réalisé, dans les grandes lignes, selon deux schémas :

1. Soit considère que l'état du texte entier, à un moment donné, constitue une version ; puis, après un certain nombre de changements partiels, on obtient un nouvel état du texte entier, qui est considéré comme la version suivante, etc.. Dans ce cas, un éventuel retour en arrière se fait en termes de version : on revient à la version numéro n du texte.

2. Soit on considère chaque modification partielle du texte comme indépendante, et le retour en arrière se fait en annulant les modifications voulues (ce en ne respectant pas nécessairement l'ordre chronologique dans lequel elles sont apparues).

La deuxième méthode est celle qui convient le mieux à un texte libre , où les changements faits en même temps : modification de la syntaxe d'une phrase, remplacement d'un mot par un autre, etc... sont assez souvent sans lien logique entre eux.

La première méthode, par contre, s'adapte nettement mieux aux programmes ; les changements sont généralement du type : introduire une nouvelle variable, modifier un algorithme, et les altérations correspondantes du texte sont logiquement reliées entre elles. Dans ce cas, on a intérêt à considérer comme distincts seulement les deux états extrêmes du programme - par exemple, avant l'introduction de la variable et après -, sans tenir compte des états intermédiaires, qui correspondent très problablement à un programme logiquement faux.

C'est cette première méthode qui a été retenue pour YETI.

La mise en place des diverses versions a été faite de façon très simple :

On considère qu'il y a nouvelle version à chaque fois que l'utilisateur demande une sauvegarde de son texte sur sa propre bibliothèque.

La première version introduite du texte porte le numéro 1, et on incrémente de 1 à chaque nouvelle version - sauf avis contraire de l'utilisateur, qui peut numéroter sa nouvelle version comme il le désire, sous réserve de ne pas utiliser un numéro déjà existant -. Il est possible de conserver jusqu'à 99 versions simultanées du même texte. Ceci n'est pas trop limitatif, d'autant plus que l'usager peut supprimer les versions qui ne l'intéressent plus (soit tout supprimer à partir d'une version donnée - commande X - soit trier parmi les versions existantes celles qui lui conviennent - commande VS ; cette commande fait également la renumérotation des versions restantes, pour clarifier la situation).

Dans le texte enregistré, les versions sont indiquées par deux numéros placés devant chaque ligne de texte (troisième et quatrième octet ajoutés devant l'enregistrement, voir § 1.2). Le premier indique le numéro de la version à laquelle cette ligne est apparue, le deuxième le numéro de la version à laquelle elle a été annulée - si la ligne est toujours valable, le numéro est 0 -.

Cette organisation n'est pas perçue par l'utilisateur, qui demande simplement, soit une copie en espace de travail d'une version donnée de son texte, soit une copie de l'espace de travail comme nouvelle version de son texte, et ne peut par conséquent avoir accès qu'à une version à la fois.

Lorsqu'on choisit cette notion de version - état du texte entier à un moment donné -, on se heurte rapidement à un petit problème : la récupération partielle seulement d'une version antérieure. S'il s'agit de supprimer des cartes, aucun ennui, mais s'il faut réactiver des cartes que l'on a supprimées, les choses se compliquent.

Ce problème peut être résolu en introduisant des octets supplémentaires devant l'enregistrement : version d'entrée, version de 1ère sortie, version de 2ème entrée, version de 2ème sortie. Cela présente deux inconvénients :

- la lourdeur du procédé : il y a beaucoup de stockage à faire, les contrôles à effectuer lors d'un recopiage s'allongent et se compliquent, etc...- tout cela pour un résultat médiocre, car il faut de toute façon limiter le nombre d'entrées et de sorties d'une carte.

- un inconvénient particulier à YETI : l'incompatibilité avec l'éditeur de texte non conversationnel UPDATE, et par suite avec toutes les bibliothèques créées avec cet éditeur.

Le problème a finalement été résolu par un moyen détourné : la commande B.
On a prévu, dans l'espace de travail, un "buffer" où l'utilisateur peut écrire (ce buffer est en fait un membre de bibliothèque ; il est créélorsque l'usager le demande, et est d'une longueur variable - sa limite est la dimension de l'espace de travail, moins la place occupée par le texte en cours de modification, ce qui représente un nombre respectable de lignes (plusieurs milliers) .

La commande B permet de placer dans ce buffer un morceau d'une version donnée d'un texte ; on peut ensuite le replacer où l'on veut - ailleurs dans la même version, dans une autre version d'un même texte, ou dans un autre texte -, simplement en utilisant à nouveau la lettre B lors d'une commande d'insertion (voir chapitre 5 - Commandes I et A) ; l'opération peut être répétée un certain nombre de fois car recopier le buffer ne l'efface pas.

§ 2 - *INTERFACE AVEC L'UTILISATEUR* :

Ce paragraphe recouvre l'ensemble des messages écrans envoyés par YETI.
Lorsqu'un usager a demandé une session de YETI, le premier message qu'il reçoit provient du système, qui annonce qu'il vient d'attacher l'écran ; l'utilisateur sait alors que le programme va effectivement commencer dans peu de temps.
Dès que YETI est activé par le système, il envoie à son tour un message, pour signaler qu'il est en service et attend une première commande.
A partir de ce moment-là, l'utilisateur a à sa disposition un certain nombre de commandes (actuellement 20) dans un langage simplifié (voir chapitre 5 : commandes de YETI).
A la réception de chaque commande l'éditeur commence par contrôler la syntaxe. Si elle est incorrecte, il répond : "erreur dans la commande ..." (les points de suspension remplacent le nom de la commande). Si l'utilisateur ne voit pas l'erreur, il peut utiliser la commande "?", qui lui donne la syntaxe correcte de la commande qu'il désire.
Ensuite, l'éditeur recherche les erreurs logiques décelables au premier abord : numéro de ligne inexistant, chaîne de caractères introuvable, etc... Cela provoque également un message "erreur dans la commande...", mais accompagné d'un autre précisant l'erreur.

Exemples : 1) P 83 demande l'impression de la ligne 83 ; si le texte en espace de travail n'a que 80 lignes, on obtiendra en réponse :

```
ERREUR DANS LA COMMANDE P
LE NUMERO DE LIGNE DEMANDE EST INEXISTANT
```

2) M /ALPHA/,/BETA/,/GAMMA/ demande de remplacer la chaîne BETA par la chaîne GAMMA dans la ligne contenant ALPHA.

Si la chaîne ALPHA n'existe pas, on lira :

ERREUR DANS LA COMMANDE M
LA CHAINE DE CARACTERES SUIVANTE EST INTROUVABLE :
ALPHA

Si la chaîne ALPHA existe, par exemple dans la ligne :

ALPHA = TABLO(K)

qui ne contient pas BETA, on aura :

ERREUR DANS LA COMMANDE M
LA CHAINE : BETA
EST INTROUVABLE SUR LA LIGNE
ALPHA = TABLO(K)

Dans ces cas-là, l'utilisateur peut se servir de la commande P (impression d'un nombre quelconque de lignes) ou N (impression de numéro de ligne) pour vérifier ce qui se passe.

Lorsque la ligne envoyée par l'utilisateur a subi victorieusement ces deux contrôles, l'éditeur passe à l'exécution du travail demandé. Il était prévu, initialement, d'envoyer un message à ce moment là ("commande reçue", par exemple), puis un autre lorsque le travail aurait été effectué. Cela s'est avéré inutile car l'attente est le plus souvent pratiquement nulle (voir chapitre 6 : Mesures - et l'annexe).

Le seul cas où un message arrive immédiatement est celui où une erreur est détectée au début de l'exécution : par exemple, demande de copie d'un membre inexistant d'une bibliothèque. Si le nom donné correspond aux caractéristiques générales des noms de membres, il n'est possible de repérer l'erreur qu'au moment où l'on essaie effectivement de lire ce membre.
Le message d'erreur est alors explicite :
"Il n'existe pas de membre de ce nom dans la bibliothèque 1"
"Il n'y a pas de bibliothèque ouverte sous ce numéro" , etc...
(pour les numéros affectés aux bibliothèques, voir chapitre 5)

Pour l'usager, il n'y a pas de différence sensible entre les délais d'attente des divers messagers d'erreurs. Ils paraissent tous immédiats. (Voir temps exacts au chapitre 6 et dans l'annexe).

Si tout se passe correctement, l'éditeur envoie une réponse explicite annonçant qu'il a effectué la commande.

Exemple : On veut copier en espace de travail la dernière version, numérotée 3, du membre MEMBER de la bibliothèque 1.

On écrit : G MEMBER , 3, 1

ou G MEMBER (en utilisant les options par défaut)

L'éditeur répond :

LA VERSION 3
DU MEMBRE MEMBER
DE LA BIBLIOTHEQUE 1
EST MISE EN ESPACE DE TRAVAIL

Ce genre de message permet à l'utilisateur de savoir exactement ce que l'éditeur vient de faire. Cela lui permet de réaliser si ce qu'il a demandé correspond bien à ce qu'il voulait, ie s'il a bien compris ce que faisait la commande (cela pour les débutants); et s'il n'a pas commis une erreur indiscernable pour l'éditeur - Par exemple : p14 au lieu de P24, W MEMBER au lieu de G MEMBER ; ces instructions sont syntactiquement correctes ; si elles correspondent à quelque chose de réalisable, l'éditeur ne peut rien déceler d'anormal. A ce sujet, un nombre important de contrôles ont été introduits dans l'éditeur pour limiter ces erreurs. (Voir chapitre 6, au sujet précisément de G MEMBER et W MEMBER).

Enfin, la réponse à certaines commandes - impression, insertion, directory - peut nécessiter plus d'un écran, et l'usager ne sait pas exactement combien. Aussi, à tout message de réponse est associée la question "commande suivante ?". Dans le cas des réponses longues, cette question n'apparait que sur le dernier morceau de la réponse. Sa présence signifie implicitement que la réponse demandée est achevée.

Il reste à citer une dernière catégorie de messages, ceux correspondant aux erreurs détectées par le système. On distingue deux catégories :

* La première recouvre toutes les erreurs sur bibliothèques ou sur disques, du type :
- bibliothèque pleine
- disque plein
- impossibilité de lecture
- impossibilité d'écriture
- erreur de lecture ou d'écriture

  etc...

L'éditeur reprend ces erreurs à son compte et les traite : il les explique à l'utilisateur, avec éventuellement la méthode pour les corriger.

* La deuxième contient toutes les erreurs système générales ; elles sont alors simplement signalées par l'éditeur.

Dans les deux cas, on dit également à l'usager s'il peut ou non continuer à utiliser YETI.

En résumé, l'interface YETI-utilisateur a été conçu de façon à :

1°) renseigner au maximum par les réponses spontanées de l'éditeur : messages d'erreur précis, description de ce qui vient d'être fait, indications pour corriger une erreur d'origine extérieure.

2°) offrir à l'usager la possibilité de poser des questions s'il ne sait plus très bien où il en est : commandes N, ?, D, P.

3°) éviter le plus possible une interruption brutale en cas d'erreur-système.

CHAPITRE V

COMMANDES DE YETI

Ce chapitre donne, pour chaque commande de YETI, une description syntaxique précise et un bref résumé de fonctionnement. On peut ainsi se faire une idée du "produit fini" YETI, tel qu'il se présente à l'utilisateur habituel.

La métalangue utilisée pour la description syntaxique est une forme dérivée de celle de Backus-Naur. Elle se présente de la façon suivante :

. les éléments entre crochets <...> doivent être remplacés par l'utilisateur selon des règles précisées pour chaque commande.
. la barre verticale | sépare les divers choix possibles pour un seul élément.
. les crochets ⌊ et ⌋ , encadrant une suite de choix possibles pour un élément, indiquent que le recopiage de l'un de ces choix est obligatoire.
. les crochets ⌈ et ⌉ , indiquent que le recopiage de ce qu'ils encadrent est optionnel.
. les mêmes crochets ⌈ et ⌉ suivis d'un numéro n en haut et à droite du crochet fermé indiquent que ce qu'ils encadrent peut être recopié de 0 à n fois.
. la notation <...> ::= permet de décrire rigoureusement la façon de remplacer l'élément entre crochets <...> lorsque ce n'est pas un simple identificateur.

§ 1 - FONCTION

YETI permet d'agir sur des bibliothèques à 4 niveaux différents.

- Niveau bibliothèque
  - Ouverture (O)
  - Fermeture (C)
  - Suppression (ZZ)
  - Impression du directory (D)

- Niveau membre (source seulement)
  - Création (W)
  - Lecture d'une version (G)
  - Ecriture d'une version (W)
  - Suppression totale ou partielle du membre (X)
  - Sélection de certaines versions du membre (VS)
  - Changement de nom (R)
  - Copie d'un fichier séquentiel sur un membre (F)
  - Copie d'un membre sur un fichier séquentiel (E)

- Niveau ligne
  - Insertion d'une ou plusieurs lignes avant une ligne (I)
  - Insertion d'une ou plusieurs lignes après une ligne (A)
  - Suppression d'une ou plusieurs lignes (S)
  - Impression d'une ou plusieurs lignes (P)
  - "Mise en conserve" d'une ou plusieurs lignes (B)
  - Impression de numéro de ligne (N)

- Niveau caractère à l'intérieur d'une ligne

  Remplacer une chaîne de caractères (éventuellement vide) par une autre chaîne de caractères (éventuellement vide) à l'intérieur d'une ligne (M).

§ 2 - UTILISATION

2.1. - Mise en oeuvre :

Pour lancer YETI, on doit envoyer par un écran ou par le lecteur de cartes : PROC$_{\Lambda}$YETI$_{n}$
où n est le numéro de l'écran sur lequel on veut travailler, suivi d'une carte DCB par bibliothèque et par le fichier que l'on veut utiliser.
Puis de $\begin{matrix}1\\7\end{matrix}$ JS (sur le lecteur de cartes) ou de # JS
END (sur écran)

Format des cartes DCB :

DCB$_{\Lambda}$<dcbname>,DV=D2,VOL=<nom>[:n],DSNAME=<nolib>,[STATUS=OLD|NEW|SHR
(CAT NOCAT)]

où : <dcbname> est un nom quelconque de 8 caractères alphanumériques au plus,
<nolib> est le nom de la bibliothèque.
<nom> est le nom du pack sur lequel elle se trouve.
n est le nombre de cylindres prévu pour la bibliothèque (à préciser seulement si STATUS=NEW)

On peut également ajouter le paramètre DIR=h, qui précise le nombre de blocs de directory demandés (1 bloc = 25 noms), toujours dans le cas où STATUS=NEW.

Exemple : On fait le lancement à partir d'un écran (n'importe lequel)
On veut travailler sur l'écran 2 ; on compte utiliser les bibliothèques existantes BIBI du pack 3 et BIB2 du pack 2, et créer la bibliothèque BINIOU sur le pack 2 (3 cylindres prévus) :

```
# PRØC_A YETI2
# DCB TRUC,DV=D2,VØL=3,DSNAME=BIBI,STATUS=ØLD(NØCAT)
# DCB CHØSE,DV=D2,VØL=2,DSNAME=BIBI2,STATUS=ØLD(NØCAT)
# DCB MACHIN,DV=D2,VØL=2:3,DSNAME=BINIØU,STATUS=NEW(NØCAT)
# JS
# END
```

2.2. - Déroulement des opérations :

- Dès qu'il est activé, YETI envoie un message sur écran annonçant qu'il attend la première commande.
On inscrit cette première commande, on transmet et on attend la réponse de l'éditeur. Cette réponse comporte 3 éléments :

. un compte rendu de ce qui vient d'être fait, ou, le cas échéant, un message d'erreur.
. la question "commande suivante ?", placée sur la dernière ligne de l'écran (voir exception plus bas).
. une flèche indiquant l'endroit où l'on doit écrire la commande suivante. On l'inscrit, etc...

Deux exceptions :

1) Lorsqu'on demande une impression (de texte ou de directory) de plus de 13 lignes: Les 13 premières lignes s'inscrivent ; lorsqu'on les a suffisamment vues, on appuie sur le bouton "TRANSMIT" pour faire apparaître les suivantes, etc... Sur le dernier morceau de texte apparaît normalement la question "commande suivante ?"

2) Lorsqu'on demande une insertion de 14 lignes ou plus : on inscrit les 14 premières lignes à insérer immédiatement à la suite de la commande d'insertion et on transmet.
L'éditeur demande "Suite des lignes à insérer ?" et envoie une flèche au début de la 2ème ligne. On inscrit la suite du texte à partir de cette deuxième ligne, on transmet, etc...
La fin d'insertion doit être signalée par une ligne spéciale (voir commandes A et I, § 5). Sa détection provoque une réponse de YETI, de la même forme qu'une réponse habituelle (voir début du §).

- En résumé : Il ne faut écrire quelque chose que lorsque c'est expressément demandé : "suite des lignes à insérer ?", "commande suivante ?", "vous pouvez inscrire votre première commande". Dans tous les autres cas, il suffit d'appuyer sur "TRANSMIT" et d'attendre la suite.

2.3. - Fin d'utilisation :

La commande TERMINE (voir § 3) achève l'exécution de YETI et libère l'écran.

§ 3 - ACTIONS AU NIVEAU BIBLIOTHEQUE

3.1. - Ouverture d'une bibliothèque :

Pour avoir accès au contenu d'une bibliothèque, il est nécessaire de l'ouvrir par la commande :

O<dcbname> [<dcbid>]

où : <dcbname> est celui indiqué sur la carte DCB correspondant à la bibliothèque

<dcbid> est le chiffre 1 ou 2. Option par défaut : 1

Le numéro <dcbid> choisi est celui qui va être attribué à la bibliothèque jusqu'à sa fermeture ; il sert à la désigner dans toutes les autres commandes de YETI.

Cette commande crée la bibliothèque si elle n'existe pas.

3.2. - Fermeture d'une bibliothèque :

Lorsqu'on n'a plus besoin d'une bibliothèque, on peut la refermer.
Cela se fait par :

C [<dcbid>]

où <dcbid> est le numéro attribué lors de son ouverture à la bibliothèque que l'on veut fermer. Option par défaut : 1.

Il n'est pas obligatoire de fermer une bibliothèque. La commande TERMINE (voir 3.6) se charge de fermer toutes les bibliothèques ouvertes en fin de travail.

3.3. - Utilisation de plusieurs bibliothèques :

Etant donné qu'il existe deux numéros distincts, on peut ouvrir simultanément au plus deux bibliothèques. Cependant, lorsqu'on ferme une bibliothèque (commande C), elle libère son numéro, et l'on peut en ouvrir une autre sous ce numéro là. Le nombre de bibliothèques ouvertes successivement de cette façon n'est pas limité.

3.4. - Suppression de la bibliothèque :

ZZ [<dcbid>]

où <dcbid> est le numéro attribué lors de son ouverture à la bibliothèque que l'on veut supprimer. Option par défaut : 1

Ce numéro est bien entendu libéré lorsque la commande a été exécutée.

3.5. - Listing du directory d'une bibliothèque :

D [<dcbid>]

option par défaut :

<dcbid> = 1

Cette commande liste les noms de tous les membres contenus dans la bibliothèque demandée, à raison de 5 par ligne, par 13 lignes à la fois, situées sur les lignes 3 à 15 de l'écran, et en plaçant devant chaque nom la lettre S, O ou C suivant que le membre est en source, code objet ou core image.

3.6. - Fin du travail :

TERMINE

Cette commande ferme toutes les bibliothèques ouvertes et termine un traitement . Elle est obligatoire.

§ 4 - ACTIONS AU NIVEAU MEMBRE

A : Actions employées seules

4.1. - Suppression d'un membre en totalité ou à partir d'une certaine version :

X <memberid>[,[<versnb>],[<dcbid>]]

<memberid> : identificateur d'un membre de bibliothèque : au maximum 8 caractères alphanumériques.

<versnb> : numéro de version du membre ; c'est un nombre compris entre 1 et 99.

Cette commande permet de supprimer toutes les versions du membre <memberid> de la bibliothèque de numéro <dcbid> à partir du numéro de version <versnb>, ce numéro compris ; cela implique la suppression totale du membre si ce numéro est 1.

Option par défaut : <versnb> = 1 (suppression totale)
<dcbid> = 1

4.2. - Sélection de certaines versions d'un membre :

VS [<dcbid>,] <memberid>,<versnb>[,<versnb>][32]

Option par défaut : <dcbid>=1
<memberid> est le nom du membre intéressé par la modification,
<dcbid> le numéro de la bibliothèque où il se trouve, les <versnb> successifs représentent les versions à garder classées par numéros croissants.

Cette commande permet de ne conserver que la ou les versions demandées du membre, en les renumérotant (par leur numéro d'ordre dans la liste donnée).

4.3. - Changement du nom d'un membre :

R[<dcbid>,]<memberid1>,<memberid 2>

Cette commande donne au membre <memberid1>, de la bibliothèque n° <dcbid>, le nouveau nom <memberid2>.
Option par défaut : <dcbid>=1.

4.4. - Copie d'un membre sur un fichier séquentiel :

E[#DCB^<dcbname fich>,] <member id>[,[<versnb>][,<dcbid>]]

(blanc significatif)

où <dcbname>fich> est le dcbname (voir 2.1) indiqué sur la carte DCB correspondant au fichier.

Cette commande copie la version <versnb> du membre <member id> de la bibliothèque de numéro <dcbid> sur le fichier séquentiel désigné par la carte DCB portant le dcbname : <dcbname fich>.

Application courante : Copier un membre constituant un job sur le fichier séquentiel SYSGIN ; ce fichier est un "System input" intérieur à la machine, ie il est considéré de la même façon que le lecteur de cartes, avec priorité sur celui-ci. Cela permet de lancer l'exécution d'un travail à partir de l'éditeur.

Options par défaut : <versnb> : dernière version enregistrée
<debid> : 1
<fichier>: SYSGIN

4.5. - Copie d'un fichier séquentiel sur un membre :

F # DCB <debname fich>,<member id>[,[<versnb>][,<dcbid>]]

Cette commande copie le fichier désigné sur la carte de dcbname <dcbname fich> sur le membre <memberid> de la bibliothèque de numéro <dcbid> ; ce fichier constitue la version <versnb> du membre.

Application courante : Recopier un fichier contenant les résultats d'un travail.

Options défaut : <versnb> : dernière version enregistrée + 1
<dcbid> : 1

B : Actions employées avant ou après une suite d'actions au niveau ligne

4.6. - Création d'un texte :

- Si B est absent : les lignes qui suivent celle-là, jusqu'à détection d'une ligne ayant % en 1ère colonne et des blancs ensuite, vont être placées en espace de travail, sous un nom provisoire, en attendant d'être éventuellement enregistrées comme première version d'un membre d'une bibliothèque.

Si le texte à créer comporte 14 lignes ou plus, agir comme décrit en 2.2., exception n°2.

Bien entendu la ligne indiquant la fin d'insertion n'est pas enregistrée dans le texte.

- Si B est présent : Le contenu de BUFF (voir 5. 7, commande B) est placé en espace de travail, sous un nom provisoire, dans le même but que précédemment.

Remarque : Le texte ainsi introduit peut être modifié par des actions au niveau ligne (tout comme un texte mis en espace de travail par la commande G décrite ci-dessous en 4.7).

Ce texte efface tout texte introduit précédemment en espace de travail par une commande G ou une autre commande A - création.

4.7. - Mise en espace de travail d'une version d'un membre :

G <memberid>[,[<versnb>][,<dcbid>]]

Cette commande va lire dans la bibliothèque de n° <dcbid>, la version <versnb> du membre <member id> et la transporte dans l'espace de travail. Elle efface tout autre texte précédemment introduit dans l'espace de travail par une commande G ou A - création.
Option par défaut : <dcbid> = 1
<versnb>= n° de la dernière version enregistrée.

4.8. - Enregistrement d'une version d'un membre (éventuellement la première)

W <memberid>[,[<versnb>][,<dcbid>]

Cas où <versnb> est ≠ 1 :

Cette commande prend dans l'espace de travail la version qui vient d'être construite et va l'écrire sur le membre indiqué avec le numéro de version <versnb> ; le texte en espace de travail est détruit.

Cas où <versnb> est égal à 1 :

Il y a création d'un membre sous le nom demandé et insertion du texte qui est en espace de travail sous le numéro de version 1, s'il n'existe aucun membre source dans la même bibliothèque qui porte déjà ce nom. S'il en existe un, un message d'erreur est écrit, la commande est ignorée et le texte en espace de travail n'est pas effacé.

Option par défaut : . <dcbid>=1
. <versnb>=numéro de la dernière version enregistrée + 1.

Remarque :

Si <versnb> est omis, le membre est supposé exister (voir option par défaut) ; s'il n'existe pas, un message d'erreur est écrit, la commande est ignorée et le texte en espace de travail n'est pas effacé.

4.9. - Remarque à propos de G et W :

On peut, dans ces commandes, choisir le numéro de la version que l'on veut lire ou écrire. L'utilisation la plus courante est celle qui correspond aux options par défaut : dernière version pour G, dernière version + 1 pour W. Il existe d'autres possibilités, que l'usager découvrira au fur et à mesure de ses besoins. Cependant, il est absolument impossible, lorsqu'on veut créer une nouvelle version, d'utiliser une autre version que la dernière pour faire les modifications. (cela tient à la structure des bibliothèques). Le résultat obtenu est complètement aberrant.

De même, il est impossible d'enregistrer sur un membre déjà existant une version d'un autre membre. Par contre, il est tout à fait possible de créer un membre à partir d'une version d'un autre ; cette version sera la version 1 du nouveau membre.

Exemples : Soit MEM1 un membre dont la dernière version enregistrée est 6 ; soit MEM2 un membre existant (dernière version : 4), et MEM3 un identificateur non encore attribué.

G MEM1, 3 N MEM1, 7	}	résultat aberrant	
G MEM1, 5 W MEM2, 5	ou	G MEM1, 6 W MEM2, 5	} interdit (message d'erreur)
G MEM1, 2 W MEM3, 1		G MEM1, 6 W MEM3, 1	} correct

§ 5 - COMMANDES AU NIVEAU LIGNE OU A L'INTERIEUR D'UNE LIGNE :

5.1. - Ces commandes permettent d'agir sur le texte qui est en espace de travail :

Il n'y a qu'un seul texte à la fois dans cet espace, celui apporté par la dernière commande G ou A-création.
(tout texte antérieur a été détruit, soit par une commande W qui l'a écrit ailleurs, soit par une commande G ou A-création qui a mis en bibliothèque de travail un autre texte à sa place).

Toute notion de ligne ou de numéro de ligne se rapporte à ce texte.

5.2. - Numérotation :

5.2.1. - Lorsque le texte est mis en espace de travail, ses lignes sont numérotées 1, 2, 3, ...etc...

A chaque insertion (commandes A et I) ou suppression (commande S), il est renuméroté, en tenant compte des lignes qui viennent d'apparaître ou de disparaître.

Afin de s'y retrouver dans cette numérotation variable, il est possible :

- de demander le numéro d'une ligne (commande N)
- de désigner une ligne autrement que par son numéro : en général, une ligne peut être désignée par <line expr> , où :

<line expr> ::=<line expr élémentaire>|<line expr élémentaire>+<nb>
|<line expr élémentaire>-<nb>
<line expr élémentaire>::=.|<nombre>|$|/<caractères>/

5.2.2. - Description de <line expr> et <line expr élémentaire> :

<nombre>.....est le numéro effectif de la ligne (nombre entier positif)

$ .....désigne la dernière ligne du texte.

/<caractères>/. est une chaîne de caractères entre slashes ; on peut repérer une ligne quelconque en donnant, entre 2 slashes, une chaîne de caractères qu'elle contient (chaîne = suite de caractères contigus) à condition :

- soit qu'elle soit la seule à contenir cette chaîne
- soit qu'elle soit la première à la contenir dans l'ordre de parcours (voir ci-dessous (5.2.3) : mode de la recherche de la chaîne).

. .... désigne la ligne courante, c'est-à-dire :

- après la commande P : la ligne qui suit la dernière ligne imprimée (voir § 5.6)
- après la commande N : (voir §5.8)
  soit <line expr> contient une chaîne de caractères, et la ligne courante est la ligne qui suit celle contenant cette chaîne.
  soit <line expr> ne contient pas de chaîne de caractères, et la ligne courante reste inchangée par rapport à la commande précédente.
- après les commandes I, S, A, M, VS, G, W : ligne 1
- après toutes les autres commandes : inchangée par rapport à la commande précédente.

<nb>..... est un nombre entier positif.

Exemple de <line expr> : Soit en espace de travail le texte suivant :

```
        SUBRØUTINE ADD (A, B, M)
        DIMENSIØN A(M),B(M),C(M)
        DØ 10 I=1,M
        C(I)=A(I)+B(I)
10      CØNTINUE
        RETURN
        END
```

La ligne C(I) = A(I) + B(I) peut être désignée par :

```
        4
        $-3
       /C(I)/
       /=A/
       /CØNTI/-1
       /DIMENSIØN/+2
        1+3
```

Si l'on vient de faire imprimer les trois premières lignes (commande P) :

```
        .+1
```

Si l'on vient juste de mettre le texte en espace de travail (commande G) :

```
        .+3
```

5.2.3. - Procédure exécutée lorsque l'éditeur rencontre une <line expr>

a) <line expr élémentaire> est mise sous forme d'un nombre si elle n'y est pas déjà. C'est-à-dire :
S'il s'agit de . ou S, ils sont remplacés par leur valeur.
S'il s'agit de / <caractères>/, la chaîne de caractères est recherchée séquentiellement à travers tout le texte, en commençant à la ligne courante et en finissant à la ligne courante (on considère que la 1ère ligne suit la dernière).
On arrête le parcours dès que la chaîne est trouvée dans une ligne. Le numéro de cette ligne - la première dans l'ordre de parcours qui contient la chaîne demandée - est mis dans <line expr élémentaire>. Si la chaîne de caractères que l'on veut rechercher contient des slashes, il faut les doubler afin qu'ils ne soient pas confondus avec le slash de fin de chaîne.

b) <line expr> est calculée

Reprise de l'exemple précédent :

Si l'on vient juste de mettre le texte en espace de travail, la ligne courante est 1 : /10/ désignera la ligne N°3

```
DO 10 I=1,M
```

Si l'on vient de faire imprimer les 3 premières lignes, la ligne courante est 4 : /10/ désignera la ligne 5 :

```
10 CONTINUE
```

5.3. - Insertion d'une ou plusieurs lignes avant une ligne :

```
I [B]< line expr>
```

L'insertion avant la ligne portant le N° <line expr> est faite. Elle peut se faire de deux façons suivant la présence ou l'absence de B.

a) B est absent : il y a insertion des lignes qui suivent la ligne portant la commande I, jusqu'à détection d'une ligne qui contient % en premier caractère et des blancs ensuite (cette ligne indiquant la fin de texte à insérer n'est pas enregistrée). Pour insérer 14 lignes ou plus, voir 2.2 exception 2.

b) B est présent : il y a insertion du contenu de BUFF (voir commande B, § 5.7)

5.4. - Insertion d'une ou plusieurs lignes après une ligne :

A [B]<ligne expr>

Le processus est exactement le même que dans (5.3) mis à part le fait que l'insertion a lieu après la ligne adressée et non avant.

5.5. - Suppression d'une ou plusieurs lignes :

S <ligne expr1>[,<line expr2>]

Cette commande supprime les lignes de <ligne expr1> à <ligne expr2>, bornes comprises.

Si <line expr1> est plus grand strictement que <line expr2>, elle supprime de <line expr1> à la fin du texte, puis du début de texte à <line expr2>

Option par défaut : <line expr2> = <line expr1>

5.6. - Impression d'une ou plusieurs lignes :

P[<line expr1>[,<line expr2>]]

Cette commande imprime le texte qui est en espace de travail, de la ligne <line expr1> à la ligne <line expr2>, bornes comprises.

Si <line expr1> est strictement plus grand que <line expr2>, elle imprime le texte de <line expr1> à la fin puis du début à <line expr2>.

L'impression se fait par 13 lignes à la fois, placées sur les lignes 3 à 15 de l'écran. (voir 2.2, exception 1 pour plus de précisions).

Options par défaut : - si <line expr1> est mis, <line expr2> = <line expr1>
- s'il n'y a rien, <line expr1> = 1,
<line expr2> = $, c'est-à-dire impression de tout le texte.

5.7. - "Mise en conserve" d'une ou plusieurs lignes :

B[<line expr1>[,<line expr2>]]

Cette commande recopie, dans un membre de la bibliothèque de travail appelé BUFF, le morceau de texte demandé (éventuellement tout le texte).

Les options par défaut sont les mêmes que précédemment dans (5.6).

Dans cette commande, <line expr1> ne peut pas être strictement supérieur à <line expr2> (restriction pour de simples raisons logiques : voir exemple d'utilisation).

Usage de cette commande : On ne peut utiliser qu'un seul texte à la fois en bibliothèque de travail (le dernier introduit efface systématiquement le précédent). Cependant, il arrive souvent que l'on veuille insérer dans un membre une partie d'un autre membre, ou dans une version d'un membre une partie d'une autre version du même membre. La commande B permet de "mettre de côté" le texte que l'on veut insérer.

C'est ce texte qui est inséré par les commandes A et I lorsque la lettre B est présente.

Exemple d'utilisation :

On avait, en version 4 d'un membre ALPHA, un certain nombre de cartes successives, par exemple 200 à 420, que l'on a supprimées par la suite et que l'on voudrait récupérer, alors qu'on en est à la version 10 du membre, pour les placer après la ligne 25 de cette version. Il faut :

1°) Ouvrir la bibliothèque contenant le membre
(la carte DCB de cette bibliothèque porte le dcbname TRUC)
2°) Mettre la version 4 du membre en espace de travail
3°) Mettre dans BUFF les cartes voulues
4°) Mettre la version 10 du membre en espace de travail
5°) Insérer les cartes à l'endroit voulu
6°) Si l'on ne veut rien modifier d'autre, enregistrer cette nouvelle version sous le numéro 11.

On écrira : (en supposant que l'on ouvre la bibliothèque sous le numéro 1)

```
Ø TRUC,1
G ALPHA,4,1
B 200,420
G ALPHA,10,1
AB 25
W ALPHA,11,1
```

Ou, en utilisant les options par défaut :

```
O TRUC
G ALPHA,4
B 200,420
G ALPHA
AB 25
W ALPHA
```

Exemple d'utilisation 2 :

On a plusieurs subroutines FØRTRAN SUB1, SUB2, SUB3, etc... dans lesquelles on veut ajouter la ligne : CØMMØN BLANC, UN, DEUX, TRØIS

On crée un texte formé de cette ligne, et on le met tout de suite dans BUFF :

```
A
  CØMMØN BLANC, UN, DEUX, TRØIS
%
B
```

On n'a plus, ensuite, qu'à insérer BUFF à l'endroit voulu dans chaque subroutine. Par exemple :

```
G SUBI
AB 6
W SUBI
G SUB2
AB 3
W SUB2
G SUB3
IB 8
W SUB3, etc
```

5.8. - Impression du numéro de ligne :

N <line expr>

Cette commande imprime le numéro de la ligne désignée par <line expr>

Utilisations : Savoir quelle ligne on va lire (si <line expr> = .)
Savoir combien le texte a de lignes (<line expr> = $)
Trouver une certaine chaîne de caractères
(<line expr> = /<caractères>/)
etc...

5.9. - Remplacement d'une chaîne de caractères par une autre :

M[D]<line expr>,⌊/<caractères1>/|<nb1>,[<nb2>]⌋/<caractères2>/

- Cette commande permet, dans la ligne numéro <line expr>, de remplacer la chaîne <caractères1> par la chaîne <caractères2>.
La chaîne à remplacer peut être donnée explicitement entre slashes ou repérée par sa position (<nb1> et <nb2> désignent alors le premier et le dernier caractères à remplacer).

Option par défaut : <nb2> = <nb1>

- La modification se fait de deux façons différentes suivant que le D est présent ou non :

a) D absent : C'est une modification type "programme". Les caractères 1 à 72 d'une part, 73 à 80 de l'autre, sont considérés comme deux zones distinctes et une modification de l'une ne touche pas l'autre.

Exemples : 1) On veut remplacer les caractères 20 à 30 par 3 caractères, les caractères 1 à 72 sont réécrits de façon à ne pas laisser de blancs inutiles. Les caractères 73 à 80 ne sont pas modifiés.

2) On veut rajouter 5 caractères après le 50ème : les caractères 51 à 67 sont poussés à droite, les caractères 68 à 72 disparaissent, qu'ils soient blancs ou non. Les caractères 73 à 80 ne bougent pas.

b) D présent : C'est une modification type "données". L'ensemble de la ligne est affecté par la modification : resserrement pour éviter les blancs inutiles ou disparition par la droite des caractères excédentaires.

- Pour supprimer une chaîne de caractères : il suffit de donner comme chaîne de remplacement une chaîne vide.
- Pour insérer une chaîne de caractères : il faut supprimer un ou plusieurs caractères de la ligne et les réinsérer avec la nouvelle chaîne (voir exemples ci-dessous pour plus de précision).
- Pour modifier quelque chose dans les colonnes 73 à 80, utiliser la modification de type données MD (plus rapide que M).

- Exemples :

Soit la ligne : IF (J.EQ.3.AND.K.LT.5) GOTO20

On suppose, pour simplifier, que c'est la ligne 13 du texte enregistré et qu'on la désigne explicitement par son numéro.

1) On veut remplacer 20 par 53. On peut écrire indifféremment :

```
M  13,/20/,/53/
MD 13,/20/,/53/
M  13, 34, 35, /53/
```

(si on suppose que 20 est écrit en colonnes 34 et 35)

2) On veut insérer dans la parenthèse : .AND.I.GT.7
On peut, par exemple, supprimer la parenthèse droite et la réinsérer :

M13, /)/,/.AND.I.GT.7)/

On peut aussi supprimer le 5 et le réinsérer :

M13,/5/, /5.AND.I.GT.7/

etc...

Le ou les caractères supprimés puis réinsérés servant à indiquer l'emplacement de l'insertion, puisqu'il n'y a aucun autre moyen de l'indiquer.

3) On veut supprimer .AND.K.LT.5

M13,/.AND.K.LT.5/,//

4) On veut ajouter TOTO après la colonne 72 :

MD13,73,76,/TOTO/

CHAPITRE VI

EVALUATIONS ET MESURES

Les mesures faites sur YETI se divisent en deux groupes : études des performances de l'éditeur, et mesures sur les utilisateurs. Dans les deux cas, à côté de mesures simples, il s'en trouve d'autres assez difficiles à interpréter. Afin de ne pas alourdir la description des résultats, on a exposé à part la méthode utilisée pour les étudier.

§ 1 - ANALYSE DES DONNEES :

(1.1) Pour les mesures sur une variable, qui n'ont besoin d'aucune interprétation, on a donné la moyenne des valeurs obtenues, l'écart-type et l'histogramme car l'écart-type est souvent élevé (mesures dispersées). Pour les mesures sur plusieurs variables, on a retenu au départ deux méthodes d'analyse de données : la méthode des nuées dynamiques, et l'analyse des correspondances.

(1.2) La méthode des nuées dynamiques ([22],[23],[24]) est une méthode de classification automatique - techniques qui consistent à "trouver une partition d'un ensemble fini E tel que chaque objet ressemble plus aux objets intérieurs à son groupe qu'aux objets extérieurs" ([22]).

Elle demande trois données initiales :

- un ensemble fini de "noyaux" autour desquels on va former les groupes ; ces noyaux peuvent être obtenus par tirage aléatoire parmi les données ou en définissant certaines contraintes qu'ils doivent respecter, etc... Ils peuvent être formés d'un ou plusieurs points.
- une fonction f permettant le passage des noyaux à une partition de l'espace E.
- une fonction g permettant la transformation d'une partition en un ensemble fini de noyaux.

La méthode consiste à appliquer f et g alternativement ; sous certaines hypothèses, la convergence est assurée. On peut utiliser cet algorithme, soit seul, soit comme prélude à l'analyse des correspondances lorsque le nombre de données est très important.

(1.3) L'analyse des correspondances ([25],[26]) est parfaitement adaptée à l'étude des tableaux rectangulaires de nombres positifs. Elle détermine les facteurs qui expliquent l'écart entre la distribution observée et l'hypothèse " toutes les variables que l'on a mesurées sont indépendantes". Elle se fait en quatre étapes :

a) Représentation des données dans un espace de dimension convenable :

Soit $K = ((k_{ij})i\epsilon I)j\epsilon J$ la matrice de données fournie à l'origine, I désignant l'ensemble des indices de ligne et J celui des indices de colonne.
On utilise les notations suivantes :

$$k = \sum_{i\epsilon I} \sum_{j\epsilon J} k_{ij} \; ; \; f_{ij} = \frac{k_{ij}}{k} \; ; \; f_{i.} = \sum_{j\epsilon J} f_{ij} \; ; \; f_{.j} = \sum_{i\epsilon I} f_{ij}$$

n=card I ; p=card J ; on suppose $p \leq n$.

Pour que l'analyse soit indépendante de l'effectif du tableau, on remplace l'étude de K par celle de $F = ((f_{ij})i\epsilon I)j\epsilon J$.
L'ensemble des f_{ij} peut être représenté de la façon suivante : A tout i de I, on associe le point de $\mathbb{R}^p$: $\left(\frac{f_{ij}}{f_{i.}}\right) j = 1,2, \ldots p$, auquel on attribue une masse $f_{i.}$.

Remarque : Tous les points ainsi définis appartiennent en fait à l'hyperplan de $\mathbb{R}^p$ d'équation $\sum_{j=1}^{p} x_j = 1$

b) Choix d'une métrique :

On choisit comme distance de deux points :

$$d(x_i, x_{i'}) = \sum_{j=1}^{p} (f_{ij}/f_{i.} - f_{i'j}/f_{i'.})^2 / f_{.j}$$

(distance du χ^2 pour la métrique de centre $f_{.j}$)

Ce choix permet, au cours de l'analyse, de remplacer deux lignes proportionnelles i_1 et i_2 par une seule ligne i, telle que : $f_{ij} = f_{i1j} + f_{i2j}$, $\forall j$, sans modifier les résultats (par définition de la distance, les trois points i_1, i_2 et i se trouvent confondus lorsqu'on les représente dans $\mathbb{R}^p$ suivant la méthode décrite en a)).

c) Projection des données dans un espace de faible dimension :

. Cette projection est destinée à rendre la représentation précédente interprétable. Pour conserver le maximum d'informations sur la forme réelle du nuage de points, on va le projeter sur ses axes principaux d'inertie.

. Il faut d'abord déterminer ces axes : pour ramener le problème à un calcul classique de mécanique, on considère la distance définie ci-dessus comme la distance euclidienne habituelle entre les deux points y_i et $y_{i'}$, de coordonnées respectives $\left(\frac{f_{ij}}{f_{i.}\sqrt{f_{.j}}}\right) j = 1,\ldots,p$ et $\left(\frac{f_{i'j}}{f_{i'.}\sqrt{f_{.j}}}\right) j = 1,\ldots,p$

(la masse du point y_i reste $f_{i.}$, masse du point x_i).

. On fait ensuite un changement d'axes pour centrer le nuage :

Si on note (y_{ij}) j=1,...p les coordonnées de y_i, on pose :

$$z_{ij} = y_{ij} - \sum_{i=1}^{n} f_{i.}\, y_{ij} \quad \forall\, j=1,2,\ldots,p$$

$$\text{Or} \quad \sum_{i=1}^{n} f_{i.} y_{ij} = \sum_{i=1}^{n} f_{i.} \frac{f_{ij}}{f_{i.}\sqrt{f_{.j}}} = \frac{\sum_{i=1}^{n} f_{ij}}{\sqrt{f_{.j}}} = \sqrt{f_{.j}}$$

D'où les nouvelles coordonnées du ième point.

$$\left(\frac{f_{ij}}{f_{i.}\sqrt{f_{.j}}} - \sqrt{f_{.j}}\right) j = 1,2,\ldots,p$$

. On sait alors (résultat de mécanique) que, si V est l'application de dans $\mathbb{R}^p$ définie par :

$$\forall j \in \{1,\ldots,p\}\ \forall j' \in \{1,\ldots,p\}\quad V_{jj'} = \sum_{i=1}^{n} f_i \, z_{ij} \, z_{ij'} = \sum_{i=1}^{n} f_i \left(\frac{f_{ij}}{f_i \sqrt{f_{.j}}} - \sqrt{f_{.j}} \right) \left(\frac{f_{ij'}}{f_i \sqrt{f_{.j'}}} - \sqrt{f_{.j'}} \right)$$

les directions des axes principaux d'inertie sont celles des vecteurs propres associés aux valeurs propres de V, c'est-à-dire, si u_α est le vecteur unitaire du $\alpha^{ième}$ axe d'inertie : $V \cdot u_\alpha = \lambda_\alpha \cdot u_\alpha$ λ_α $\alpha^{ième}$ valeur propre

. Si l'on note $u_{\alpha j}$ la $j^{ième}$ composante de u_α, et $F_\alpha(i)$ la valeur de la projection du point x_i sur le $\alpha^{ième}$ axe d'inertie,

$$F_\alpha(i) = \sum_{j=1}^{p} \frac{}{f_i \sqrt{f_{.j}}} \, u_{\alpha j}$$

. Remarque : On a choisi de représenter les données dans $\mathbb{R}^p$; en fait, on peut, dans tout ce qui précède, intervertir le rôle des lignes et des colonnes de la matrice d'entrée, et définir un nuage de p points dans $\mathbb{R}^n$. On retrouve, par un calcul symétrique, exactement les mêmes valeurs propres non nulles, c'est-à-dire les mêmes axes d'inertie. Mais, la matrice V étant d'ordre $n \geq p$ par hypothèse et le nuage de p points se trouvant nécessairement dans un sous espace de $\mathbb{R}^n$ de dimension p-1 , on obtient en plus la valeur propre 0 à l'ordre n-p. C'est donc pour éviter des calculs inutiles que l'on choisit la représentation dans la plus petite dimension (le fait que cette plus petite dimension corresponde aux indices de colonne est une simple convention de programmation).

d) Interprétation des résultats :

Pour visualiser les calculs précédents, on fait une série de graphiques, chacun représentant les points dans un plan déterminé par deux des axes d'inertie (ces graphiques sont assez simples car les vecteurs propres sont toujours deux à deux orthogonaux - cela tient à la forme de V-).

On regarde ensuite, pour chacun des deux axes de chaque dessin, les positions des points par rapport à cet axe : des points éloignés de part et d'autre de l'axe "s'opposent", des points proches se "ressemblent". L'interprétation effective (pour laquelle l'ordinateur ne peut jouer aucun rôle) consiste à trouver quelles propriétés font que des points s'opposent ou sont voisins. Le plus souvent, cette interprétation n'est pas faite pour tous les graphiques : en effet, à chaque axe d'inertie est associé un pourcentage de l'inertie totale du système de points, et un axe ayant peu d'inertie donne des résultats peu sûrs, auxquels on peut faire dire à peu près n'importe quoi.

Comme on l'a vu dans la remarque précédente, la représentation en "points-colonnes" au lieu de points-lignes donne les mêmes axes. On peut donc représenter simultanément les deux nuages de points sur un même graphique. Cela permet, en plus de l'interprétation relative à chaque nuage, de comparer les comportement de chacun des deux par rapport à un axe donné ; mais il s'agit d'une comparaison globale ; la notion de proximité ou d'éloignement entre deux points appartenant à des nuages différents n'a pas de sens.

Cependant, si l'on veut situer les points d'un nuage par rapport à ceux de l'autre, on peut le faire en construisant d'autres graphiques de la façon suivante : on place les points du nuage de référence comme précédemment, et on met ensuite chacun des points de l'autre nuage au barycentre des valeurs qu'il a obtenues.

Exemple : On veut situer le point x_i (point ligne) par rapport aux points x_j de l'autre nuage. Si l'on note G (j) la projection de x'_j sur le $i^{ème}$ axe d'inertie, et H (i) la valeur de la composante (à déterminer) de x_i sur ce même axe on a :

$$H_\alpha(i) = \sum_{j=1}^{p} \frac{f_{ij}}{f_{i.}} G_\alpha(j)$$

§ 2 - MESURES SUR L'EDITEUR PROPREMENT DIT :

(2-1) Compteur de fréquence :

Il existe, affectée à chaque commande, une variable qui est incrémentée de 1 à chaque utilisation ; elle permet de calculer le nombre moyen d'apparitions d'une commande donnée au cours d'une session de YETI, et de comparer ces moyennes entre elles.

Pour les commandes de base : désignation du texte (G), insertion (I ou A), suppression (S), existant dans tout éditeur, on a fait un parallèle avec l'éditeur non interactif UPDATE, en usage à l'Ecole des Mines, et qui a donc les mêmes utilisateurs que YETI.

Pour les autres, on s'est provisoirement contenté d'opinions "qualitatives" liées à la nature de la commande : il est normal qu'une commande R (renommer un membre) soit plus rare qu'une commande I (insertion), que la commande Z (suppression d'une bibliothèque) apparaisse peu, etc... Les premières mesures, portant sur environ 80 sessions de YETI, ont donné les résultats suivants (la première ligne porte les noms des commandes, la deuxième le nombre d'apparitions moyen pour une session de YETI):

O	C	D	Z	G	W	X	I	N	A	A création	S	P	R	B	VS	M
1,6	0,4	1	0,1	5,5	8	0,4	24,5	9,2	24,3	3,5	17,8	41,5	0,5	4,2	0,3	24,9

Elles n'appellent pas pour le moment de commentaires particuliers : les commandes de suppression de bibliothèque (Z) ou de membre (X), de sélection de versions de texte (VS), sont rares, les commandes d'insertion (I ou A), suppression (S), modification dans une ligne (M), plutôt fréquentes. La comparaison avec UPDATE a porté, d'une part, sur le nombre de textes modifiés au cours d'une session : il y en a 9 (G+A-création) contre 5 environ à UPDATE ; d'autre part, sur le nombre d'insertions et de suppressions faites par rapport au nombre de textes désignés : pour les insertions, on trouve 5,4 (nombre de I + nombre de A divisé par le nombre de G+A-création), contre 8 à UPDATE, pour les suppressions (nombre de S divisé par le nombre de G+A-création) 2 contre 6 à UPDATE.

Ces différences semblent essentiellement dues, pour le moment, au fait que YETI est interactif : lorsqu'on veut travailler avec UPDATE, on note soigneusement toutes les modifications que l'on veut apporter à un texte et on les fait dans un seul passage (indispensable car il faut donner les changements à faire par numéros de ligne croissants); avec YETI, par contre, on peut modifier un programme, puis un autre, puis revenir au premier parce qu'on a pensé à autre chose en voyant les résultats du second, enfin, faire sur écran ce que l'on fait habituellement sur un listing avant d'utiliser UPDATE. Cependant, ces mesures sont poursuivies, afin d'avoir des résultats sur un nombre de sessions beaucoup plus important, ce pour deux raisons :
- tout d'abord, confirmer qu'il n'y a pas des commandes "sous-utilisées", dont il faudrait revoir la syntaxe (voir 2.2) ou l'exécution (voir 2.3);
- ensuite, établir une fréquence moyenne normale des diverses commandes ; cela servira de référence pour étudier le comportement d'un utilisateur donné (voir § 3).

(2.2) Compteur d'erreurs :

Ce compteur - il y en a également un par commande - est incrémenté de 1 à chaque fois qu'une erreur est commise par l'utilisateur dans la syntaxe de la commande. Ce qui est pris effectivement en compte est le rapport $\frac{\text{compteur d'erreur}}{\text{compteur de fréquence}}$. On calcule ces rapports pour chaque commande, on en fait la moyenne et on compare chaque rapport à cette moyenne. S'il est trop élevé, il y a avantage à revoir la syntaxe de la commande, problablement trop compliquée. Un exemple : les commandes de YETI n'ont actuellement que des paramètres positionnels ; il peut être utile de remplacer certaines d'entre eux par des paramètres à mots-clés, plus longs mais plus "parlants". Aucune mesure n'est présentée dans ce paragraphe car elles ne sont pas encore en nombre suffisant.

(2.3) Compteur de temps d'exécution :

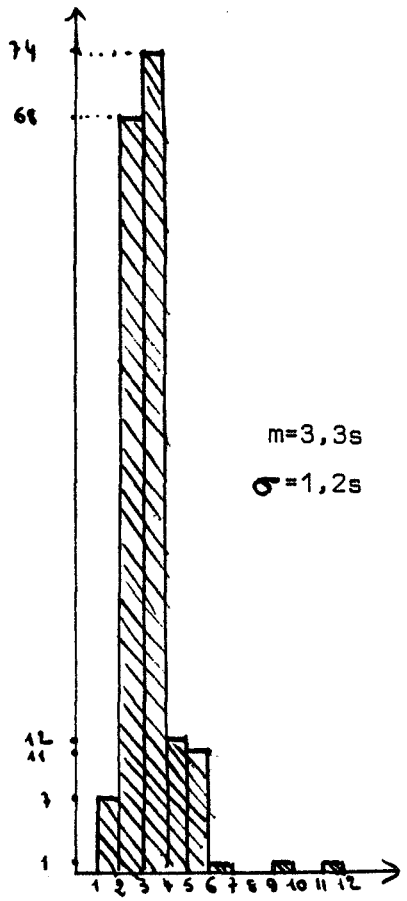
Ce compteur permet de découvrir les commandes trop longues ou trop courtes. On a établi un temps moyen de la façon suivante : d'après, en particulier, l'article de R.B. MILLER ([7]), il faut compter :
- 5 secondes maximum pour une réponse à une question du type "système m'entendez-vous ? "
- 2 à 5 secondes pour une réponse à une demande de travail ou d'information.

On a donc supposé qu'un temps moyen d'exécution pour une commande donnée, pouvait être considéré comme normal s'il était inférieur à 5 secondes (temps d'exécution : temps total écoulé entre le moment ou l'usager transmet sa demande et le moment où l'éditeur lui envoie sa réponse, accompagnée du message "commande suivante ?".).
Les résultats des mesures faites sont donnés dans les pages suivantes, sous la forme d'un histogramme pour chaque commande, accompagné de la moyenne m et de l'écart-type o des chiffres obtenus. Chaque histogramme porte en abscisse les temps d'exécution et en ordonnée le nombre d'exécutions dans un intervalle de temps donné. Tous les histogrammes sont représentés à la même échelle.

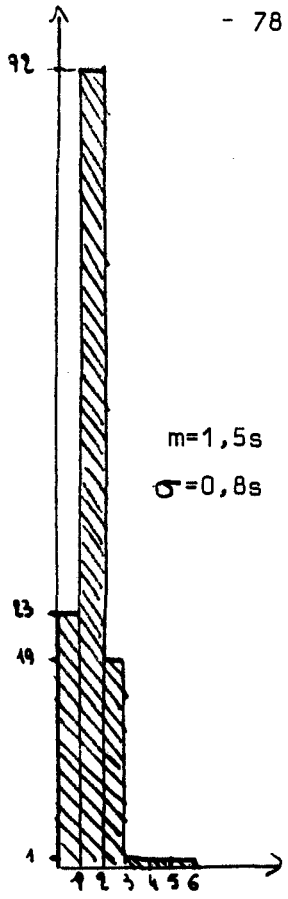
Remarques : 1) il n'y a pas de mesures pour les commandes F et E, car leur implémentation est très récente.

2) les temps très longs (plus de 8 secondes) sont dus à une perturbation des transmissions-écran ; cette perturbation, causé par les essais d'une liaison entre le P1175 et un ordinateur Télémécanique 1600, doit durer quelques mois, aussi on a quand même tenu compte de ces valeurs pour les premiers résultats.

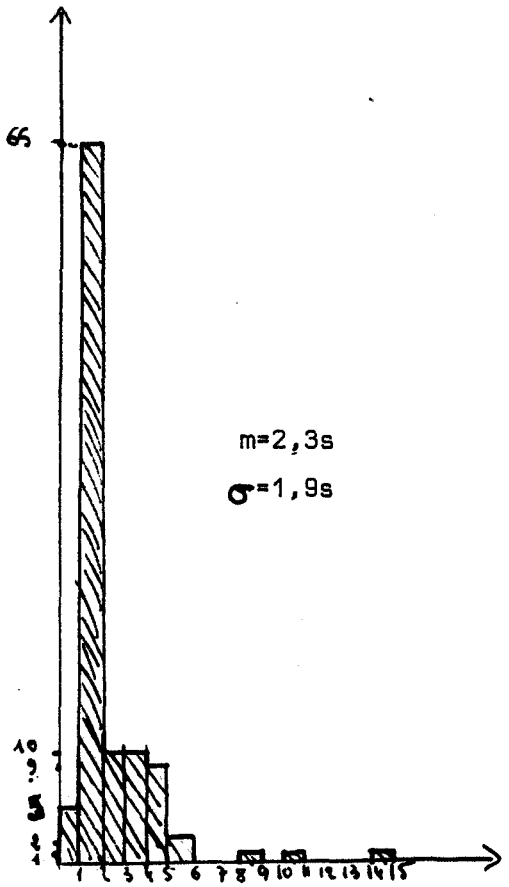
3) pour les commandes d'impression, ou d'insertion (P,I,A), où l'éditeur répond en plusieurs fois, on a compté comme temps d'exécution le temps écoulé entre la transmission de la demande et l'arrivée de la première partie de la réponse.



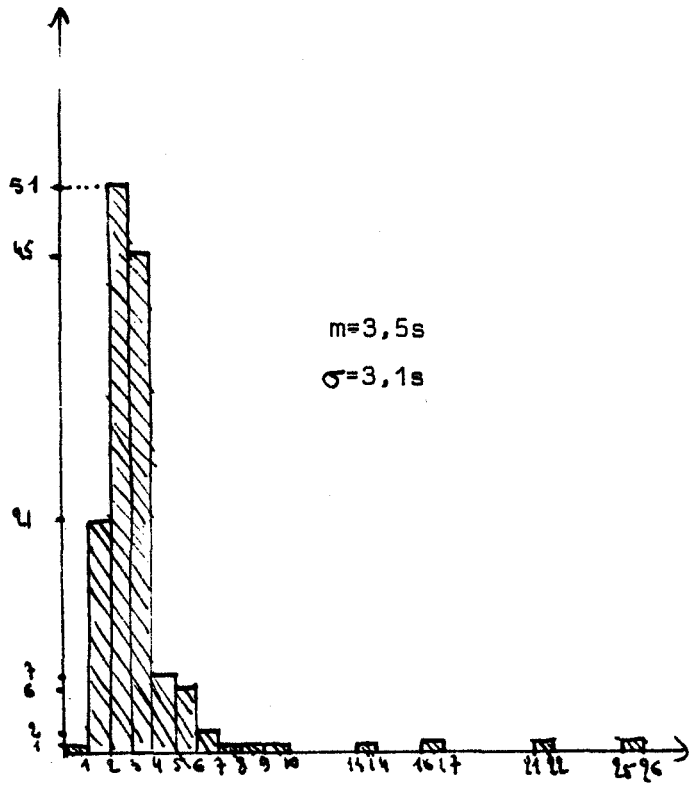
Commande O



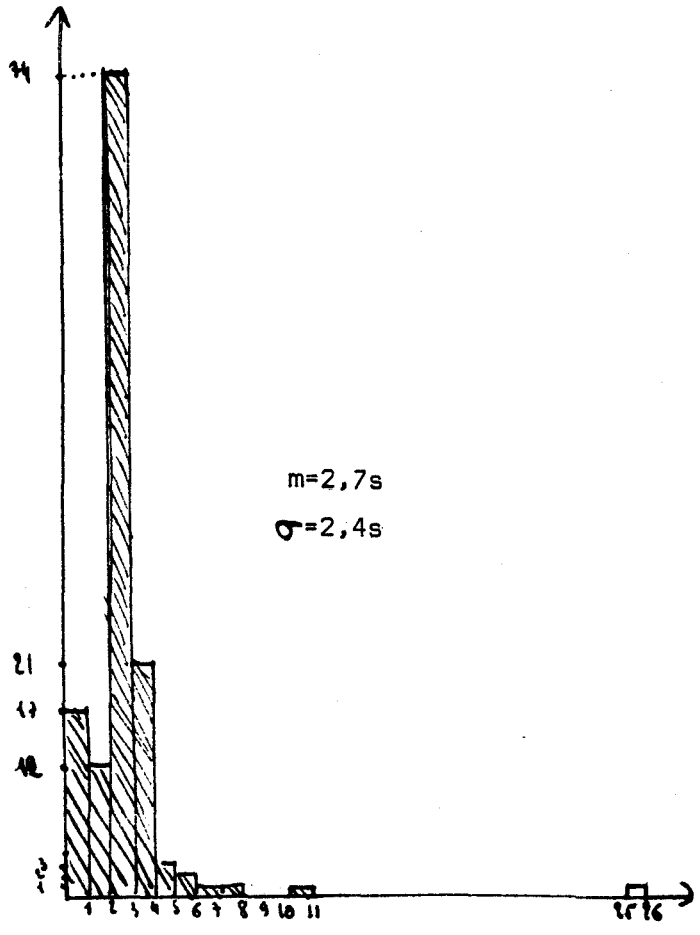
Commande C



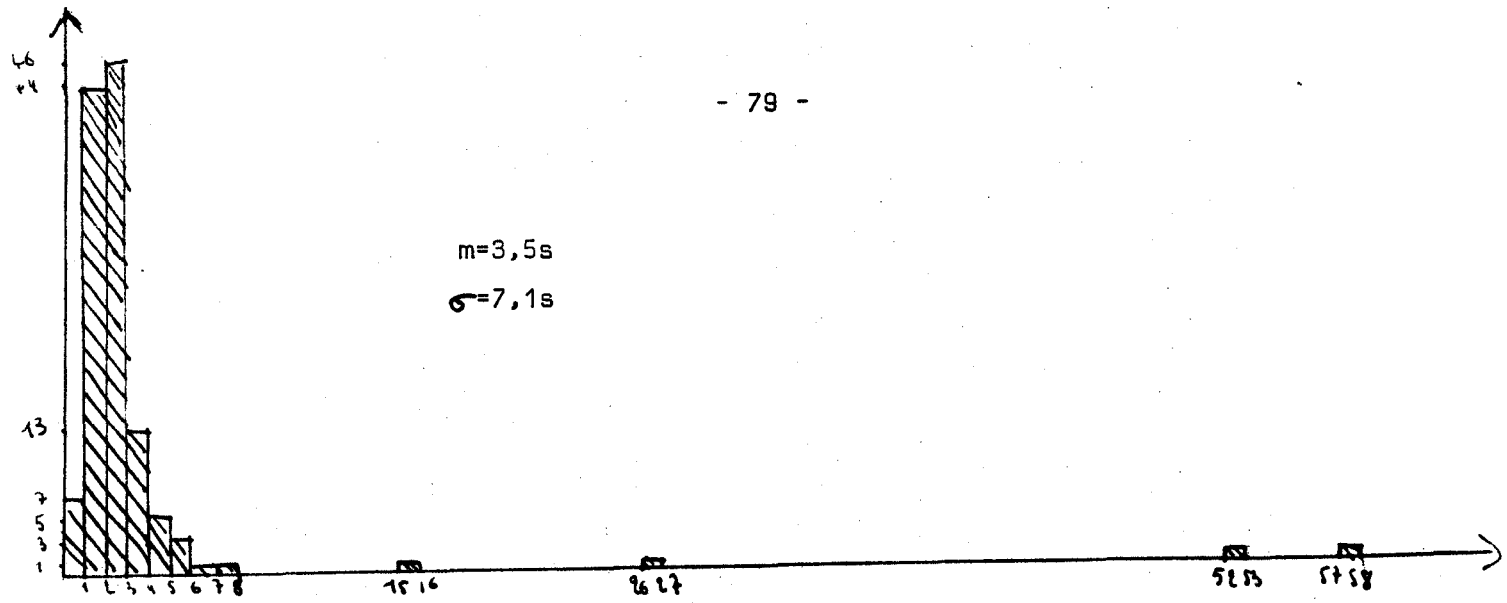
Commande D



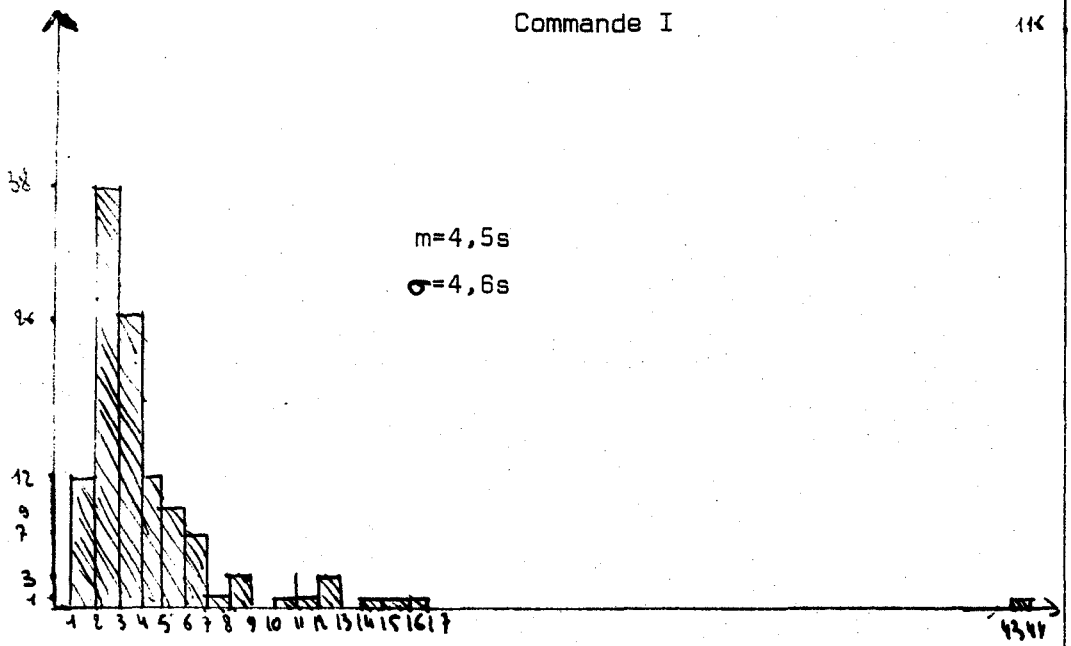
Commande W



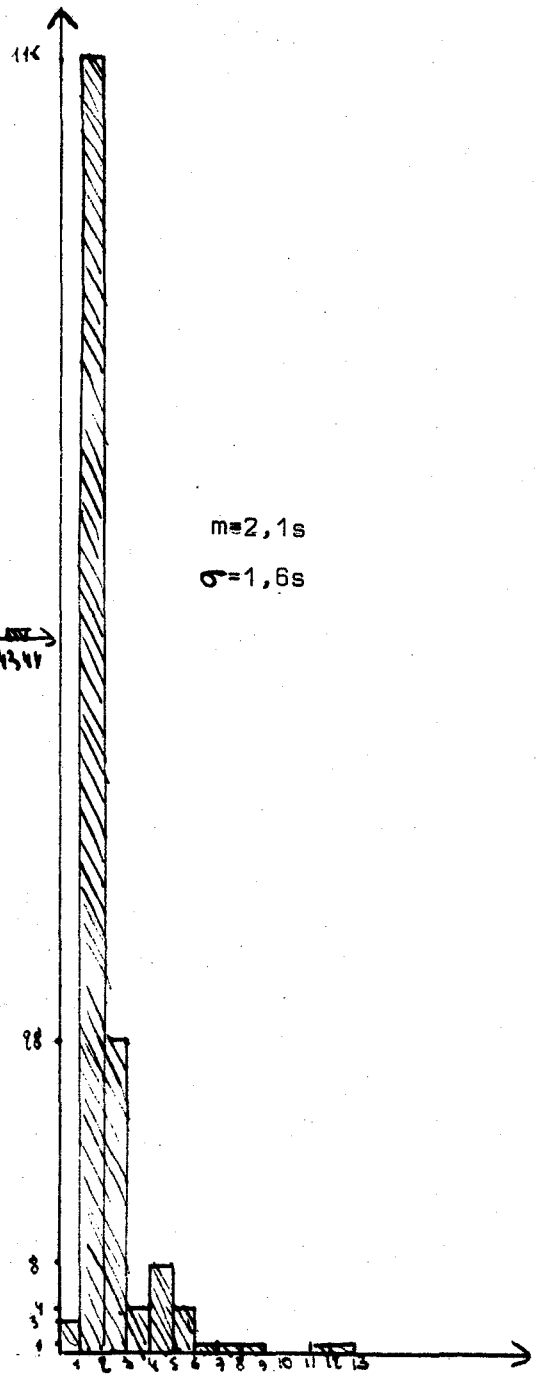
Commande X



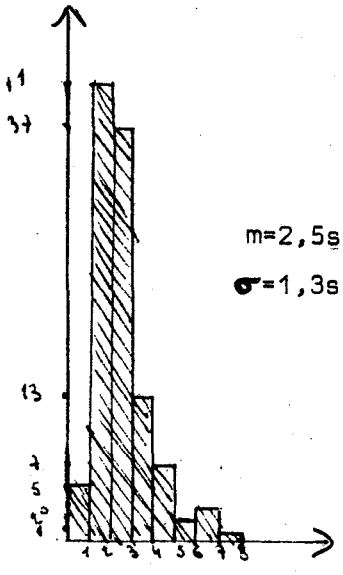
Commande I



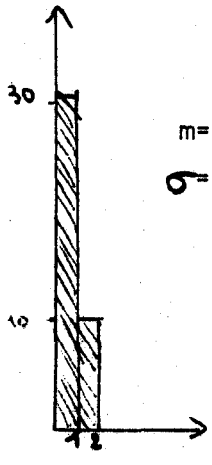
Commande B



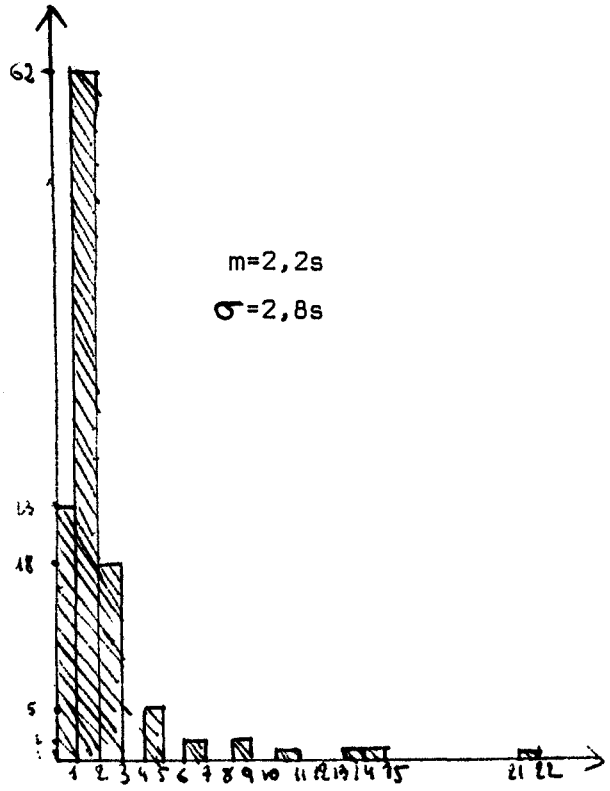
Commande G



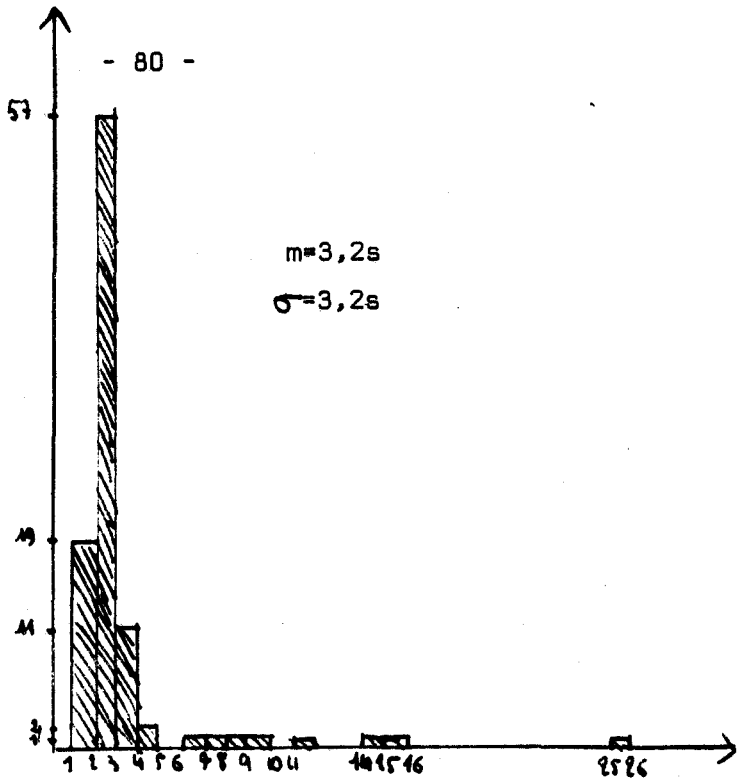
Commande S



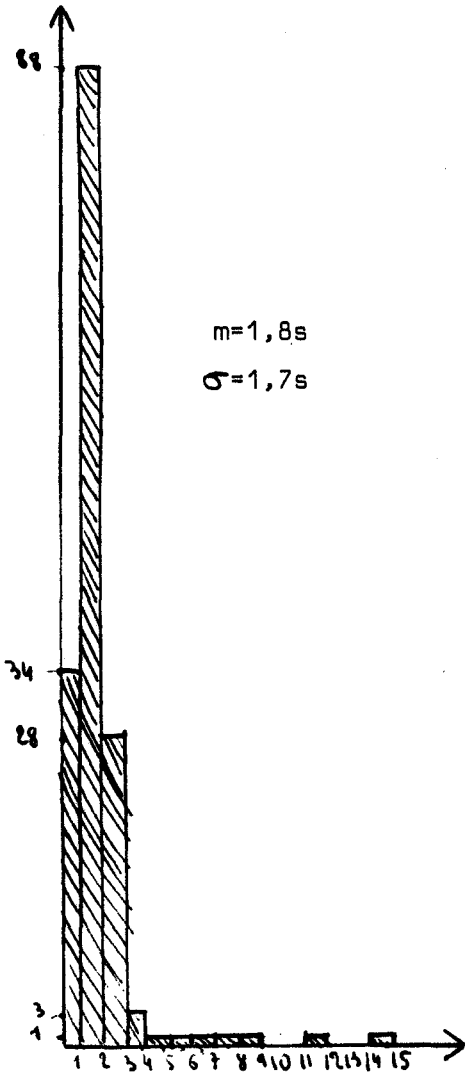
Commande Z



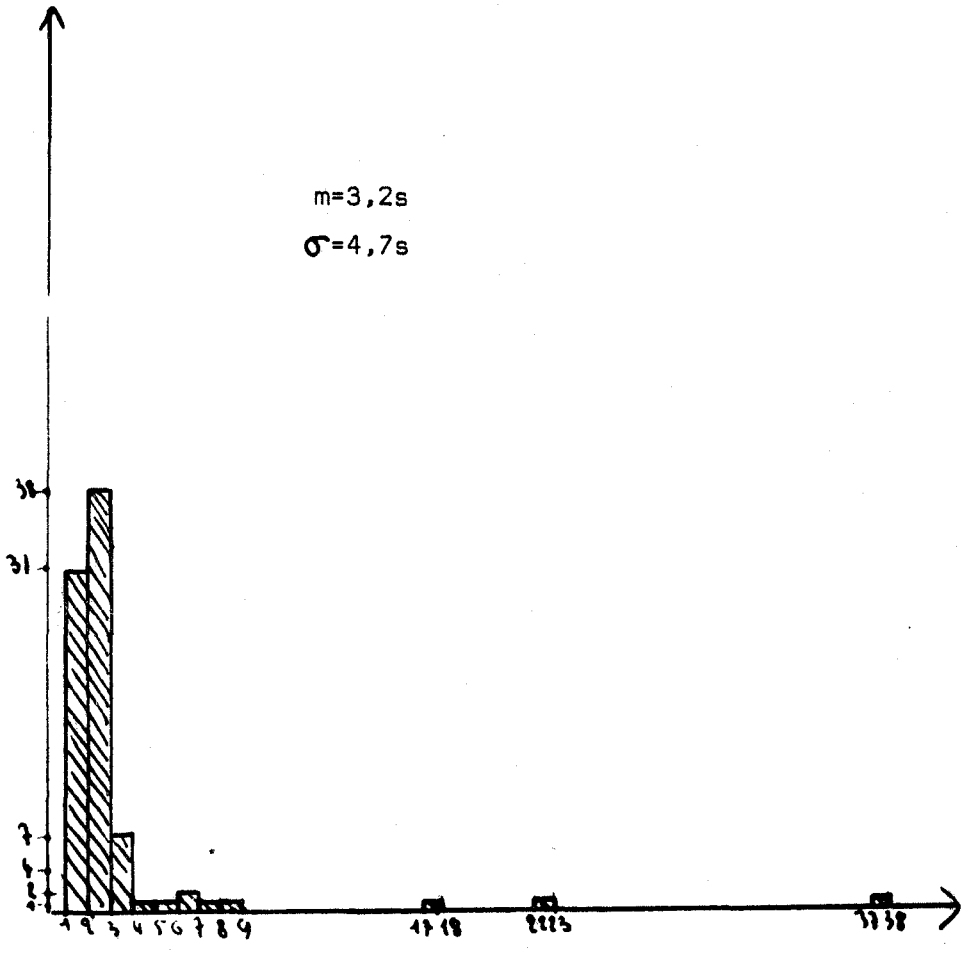
Commande N



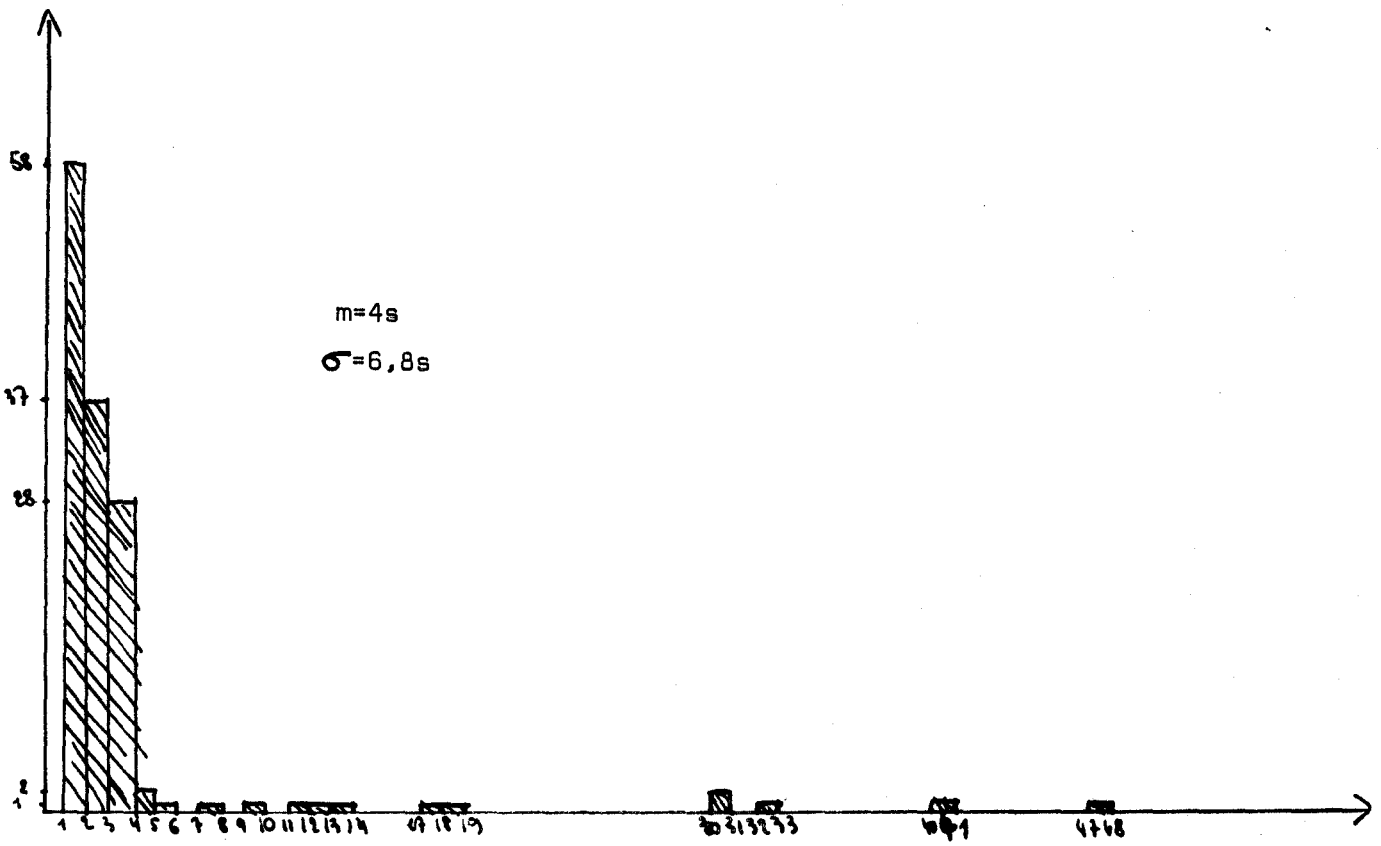
Commande VS



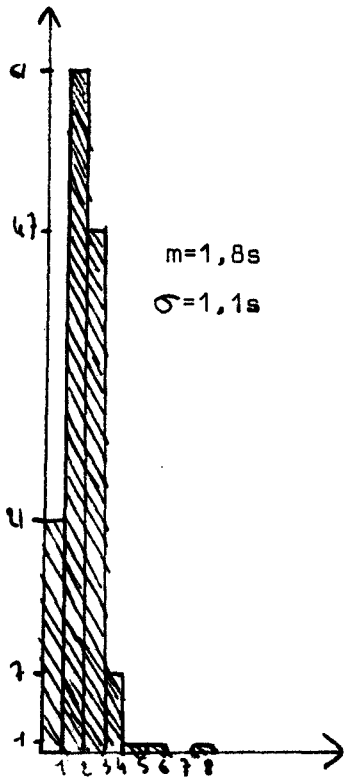
Commande P



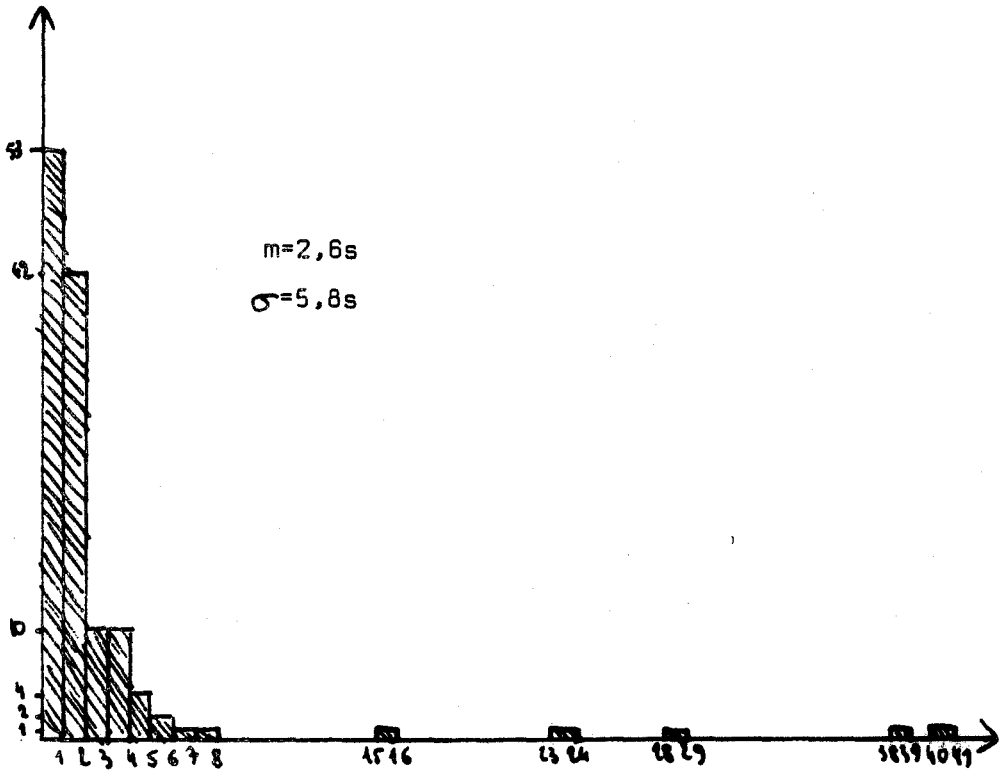
Commande A - insertion.



Commande M



Commande R



Commande A - Création.

(2.4) Comptabilisation des chaînes de commandes :

Cette comptabilisation est faite sous la forme d'une matrice carrée (S_{ij}) : les commandes sont numérotées en les classant dans un ordre arbitraire ; on ajoute 1 à l'élément s_{ij} si la commande numéro j suit immédiatement la commande numéro i.

Après un nombre suffisant de mesures, on peut essayer de déterminer les commandes qui se suivent très souvent. La méthode employée sera, soit la méthode des nuées dynamiques, soit l'analyse des correspondances ; actuellement, les premières mesures ont été traitées par l'analyse des correspondances, mais n'ont donné aucun résultat interprétable, car ils étaient trop peu nombreux. Si l'on découvre des séquences de commandes privilégiées, on peut alors les rassembler en une commande unique. Un exemple : la commande M permet de modifier une chaîne de caractères dans une ligne donnée ; il peut être intéressant de la généraliser au remplacement d'une chaîne donnée partout où elle se trouve entre telle ligne et telle autre.

Toutefois, même si rien de particulier ne ressort de ces mesures, il est prévu d'ajouter à YETI une possibilité de stockage des chaînes de commandes. Aussi un utilisateur qui a besoin plusieurs fois de la même suite pourra la faire exécuter sans avoir à la réécrire.

§ 3 - *EVALUATIONS ET MESURES SUR LES UTILISATEURS* :

(3.1)

Un des buts originels, dans la construction de YETI, était d'arriver à produire un éditeur adaptable à chaque utilisateur. D'autre part, les problèmes soulevés en cours de mise au point, ou pour l'interprétation de certaines mesures, ont souvent eu pour conclusion : "cela dépend surtout de l'usager". On a donc été conduit rapidement à étudier de près cet utilisateur.

Dans une première approche, on a relevé :

- ses caractéristiques personnelles : débutant ou expérimenté (pour le maniement de YETI), modifie des programmes écrits en tel ou tel langage, répond vite, etc...
- ses réactions vis-à-vis de l'éditeur : quelles commandes il utilise le plus souvent, quelles chaînes de commandes il préfère, s'il prend beaucoup d'options par défaut, quelles erreurs il fait,...

Le plan général prévu est le suivant :
- obtenir des paramètres précis
- en possession de ces divers paramètres, essayer de les relier entre eux, afin de déterminer quel utilisateur fait tel type d'action.

Un exemple de chacun de ces cas est donné ci-dessous.

(3.2) Exemple 1 : Mesure des temps morts entre les commandes :

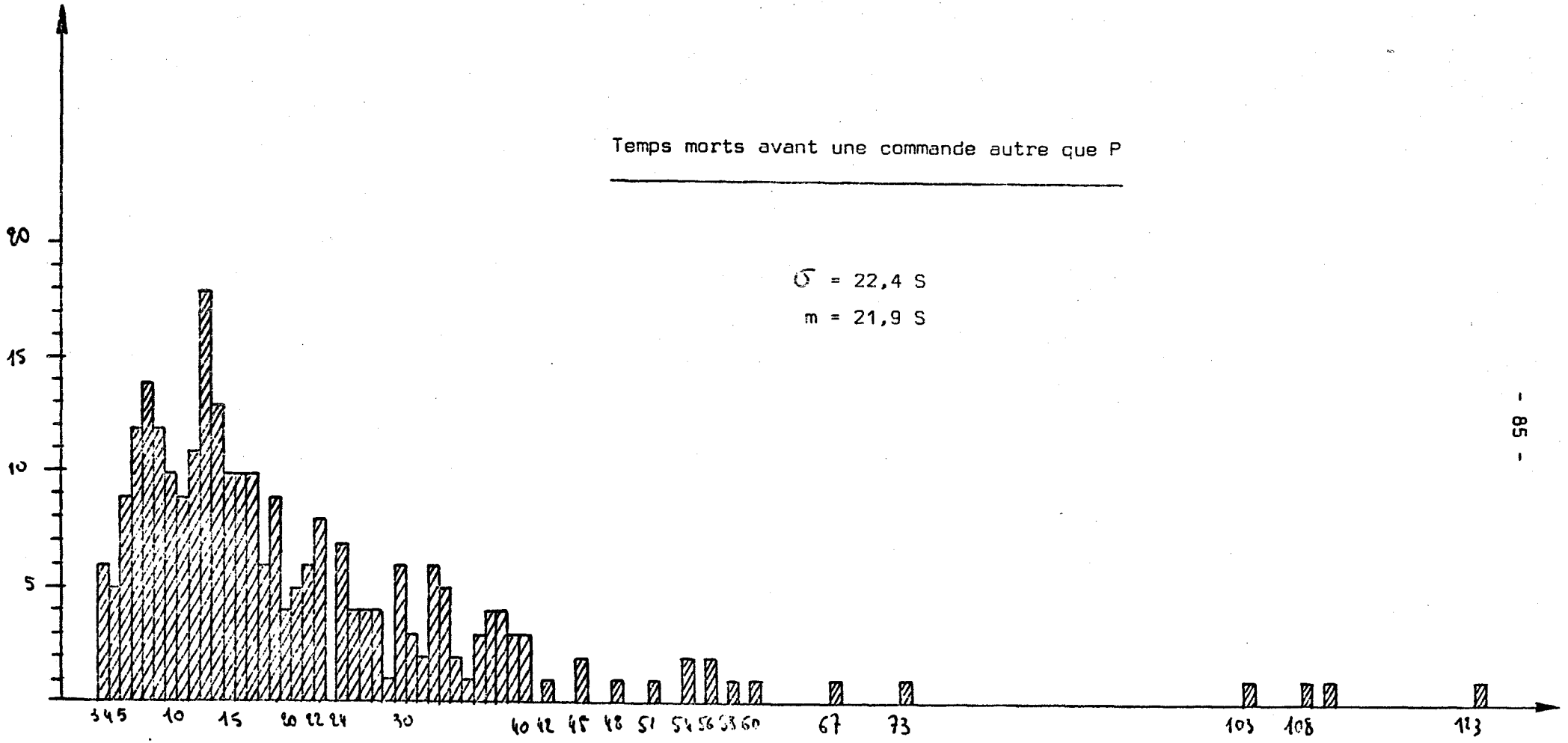
On s'est inquiété de cette notion afin de pouvoir préciser le paramètre "rapidité de réponse de l'utilisateur". On a relevé les intervalles de temps écoulés entre le moment où l'usager reçoit une réponse (la dernière partie s'il s'agit d'une réponse en plusieurs morceaux) et le moment où il transmet une nouvelle commande (la première partie s'il s'agit d'une insertion de plus de 13 lignes).
On a obtenu un temps moyen de 17,7 secondes ; ces mesures étant encore assez dispersées (écart-type : 20,6 secondes ; voir histogramme ci-dessous), on n'a considéré ce résultat que comme une première indication.
Cependant, un fait curieux apparaissait : le relevé fait prenait note, non seulement de la durée du temps mort, mais des commandes entre lesquelles il était intercalé. Or, il semblait que les couples terminés par P (impression) correspondaient à des temps morts assez courts.
Pour vérifier cela, on a refait les calculs en isolant les temps morts avant la commande P : cela a donné (voir également histogrammes ci-dessous) :
Temps mort moyen avant la commande P : 9,7 secondes
Temps mort moyen avant une commande autre que P : 21,9 secondes.
Ce résultat peut s'expliquer avec la notion de "tranches d'activité" citée au chapitre 3. Pour l'usager, faire une insertion, ou une suppression, est une tranche distincte de l'activité "modifier un programme" . Par contre, visualiser une partie de texte correspond à un simple contrôle des événements ; la commande P est donc ressentie le plus souvent par l'utilisateur comme partie intégrante de la "tranche" insertion, ou suppression, ou autre, qui la précède, et il la compose pour vérification dès qu'il a reçu le message annançant l'exécution de la commande précédente. On remarque d'ailleurs sur l'histogramme le nombre important de commandes P qui suivent un temps mort inférieur à cinq secondes.

Le fait que l'on obtienne quand même un temps moyen de 9,7 secondes vient d'un petit nombre de commandes P qui suivent un temps mort exceptionnellement long. Cela peut s'interpréter par le fait que l'utilisateur a "perdu le fil" - peut être parce qu'il a été interrompu par un événement extérieur - et qu'il essaie, par une visualisation, de retrouver où il en était.
En même temps que ces mesures générales, on a implanté des mesures semblables pour chaque utilisateur. D'ici quelques mois, on aura donc les valeurs des temps morts moyens pour chaque utilisateur, et on pourra ranger les utilisateurs en classes en fonction de leurs temps de réponse.
(voir conclusion).

Temps morts avant une commande autre que P

σ = 22,4 S

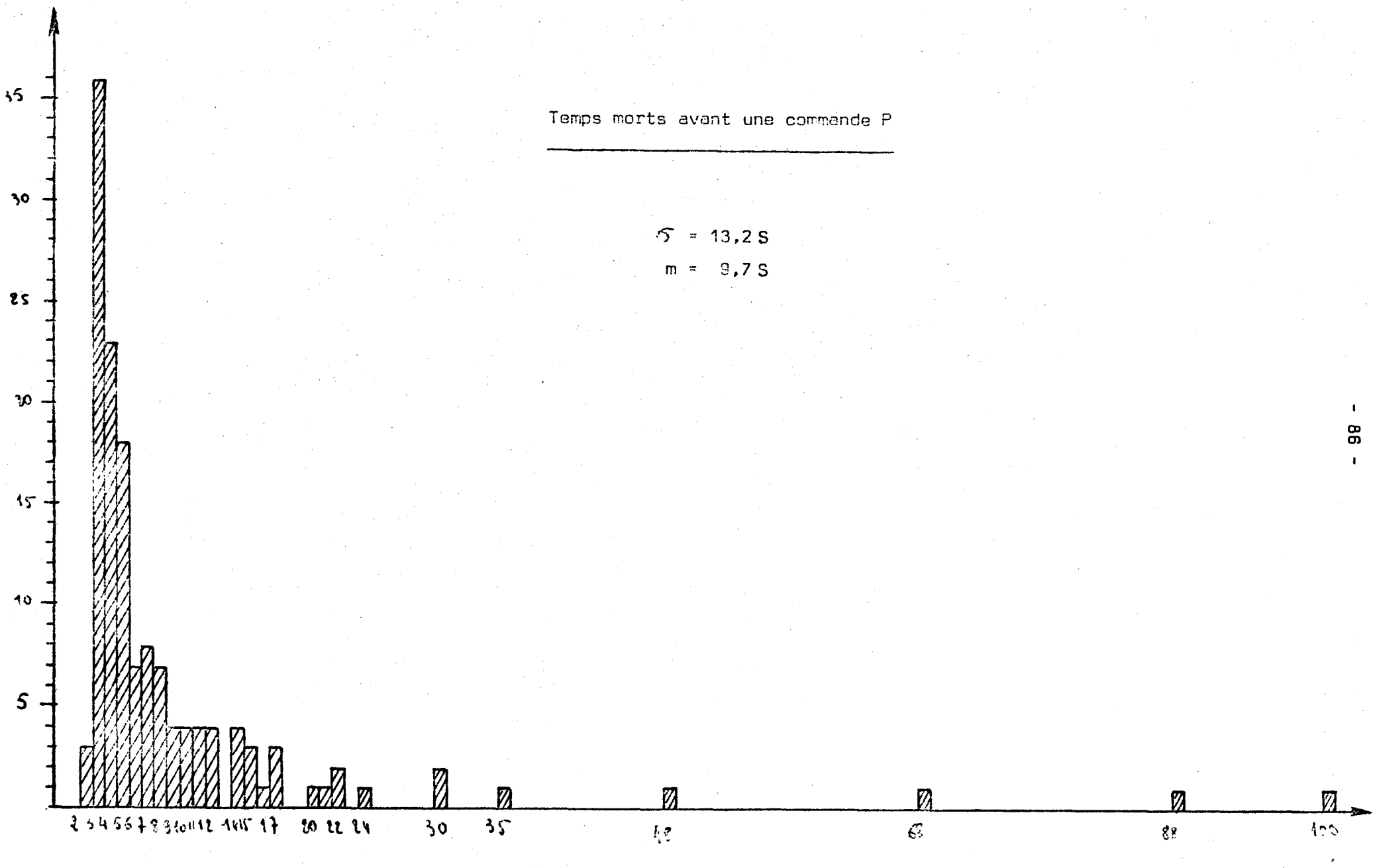
m = 21,9 S



Temps morts avant une commande P

σ = 13,2 S

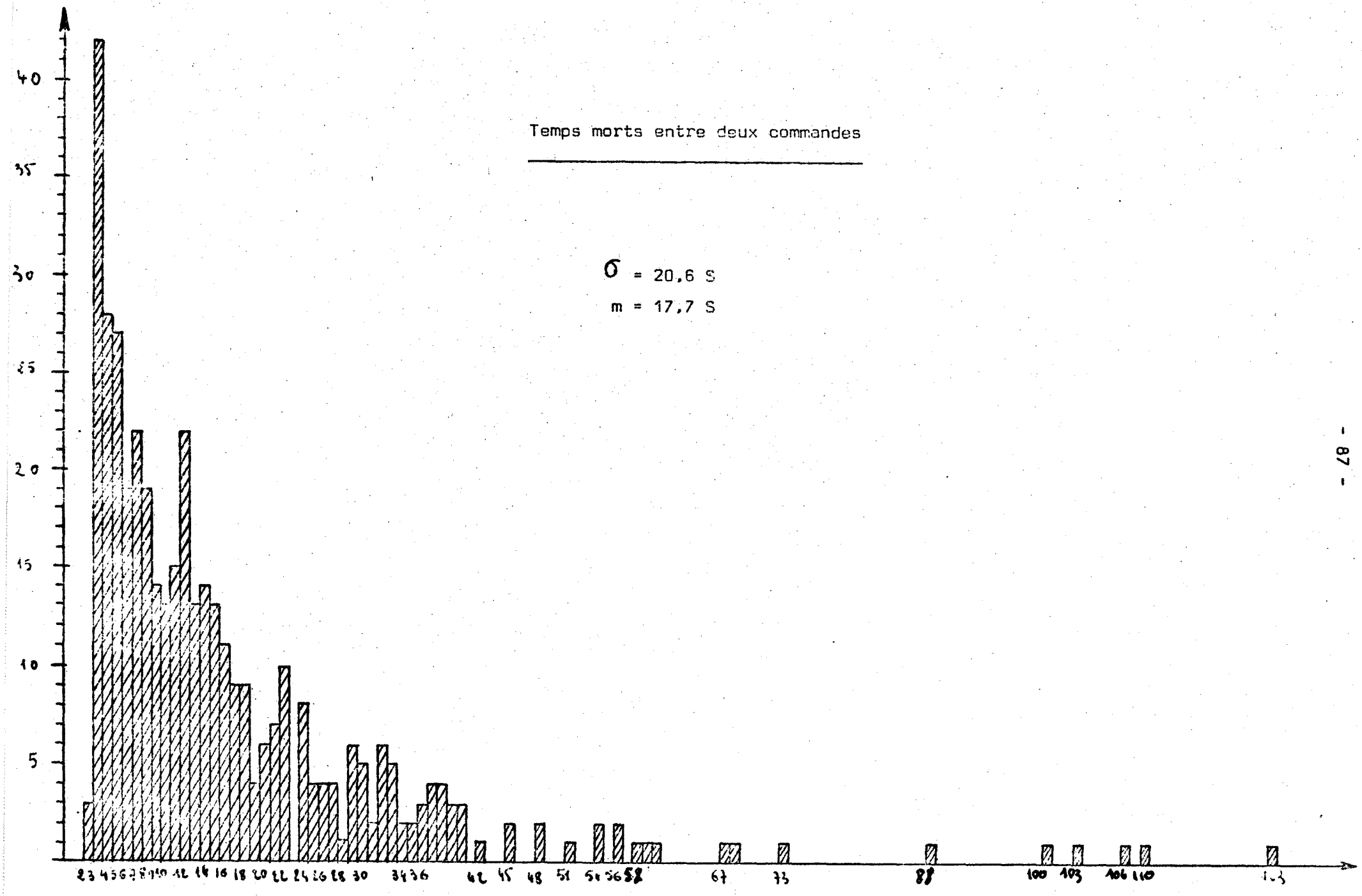
m = 9,7 S



Temps morts entre deux commandes

σ = 20,6 S

m = 17,7 S



(3.3) Exemple 2 : Recherche de préférences pour certaines commandes suivant le type d'utilisateurs et le langage des textes modifiés :

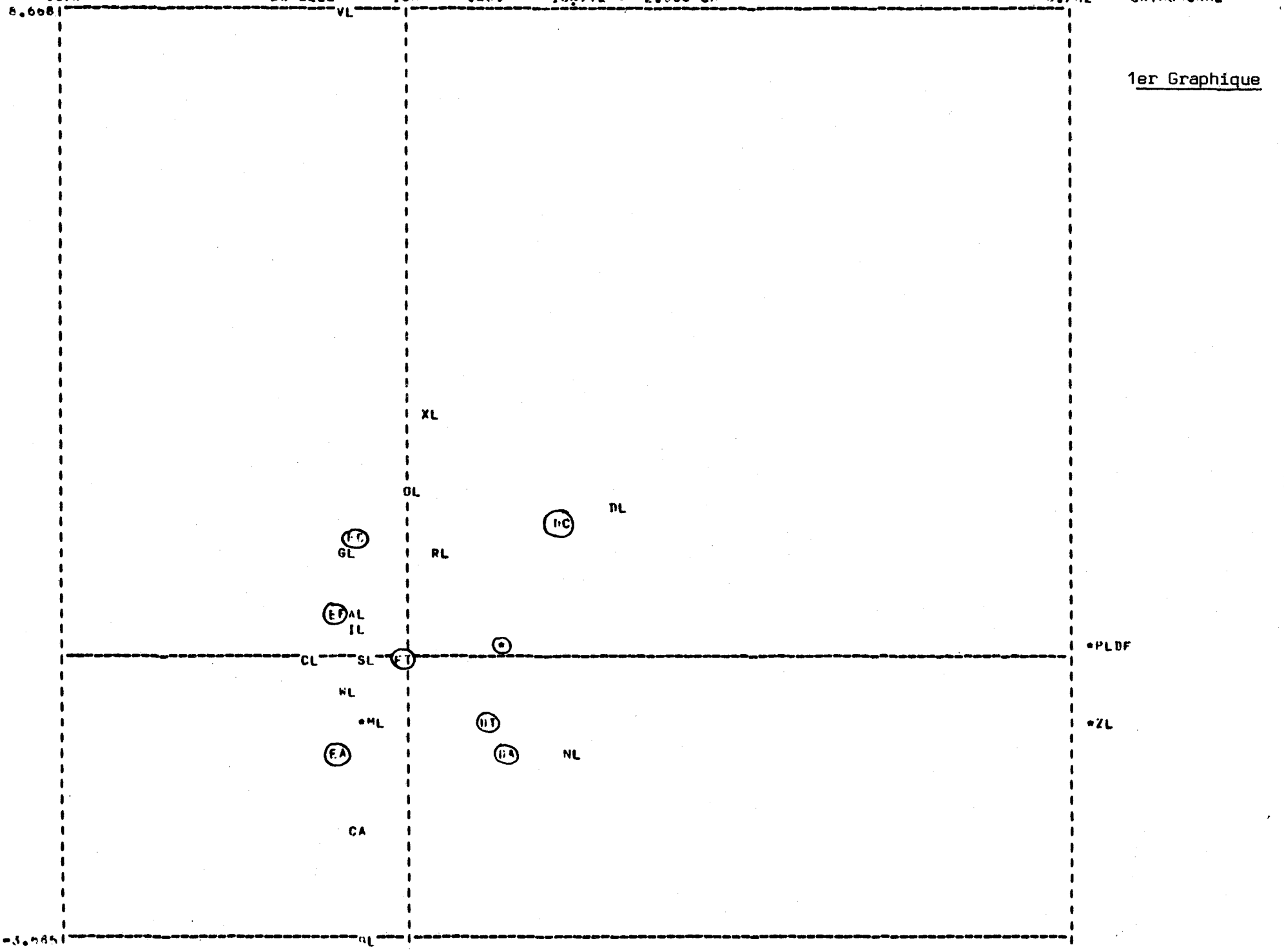
On a noté pour chaque utilisateur les deux caractéristiques suivantes :

- commence à travailler avec YETI on le connaît déjà bien.
- langage des textes modifiés : fortran, cobol, assembleur du P1175, autres (cette quatrième catégorie recouvre principalement les langages PL1600 et assembleur T1600, implantés sur l'ordinateur Télémécanique relié au P1175 ; le T1600 ayant peu de possibilités de stockage externe, ses utilisateurs conservent, leurs programmes sur les disques du P1175 ; les deux langages n'ont pas été distingués car leurs caractéristiques sont très proches).

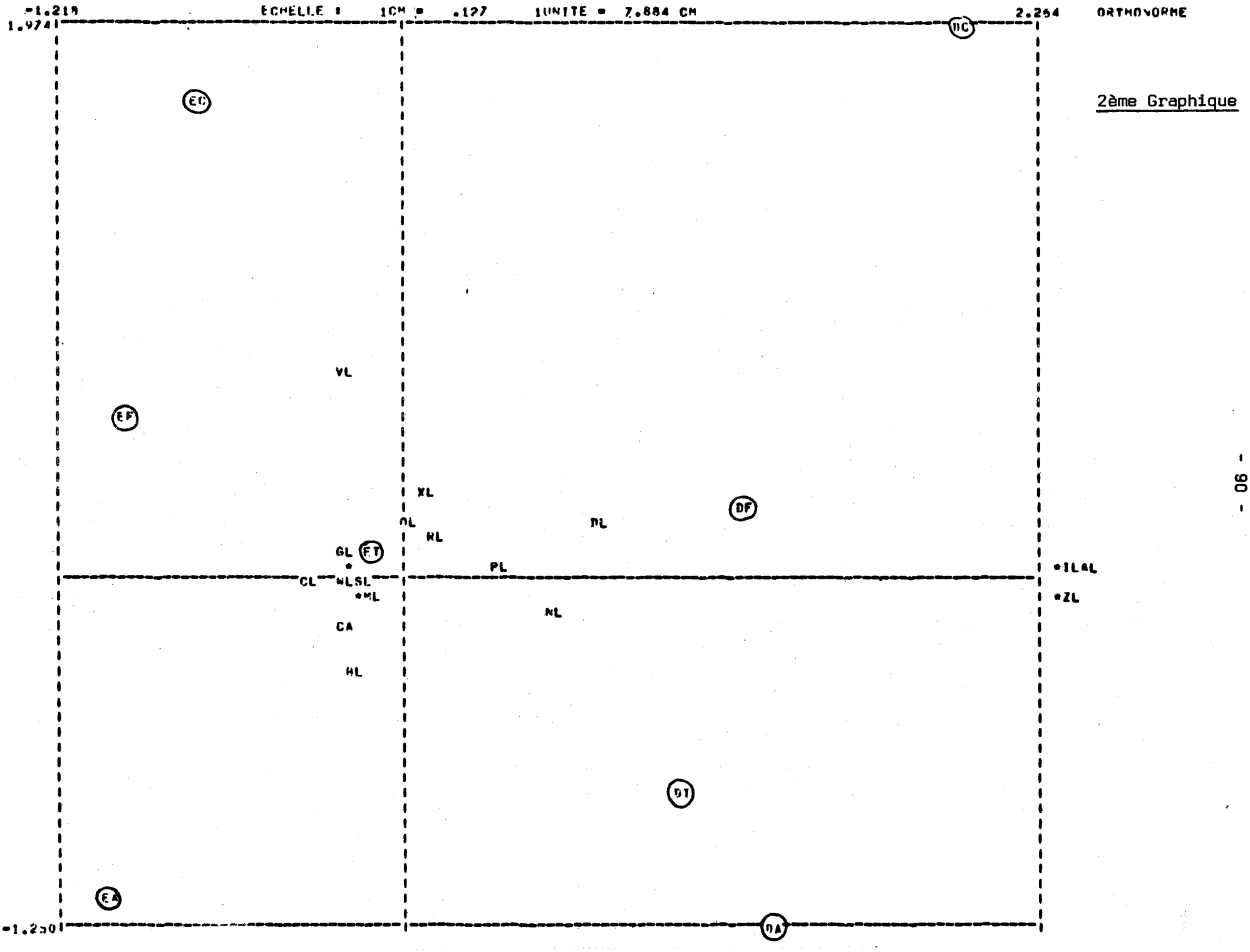
On a ensuite relevé le nombre de fois où chaque usager composait une commande donnée. Cela a permis de construire une matrice à 17 lignes et 8 colonnes (voir détail ci-dessous), que l'on a soumise à un programme d'analyse des correspondances (M. Mathon [25]).

Le premier graphique obtenu (voir pages suivantes) est la projection sur les deux premiers axes d'inertie du nuage I des points lignes et du nuage J des points colonnes. Sur le deuxième graphique, on retrouve les points de I à la même place, mais les points de J sont mis au barycentre des valeurs qu'ils ont obtenues dans les diverses lignes.
On n'a pas poussé l'interprétation plus loin que ces deux premiers axes, car ils expliquent à eux deux 88% de l'inertie totale, les axes suivants correspondants chacun à deux ou trois pour cent de l'inertie restante.
Les abréviations utilisées sont les mêmes pour les deux dessins :
OL, CL, DL, ZL, GL, WL, XL, IL, NL, AL, CA, SL, PL, RL, BL, VL, ML désignent respectivement les points-lignes correspondant aux commandes O,C,D,Z,G,W, X, I, N, A-insertion, A-création, S, P, R, B, VS, M.
DC, DF, DA, DT, EC, EF, EA, ET désignent les points correspondants aux débutants utilisant le cobol, débutants-fortran, débutants-assembleur, débutants- langages Télémécanique, et aux quatre colonnes analogues pour un usager expérimenté.

Le caractère * est mis à la place d'un des sigles précédents lorsqu'il n'est pas possible d'indiquer ce sigle (chevauchement avec un autre, ou deux à écrire au même endroit).



1er Graphique



2ème Graphique

Interprétation du premier graphique :

Nuage J : l'axe horizontal sépare les points EC,DC,EF,DF des points EA,DA,DT ; les langages Télémécanique sont des langages proches de la machine ; l'opposition peut donc s'interpréter comme une séparation entre langages proches de la machine (Télémécanique et assembleur 1175) et langages évolués (Cobol et Fortran).
L'axe vertical laisse à sa gauche les points EC, EF, ET, EA, et à sa droite les points DC, DF, DT, DA ; on a là une interprétation évidente : opposition entre usager débutant et usager expérimenté.

Nuage I ; l'axe horizontal oppose très nettement la commande B et la commande VS ; une explication qui est apparue plausible fait intervenir le nuage J : la commande B serait plus utilisée pour les textes en langage-machine, et la commande VS pour les textes en langage évolué. (On reviendra sur ce point en étudiant le deuxième graphique). Il oppose d'autre part les commandes G, O, R, X, VS, D aux commandes A-création, B, M, N, W. Cela peut s'interpréter ainsi : au dessus de l'axe, en compagnie de la commande G, on trouve toutes les commandes associées à un texte déjà entré en machine (C, X, VS, D), alors qu'au dessous, en même temps que A-création, on trouve les commandes un peu plus utilisées pour un nouveau texte : W, N, M, B ; cette explication est toutefois peu sûre car elle laisse de côté la commande Z.
L'axe vertical sépare les commandes de visualisation (P, N, D) des commandes plus complexes : VS, B, C (C est une commande simple ou elle même, mais son utilisation suppose l'emploi de plusieurs bibliothèques).

Interprétation du deuxième graphique :

Le nuage J n'est pas modifié, et on retrouve les interprétations précédentes. Chaque point du nuage I est situé au barycentre des points colonnes affectés des valeurs qu'il a obtenues ; un point ligne symbolisant une commande est donc d'autant plus proche d'un point colonne représentant un usager que cet usager s'est plus servi de la commande.
Tout d'abord, on trouve les commandes de visualisation - impression de texte, demande de numéro de ligne, demande de directory - dans le demi plan correspondant aux débutants, à droite de l'axe vertical. Cela correspond à une chose assez prévisible : un débutant passe beaucoup plus de temps à vérifier son texte et à contrôler ce qu'il fait qu'un utilisateur habitué.

En contrepartie, on retrouve dans le demi plan des usagers expérimentés les commandes supposant une utilisation plus complexe de YETI (VS, B, C). Ensuite, on trouve le point ET - utilisateurs expérimentés modifiant des textes écrits en langage assembleur T1600 ou PL1600 - à peu près au milieu de toutes les commandes. Cela vient de deux choses : tout d'abord, parmi les "habitués" de YETI, ce sont les plus anciens ; ensuite, ils ont beaucoup de pratique du conversationnel - les terminaux propres au T1600 étant tous interactifs. Chose que l'on avait déjà vue dans le premier graphique, la commande VS - sélection de certaines versions d'un texte - semble plus utile pour les langages évolués (effectivement, c'est plutôt dans ces langages que l'on garde des versions voisines d'un même programme) ; inversement, la commande B - mise en buffer d'un morceau de texte - est plus fréquente pour les langages de type assembleur - peut être parce qu'on a plus facilement des séquences d'instructions identiques qu'en Fortran ou en Cobol, séquences que cette commande permet de ne pas réécrire) ;
Il n'y a pas d'autres éléments faciles à interpréter ; certaines proximités - par exemple l'affection des débutants utilisant le fortran pour la commande "impression de directory" - restent mystérieuses et demandent à être précisées par des mesures supplémentaires.

CONCLUSION

YETI est actuellement opérationnel depuis un mois et demi. Entre cette date et la rédaction de ces pages, trois commandes ont été modifiées :

- Les commandes Z et T ont été respectivement remplacées par ZZ et TERMINE. En effet, les commandes de YETI ne comportent en général qu'une lettre ; c'est plus simple qu'un mot entier, mais le risque d'une faute de frappe est beaucoup plus grand. Cela a peu d'importance quand la syntaxe de la commande est complexe ; dans ce cas, la lettre erronée a peu de chances de cadrer avec la suite, et l'on obtient le plus souvent un message d'erreur. Mais des commandes telles que Z et T se composent uniquement de la lettre, et d'un chiffre qui peut éventuellement être omis. Si elles sont tapées par erreur, elles ont de fortes chances d'être admises, et leur action est irréversible (suppression de bibliothèque pour Z, fin de session, donc perte de tout ce qui est dans l'espace de travail, pour T). C'est pour éviter cela qu'elles ont été allongées.
- la commande P a été munie d'un système d'interruption : on peut arrêter à tout moment une impression en cours. Cela est utile, par exemple, lorsqu'on a voulu visualiser l'ensemble du texte et que l'on aperçoit, au milieu, quelque chose à modifier.

Une nouvelle commande : "?" a été créée. Cette commande recouvre toutes les demandes d'information possibles sur le déroulement de YETI.
Pour le moment, on peut demander :

- les noms des bibliothèques ouvertes (?BIBL) ; ces noms sont affichés sur l'écran lors de la mise en service de la bibliothèque, mais l'usager se sert ensuite seulement du numéro, et il peut oublier à quelle bibliothèque il correspond.

- la syntaxe d'une commande (? identificateur de la commande) ; utile pour les débutants et les distraits.
- les "dcbname" utilisés (?DCB). En effet, dans le système actuel du P1175, il est nécessaire, lorsqu'on demande l'exécution d'un travail, de donner la liste des bibliothèques et fichiers que l'on désire utiliser ; cette liste est composée de cartes, chacune donnant toutes les indications sur un fichier, et commençant par DCB <dcbname> (voir chapitre 5). Le nom <dcbname>, choisi par le programmeur, est celui employé dans la commande d'ouverture des bibliothèques de YETI. Cela évite à l'utilisateur de répéter les informations déjà écrites sur la carte annonçant la bibliothèque. Mais il est possible que l'usager oublie l'un des noms qu'il a attribués. La commande ?DCB lui permet alors de revoir les cartes qu'il avait données.

Enfin, la commande M devrait bientôt permettre de changer une chaîne de caractères dans toute une portion de texte, et non plus seulement sur une ligne. L'idée de cette amélioration avait été soulevée au chapitre 6 ; on avait pensé en déterminer l'utilité en étudiant les séquences de commandes fréquemment employées ; mais le nombre de données était insuffisant pour que l'ordinateur puisse en tirer une conclusion précise. Il semble que dans ce domaine l'homme soit plus rapide que la machine : ce sont les utilisateurs eux mêmes qui sont venus réclamer.

Le nombre des demandes : "YETI ne pourrait-il pas aussi faire cela? "a d'ailleurs été fort important, et presque toutes les modifications en cours sont la conséquence de ces demandes.

Cet engouement des utilisateurs pour YETI rend d'autant plus intéressantes les mesures faites sur ces utilisateurs. Bien entendu, on retrouve dans ces mesures, comme dans celles sur les performances de l'éditeur, le souci d'améliorer le produit. Cependant, on essaie également de poursuivre un autre but : la "personnalisation" de l'éditeur. En effet, lorsqu'on aura collationné un nombre suffisant de mesures, il sera possible de diviser les utilisateurs en classes, en fonction de leurs caractéristiques et de leurs réactions ; d'autre part, l'interprétation des résultats des mesures permettra de définir la conduite vis-à-vis de YETI d'une classe donnée d'utilisateurs, ie les erreurs les plus fréquentes,la façon d'employer les commandes, etc...

Ceci étant supposé réalisé, imaginons l'arrivée d'un nouvel utilisateur ; il donne ses caractéristiques ; en quelques passages, l'éditeur peut être à peu près fixé sur ses réactions, et le situer dans une classe donnée. Dès lors, il peut l'aider d'une façon beaucoup plus efficace : par exemple, lui signaler particulièrement les erreurs auxquelles il est prédisposé. Après un temps d'utilisation plus long, il peut également rendre service ; par exemple, il est prévu d'ajouter à YETI une possibilité de stockage des chaînes de commandes ; évidemment, le choix des chaînes à stocker est laissé à l'usager ; mais il ne s'apercevra peut être pas immédiatement qu'il emploie fréquemment telle ou telle chaîne. L'éditeur pourrait alors le lui signaler. On aurait ainsi un éditeur adaptable à chaque usager, et donc plus efficace. Dans le même ordre d'idées, on pourrait dégager, à long terme, de l'ensemble des classes d'utilisateurs, un profil général d'usager de centre de calcul. Après tout, les machines sont faites pour eux, et il peut être intéressant de les connaître mieux. Cette idée est donnée là pour ce qu'elle vaut, ie peut être pas grand-chose.

Mais laissons là les spéculations d'ordre psychologique pour revenir à des problèmes plus pratiques.

La construction de YETI n'est pas allée sans difficultés. Entre autres raisons - problèmes de matériel, de temps, etc...-, cela tient à un fait : lorsqu'on n'a pas eu l'occasion de pratiquer soi même divers éditeurs de texte, il n'est pas très facile d'avoir des connaissances en la matière. En effet, la littérature à ce sujet se compose essentiellement d'une part des manuels de référence des éditeurs existants, d'autre part d'articles qui résument ces manuels. Les premiers sont pratiquement introuvables si l'on ne connaît pas d'avance le nom de l'éditeur de texte et le lieu exact où il a été produit. Les seconds peuvent être découverts en feuilletant une quantité astronomique d' "AFIPS", "communications ACM", "ACM computing Surveys",et autres classiques de la littérature informatique de langue anglaise. Mais les uns et les autres ne donnent que l'image partielle de l'éditeur qu'ils décrivent, et peu ou pas d'informations générales. Il reste donc à voyager ou à inventer soi même son éditeur... à moins que l'on ait la chance de croiser au hasard d'une page, un certain article de A. Van Dam et David E. Rice. Cet article, intitulé "Online text editing : a survey"([3]) décrit brièvement un certain nombre d'éditeurs existants, avec leurs caractéristiques. Que ses auteurs soient ici remerciés...

Leur travail a fourni la trame des chapitres 1 et 2 du présent ouvrage. Il a été augmenté de renseignements en provenance de diverses sources (voir bibliographie). Le résultat, tel qu'il est, peut être utile aux éventuels fabricants d'éditeurs de textes : il leur évitera une fastidieuse compilation. Mais il n'est certainement pas parfait, probablement pas complet. En particulier, il ne comporte pas d'indications sur la meilleure forme possible d'éditeur de texte (en fonction du support, bien entendu) ; tant d'éléments contradictoires coexistent actuellement qu'il n'est guère possible d'en extraire un enseignement précis. On peut cependant, après expérience, tenter de tirer quelques conclusions :

- premièrement, un éditeur doit pouvoir être utilisé rapidement par un débutant. Il faut donc que son principe soit facile à comprendre, d'une part. D'autre part, il faut penser à fournir quelques commandes simples "à tout faire" ; cela n'empêche pas d'avoir des commandes plus raffinées pour les utilisateurs expérimentés.
- deuxièmement, un éditeur actuel doit accéder au caractère, quel que soit son type d'organisation interne.
- troisièmement, il est important de penser au facteur "sécurité du texte", il ne faut cependant pas lui donner une place exagérée. (YETI a eu des problèmes d'attente causés par des sauvegardes-disque trop fréquentes...)
- enfin, dernière chose et la plus importante : le programme doit être à tout moment aisément modifiable. Un éditeur est fait pour être continuellement changé, autant le prévoir.

Au delà de ces conclusions, la réalisation de YETI, et les mesures associées, ont eu également un autre effet. Elles ont été une incitation à réfléchir au rôle d'un éditeur de texte dans un système.
Il est actuellement confiné, la plupart du temps, à la correction de cartes pour la mise au point d'un programme (il suffit, pour s'en convaincre, de compter le nombre de fois où d'autres commandes qu'insertion et suppression sont utilisées ; pour l'éditeur non interactif de l'Ecole des Mines, UPDATE, les trois quarts des utilisateurs ignorent même qu'il peut faire autre chose). Il semble pourtant que l'éditeur a un rôle plus important à jouer. Des recherches ont été ébauchées dans deux directions.
La première est une tentative pour encourager l'utilisateur à concevoir vraiment ses programmes à l'écran ; cela peut se faire en pratiquant une forme d'écriture globale, et non plus la fastidieuse production ligne à ligne, en suivant un organigramme pas toujours rédigé. Bien sûr, il ne s'agit pas là de rivaliser avec Emily ([4]), qui permet pratiquement de générer des programmes corrects, mais dans un seul langage.

L'idée, suggérée par le fonctionnement des macro-assembleurs, est simplement de fournir la possibilité d'écrire des macro-instructions, dans un programme en langage quelconque.
Comme pour un macro-assembleur classique, on aurait d'une part, une série de macro-définitions, ayant chacune leur nom (créées par l'utilisateur ou fournies par YETI), d'autre part, dans le texte du programme, des macro-instructions. Ces macro-instructions se composeraient de deux caractères spéciaux laissées au choix du programmeur, et d'un identificateur qui serait celui de la macro-définition correspondante. Le premier caractère spécial pourrait être par exemple, celui qui annonce une ligne de commentaire dans le langage utilisé, pour éviter tout problème avec le compilateur ; le deuxième servirait à YETI, pour différencier les macro-instructions des vraia commentaires.
Une macro-instruction serait ignorée tant que l'utilisateur n'en demanderait pas explicitement le remplacement ; d'autre part, c'est dans cette demande de remplacement que seraient précisés les éventuels paramètres.
Ainsi, l'usager pourrait d'abord écrire son programme en employant autant de macro-instructions qu'il le désire ; il pourrait ensuite les remplacer au fur et à mesure. Il est à remarquer que la macro-définition correspondant à une macro-instruction n'a pas besoin d'exister tant que l'utilisateur ne demande pas l'exécution effective du remplacement. Cela permet donc de retarder le plus possible la rédaction exacte du morceau considéré (voir [28] à ce sujet). Bien entendu, le texte d'une macro-définition pourrait contenir une macro-instruction faisant appel à une autre macro-définition (éventuellement à la même si les problèmes posés par cette récursivité ne sont pas excessifs par rapport au résultat obtenu ; une solution simple peut être de faire le remplacement en recopiant telles quelles les macro-instructions contenues dans la macro-définition, et de laisser à l'usager le soin de commander les remplacements à faire). D'autre part, en choisissant effectivement comme premier caractère spécial celui qui annonce une ligne de commentaires dans le langage du texte, l'usager pourrait compiler et exécuter son programme sans avoir fait tous les remplacements. Cela faciliterait certainement la mise au point.
Cependant, pour qu'un tel système soit intéressant, il faut pouvoir compiler ou exécuter un programme, et récupérer les résultats immédiatement.

Actuellement, YETI possède deux commandes, permettant, l'une de copier un membre de bibliothèque sur un fichier, l'autre de faire l'application inverse ; on a vu (chapitre 5) que la première peut être utilisée pour lancer l'exécution d'un job, l'autre pour obtenir les résultats d'une compilation ou d'une exécution. Il faudrait leur adjoindre une méthode d'interrogation du système ; l'usager pourrait alors continuer à travailler, et demander de temps en temps où en est son programme. Il va sans dire que cet aspect "exécution possible à partir d'un éditeur" n'a pas le mérite d'une grande originalité - (voir WYLBUR [6]). Il est cité ici simplement comme complément naturel à ce qui précède.

La deuxième orientation possible pour YETI en est encore au stade des réflexions ; elle ne propose pas, comme la précédente, une extension de l'éditeur à des fonctions que rien d'autre ne remplit dans le système ; mais elle vise à faire accomplir par l'éditeur de texte des fonctions traditionnellement réservées au complilateur. En effet, que fait le compilateur ? En dehors de l'analyse du texte proprement dit, il repère toutes les références externes, fait un tableau de leurs noms et garde trace de chacune de leurs occurrences. Ce travail est destiné à préparer le terrain pour l'éditeur de liens, et n'est d'aucune utilité particulière au compilateur. Disons qu'en général il l'effectue parce qu'il est le premier maillon du système qui analyse le programme. Or, cet aspect de l'analyse est une recherche de chaînes de caractères à travers le programme. Cette fonction existe dans l'éditeur de texte. Dès lors, à quoi bon la doubler dans chaque compilateur ?. Bien sûr, une référence externe ne se présente pas sous la même forme suivant le langage employé, et peut même avoir plusieurs apparences dans un même langage. Mais les différents formats possibles sont assez peu nombreux pour être introduits dans l'éditeur sous forme de table.

D'autre part, pour être employée à cet usage, la recherche de chaînes de caractères doit être améliorée. En effet, en utilisation courante, on ne se préoccupe pas de l'environnement de la chaîne. Par exemple, si l'utilisateur demande le changement du nom de la variable x en y dans tout le texte, toute occurrence de x est transformée en y. C'est à l'usager de prendre garde aux x qu'il ne voudrait pas modifier, par exemple parce qu'ils font partie du nom d'une autre variable. Cela est évidemment inacceptable lorsqu'il s'agit de relever les références externes ; dans les langages tels que Fortran, où il n'y a pas de mots réservés, on aboutirait à des absurdités. Il faut donc modifier la recherche de chaînes pour qu'elle tienne compte de l'entourage de la chaîne à découvrir.

Cette amélioration pourra d'ailleurs être introduite dans la commande M normale de l'éditeur, ce qui évitera à l'usager le souci d'une modification pas toujours facile.
Cette partie du travail est problablement réalisable, bien que d'une taille énorme. Mais son intérêt ne peut être vérifié qu'en supprimant du compilateur la partie correspondante : cela pose donc certainement beaucoup de problèmes.

On a là quelques aperçus de ce qu'il serait possible de faire. Bien sûr, l'étude n'a pas été poussée plus avant pour le moment, et il se peut qu'elle ne donne rien. Mais il est problable qu'un éditeur de texte peut acquérir, dans un système, un rôle beaucoup plus grand que celui qu'il a actuellement.

BIBLIOGRAPHIE

[1] Program composition and editing with an online display (IPSS)
Bratman, Harvey, H.G. Martin, Ellen C. Perstein
AFIPS 68 (Vol. 33 p. 1349-1360)

[2] An online editor (QED)
Deutsch L. Peter, Butler W. Lampson
Communications A C.M. décembre 1967.

[3] Online text editing : a survey
Andries Van Dam et David E. Rice.
ACM computing surveys. septembre 1971.

[4] User engineering principles for interactive systems (EMILY)
Wilfrid J. Hansen
AFIPS 71 (vol. 39 p. 522).

[5] Computer assisted tracing of text evolution
Elliot W. David, Warren A. Potas et A. Van Dam
AFIPS 71 (vol. 39 p. 533)

[6] WYLBUR : An interactive text editing and Remote Job Entry System
Roger Fajman et John Borgelt
Communications ACM . juin 1973.

[7] Response time in Man Computer Conversational transactions.
R.B. Miller
AFIPS 68.

[8] Structures de données
Robert Mahl.
(Cours enseigné à l'Ecole des Mines de St Etienne et à l'UER Sciences de St Etienne).

[9] Computer Science - A first course.
Forsythe, Keenan, Organick et Steinberg.

[10] La Science Informatique.
J. Arsac.

[11] Techniques de l'ingénieur - Traité pratique d'informatique.
Nicolas Manson.

[12] Introduction à l'Informatique
J. Dondoux, Ph. Marano, J.C. Merlin.

[13] Techniques de Compilation
F.R.A. Hopgood

[14] An introduction to Data Communications
Data communications system description
P1088 display unit
Data communications programming information
Philips

[15] Basic Partitioned Access Method
Librairies
Use of Monitors
Philips

[16] Performance détermination
T.E. Bell
AFIPS 1973 (vol. 42).

[17] Performance évaluation
(Titre général de 8 articles) AFIPS 1973 (vol. 42)

[18] Performance évaluation : a structured approach
S. Kimbleton
AFIPS 72 (vol. 40 p. 411).

[19] Performance prediction (modeling and measurement)
- J.G. Williams
- J.D. Noe et G. J. Nutt
- J. Rodriguez, Rosell et J. Dupuy
AFIPS 73 (vol. 42 p. 739).

[20] Modeling, measurement and computer power
G. Estrin, R.R. Muntz et R. Uzgales
AFIPS 73 (vol. 42 p. 725).

[21] Measurement of computer systems-An introduction
Arnold. F. Goodman
AFIPS 72 (vol. 41 - p. 669).

[22] Optimisation en classification automatique et reconnaissance des formes
E. Diday
Note scientifique n° 6. Supplément au bulletin de l'IRIA n°12- juin 72.

[23] Introduction à l'analyse factorielle typologique
E. Diday
Rapport de recherche n° 27. IRIA - Août 73.

[24] " Les méthodes de l'analyse de données"
"Algorithmes rapides d'agrégation"
J.P. Benzecri
Extraits de "Analyse de données - Tome 1 - La Taxinomie"
Laboratoire de Statistique Mathématique - 1971.

[25] Les méthodes de l'analyse de données.
M. Mathon
Note interne - Ecole des Mines de Saint-Etienne.

[26] "La pratique de l'analyse des correspondances"
"Distance distributionnelle et métrique du χ^2 en analyse factorielle des correspondances"
J.P. Benzecri.
Articles extraits de "Analyse de données - Tome II - L'analyse des correspondances". 1971.

[27] "Assembleurs et chargeurs"
D.W. Barron.

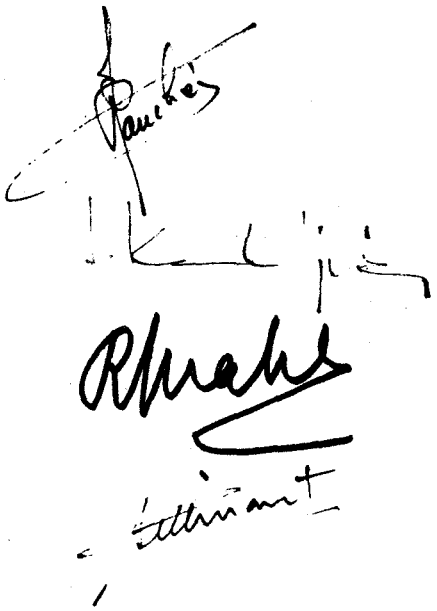
[28] "Le système CIVA : Un système de programmation modulaire"
J.C. Derniame
Thèse d'Etat -Nancy- Janvier 1974.

[29] "Les systèmes CP et CMS"
Alain Hauroux et Claude Hans - Centre scientifique IBM de Grenoble.
"Description externe de l'éditeur du Système conversationnel CMS"
Michel Adiba (IMAG)
Claude Hans (Centre scientifique IBM de Grenoble).

[30] "PDP 10 référence handbook"
Digital équipement Corporation.

[31] "TVEDIT"
John Prebus
Stanford University 1970.

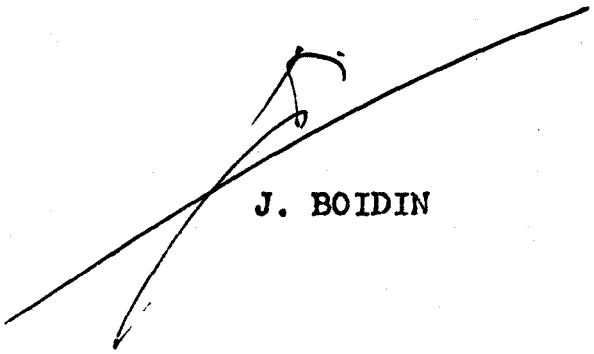
Vu.
Les membres du jury :



Vu et approuvé.
Le Directeur de l'U.E.R.
de Mathématiques.

Vu et accordé le permis d'imprimer.
Lyon, le 18 juin 197

Le Président de l'Université
Claude Bernard,



J. BOIDIN