ULANG- AN EXTENSIBLE USER-LANGUAGE

by

Imants F. Freibergs, M.A.Sc.

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

School of Computer Science
McGill University, Montreal

[redacted]

I. F. Freibergs
School of Computer Science
Degree: Ph.D.

ABSTRACT

ULANG - AN EXTENSIBLE USER-LANGUAGE

A new approach for implementing application systems has been developed. The properties for a generalized user-language, called ULANG, have been defined. This language allows a user of an application to communicate with the computer in a natural-language like way and with the terminology of his profession.

For the application programmer, ULANG is a tool that eases and speeds up the implementation considerably. It provides an "intelligent" interface between the users and the problem-solving logic. It is particularly well suited to interactive system implementation. For each user-request, this interface obtains the values of the required processing-parameters from the user's input, from prestored default-values, or otherwise, and transmits them to the processing routines. ULANG also provides an overall control and communication framework for all the processing routines of an application system.

ULANG can be used interactively or in a batch mode for command-language applications requiring a natural-language like user-interface.

SOMMAIRE

ULANG - un langage extensible pour les applications

Une nouvelle méthode pour faciliter l'implantation des systèmes d'information et de gestion a été développée. Les caractéristiques d'un langage de commande, appelé ULANG, ont été définies. Ce langage est destiné aux utilisateurs d'applications mécanisées diverses. Il permet aux utilisateurs de communiquer avec l'ordinateur d'une manière assez proche des langues naturelles et d'employer le vocabulaire habituel de leur profession.

Pour les programmeurs, ULANG est un aide de programmation, qui va simplifier considérablement l'implantation des applications. ULANG met à leur disposition un interface "intelligent " entre l'utilisateur et la logique du traitement. Pour chaque commande d'un utilisateur, cet interface obtient les valeurs de tous les paramètres nécessaires pour traiter la demande, à partir de l'information offerte par l'utilisateur, ainsi qu'à partir de valeurs préassignées. Ces paramètres sont transmis aux modules de traitement. En plus, ULANG rend disponible une structure de contrôle et de communication entre tous les modules faisant partie du système.

ULANG peut être utilisé d'une manière interactive ou traductive pour les applications qui emploient des langages de commande et qui nécessittent un interface proche du langage naturel.

ACKNOWLEDGMENTS

This thesis would not have materialized without the encouragement and the generous hospitality offered by Dr. Juris Hartmanis, Chairman of the Department of Computer Science at Cornell University, where the design phase of this work was carried out. Thanks are due to Dr. Pierre Robert, chairman of the Département d'Informatique at Université de Montréal for the support given during the implementation phase of ULANG on the CDC 6600.

The guidance and advice at Cornell University by Dr. Howard L. Morgan (now at the University of Pennsylvania) are gratefully acknowledged. Thanks are also due to Dr. Glen Bach of McGill University for his support throughout the doctoral program. Also appreciated are the helpful comments by Professors Harvey Abramson, David Gries, Alan Shaw, David Thorpe, and Jean Vaucher, who read parts of earlier versions of this manuscript.

Finally, the sustained support and encouragement by Dr. Vaira Freibergs of Université de Montréal is appreciated, as well as her valuable comments on the manuscript, which helped greatly to improve its organisation and its style.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

This dissertation makes a contribution to the methodology of designing and implementing interactive application systems. In particular, a new language and an efficient system have been implemented, which provide

a) general interfacing capabilities between users and problem-solving logic,

b) a data structure for processing parameters, and

c) an overall control and communication framework.

This research synthesizes the knowledge from several areas in Computer Science: language design, data structuring and system design into a new and useful tool, called ULANG.

In chapter 1 the communication and implementation problems of application packages are examined and and an overview of the solutions implemented in this work is given. This is followed in chapter 2 by a survey of other interfaces and implementation systems. In chapters 3 and 4 a user-language and the implementation system for applications are presented. Then in chapter 5 the design and implementation experience of the ULANG system itself is given, the capabilities of which are demonstrated on three application examples in chapter 6. The conclusions and suggestions for further research of chapter 7 are followed by two appendices A and B, the first containing a metalinguistic description of the syntax of the user and implementation languages herein provided, and the other containing complete FORTRAN IV source listings of the ULANG system.

CHAPTER 1 - INTRODUCTION

Computers have proven themselves able to carry out many tasks with efficiency and exactitude. This accounts for their widespread use. According to the annual survey by Computers and Automation (Berkeley (72)) computers have been used in over 2300 different fields of applications, such as engineering, finance, accounting, manufacturing, to name a few.

As the use of computers spreads into all fields of human endeavor, more and more applications are reaching an increasingly less sophisticated user base. A user here means the end-user of an application, that is an engineer, a manager, a clerk, or other, who in general is not familiar with computers and programming. An application is meant to be an integrated system, consisting of a collection of procedures, capable of performing a variety of tasks ; in many cases it is called an application package.

At present, there exists a dual problem of communication and implementation :

1) The communication problem : non-technical users have difficulty in communicating with the computer, unless this can be done in a language natural to the users' profession (Thompson & Dostert (72)). Computer here is synonymous with virtual machine, that is what the user "sees", namely the interface of the application package, with various levels of software and hardware behind.

2) The implementation problem : as the demand for application packages grows, implementers find it costly and difficult to supply software with flexible user interfaces(Sammet (69)). An implementer is a professional programmer or analyst, who is setting up the application and making it operational on the computer.

This work attempts to solve the communication problem by providing language facilities to the user and to solve the implementation problem by providing programming facilities to the implementer. The whole concept is called ULANG, an acronym for user-language.

ULANG is a way of quickly implementing interactive application systems. The user-interface aspects are clearly separated out from problem-solving functions. The interfacing functions are standardized and parametrized into an "intelligent" inteface module. The problem-solving activities are different for each application and can be adequately described by existing programming languages. In addition, ULANG provides a data-structure and a control framework, where the various problem-logic modules can be "plugged-in". In short, ULANG is an interface "generator".

Although systems with similar goals have been proposed before, relatively little has actually been accomplished (Sammet (72)). The present system has been implemented and proves that it is practical and efficient.

1.1 The communication problem

The exactitude of a computer, while being one of the assets of its performance, becomes a liability when the machine expects to receive instructions from humans with the same exactitude. This inherent man-machine communication problem has caused many frustrations, delays, and so called "machine blunders". Because of this, the instructing of computers has been relegated to programmers having the appropriate mental attitude. In the early days computers were mostly used by highly trained engineers and scientists, who could easily adapt themselves to a mathematical and symbolic way of communication. This is no longer true, as prospective users now include accountants, doctors, management staff, teachers, etc., who do not have the background nor the time to learn an artifical computer language.

These users communicate with the machine via application packages. It is therefore crucial that the user-interfaces of packages accept a language of communication which is natural for a given profession. The user-language for most applications is close to ordinary English (Thompson & Dostert (72)).

To solve the communication problem, the use of natural languages has been advocated for applications (Halpern (67), Sammet (69)), whereby a user would be able to define his problem and data to the machine in plain English, and then ask it to carry out various tasks, with additional information supplied as needed. For this reason a great deal of effort has gone into English sentence parsing and their syntactic structure representation by computers.

In practice, natural-language processing efforts have run into difficulties with semantics, or the difficulty for the machine to get at the meaning of sentences. Recent efforts by Dostert (71) and Thompson, Woods (70), and Winograd (71) have tried to overcome the semantics problem. At this stage their work remains at an experimental level and one may safely say that it is going to be a long time before general purpose natural-language processing systems become practical enough to be used in every day applications.

Yet the need for applications software is growing and cannot wait. A recent survey by Booz, Allen, and Hamilton (71) of the data communications applications for seven major U.S. industries indicates an annual compound growth rate of about 40 % for the first half of the decade. A good indicator of software needs is the forecasted number of terminals, which would increase by 600,000 in 1975 over 1971 levels. Similar views have been expressed by others. Jean Sammet (69) points out that at the rate at which hardware and

applications are both developing, the programmers may well have to outnumber the people who have problems to solve.

Working under the pressure of time to get more applications computerized, the implementers may have to impose restrictions on the format of the user commands in order to ease the job of the machine and of the programmer, at a corresponding expense on the part of the users. An example of a user interface, difficult to master, is the Job Control Language for IBM 360, of which it has been said that "the language is both complex and demanding of extreme accuracy. Mistakes can result in anything from annoyance to near catastrophe in processing" (Canning (71a , p. 7)). Another example is the STRESS structural analysis package. A frequent user complaint about STRESS is its rigid and demanding interface (Fenves (66)). The importance of ease of use for applications software cannot be overemphasized to assure that a package will be useful to the largest number of users (Sammet (69, p. 34), Nelson (73)).

1.2 The implementation problem

Assuming that the communication problem can be overcome by additional implementation efforts, what is the solution to the implementation problem ?

One solution would be to let the users become the implementers of their own applications, but for that they would need education and systems with natural-language processing capabilities beyond what is presently available.

General-purpose programming tools. Another solution, advocated by Frank (68), Ross (67), and others, is for the programmers to rely more on general-purpose programming tools, which simplify the implementation

job. In practice, this has been the trend over the years. The implementers rely more and more on general purpose packages to perform those tasks which are not directly related to the logic of the application. Successively the programmers have become familiar with the use of different types of programming aids :

i) Compilers for problem-oriented languages, which have made the description of the problem-logic less machine-dependent.

ii) Operating systems, which have relieved programmers of the details of input and output, of interrupts, and in general of machine operating details.

iii) General-purpose file-management systems (IDS, ASAP, Mark IV), which have facilitated the task of data-management.

The work presented in this thesis adds another general-purpose programming aid to those already available. This is in agreement with the view expressed by Halpern (68) :

> "The principle by which progress has traditionally been made in programming technology is that of the replacement of the special by the general. Programmers discover from time to time that they have been repeatedly rewriting substantially the same routine for one job after another ; when the underlying identity of these several versions becomes apparent, a single general routine, needing nothing more than suitable parametrization to adapt it to each particular application, is produced to replace all of them".

From the implementer's point of view, a computerized application usually consists of three parts, as shown in Fig. 1-1 : a man-machine interface, processing-logic of the problem, and a file-management system.

The processing logic can be adequately described by algorithmic languages such as PL/I, Fortran, Cobol. The logic is so different for each application, that it would appear difficult, at this stage, to provide

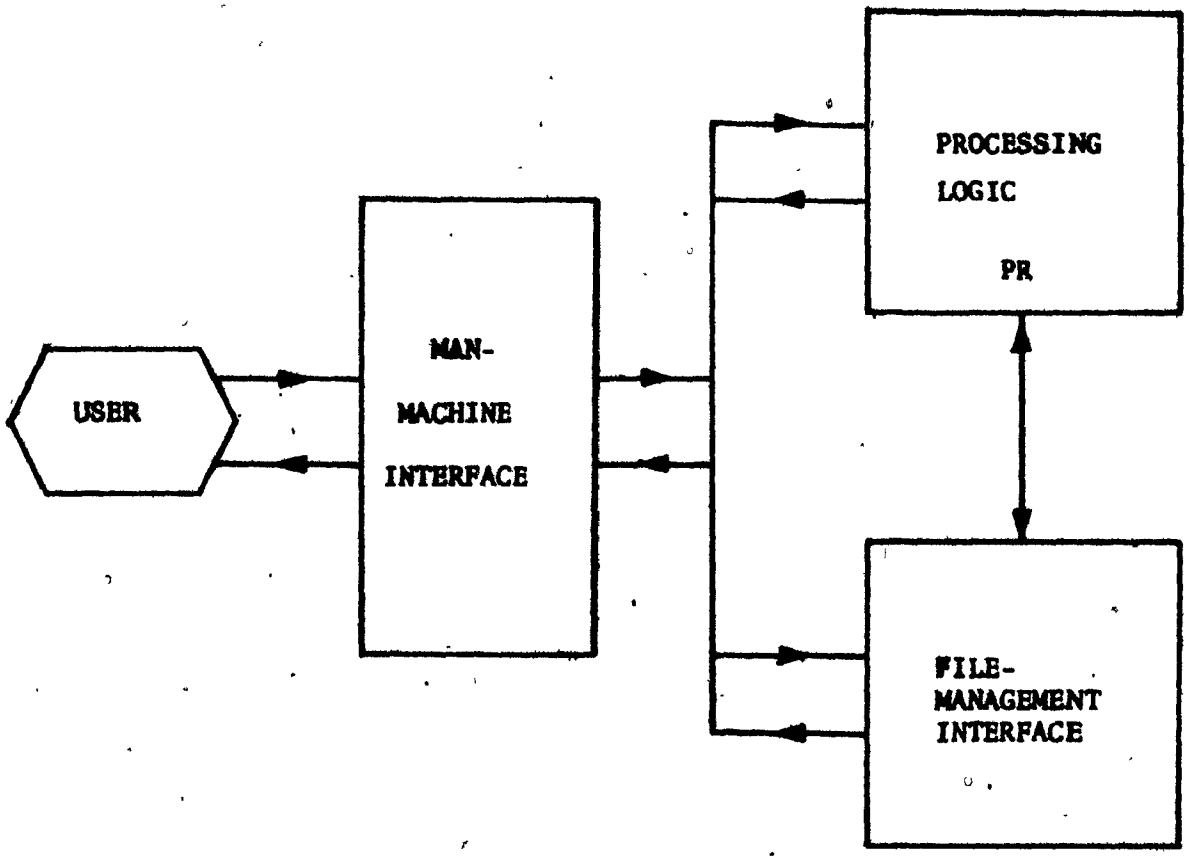


Figure 1-1. Components of an application system.

more generalized, yet practical tools to further automate this aspect of application writing. This does not imply however that no further progress is desirable for algorithmic languages. Evidently, all advances in programming methodology, such as structured, modular, or top-down programming, help the application programmer to systematize and to improve the quality and efficiency of his algorithmic activity.

An alternative to the algorithmic approach would be to ask the user to define his problem and his requirements in a "problem statement language" PSL and then have a "problem statement analyser" PSA decompose these and decide on the best method of achieving them. This is the approach taken by the ISDOS project under D. Teichroew at the University of Michigan (Hershey et al. (73)) for business information system applications.

General purpose modules can be used for the other two components of an application. A number of general-purpose file-management systems are already available (ASAP, Mark IV, TDMS). On the other hand, there is little evidence of generalized man-machine interfaces having been implemented, specifically designed for application packages (Sammet (72)). In this research such an interface generator has been implemented.

Greater use of generalized programming tools, such as file-management systems and user-interfaces will simplify the implementation job by allowing the programmer to concentrate only on the part which is different for each application, namely the processing-logic.

Empirical support for a generalized interface. When analyzing the total implementation effort for an application package, the author's own experience, corroborated by that of other implementers (Homa (70)), shows that well over 50 % of it will go into implementing input and output

interfaces between the user and the processing-logic proper. The percentage is even higher for interactive time-sharing applications. This is understandable, since it is the quality and success of the man-machine interface which in the end determines the acceptance of the application by its intended users. As a result, the implementers spend a great deal of effort in deciding on the terminology, syntax, and semantics of the interface, and in programming it.

Since 1965, the author has participated in the development of several application-systems, both for batch-processing, and for time-sharing, e. g. the AMECO (70) structural engineering design system and the BCS (70) financial portfolio evaluation package. It was observed that although the terminology and the processing-logic for each application were different, the man-machine interfacing requirements were about the same. It seemed that it would be possible to set up an efficient front-end, valid for many types of applications, having the same basic structure, but accommodating different terminologies. This has been now achieved.

1.3 A formal model

The user-interface translates user-requests into calls on appropriate procedures and activates certain parameters. By considering this translation process in a more formal manner we can show the important steps involved therein.

The model. Given a user-language L for some application, with its associated rules of grammar G, the user U formulates a request r, by applying the rules of grammar G to produce a sentence I to the computer

$$U(r, G) \rightarrow I$$

This input I is processed by the application system function

C and produces results R . The function C may also require some additional stored file data F ,

$$C\ (I,\ F) \rightarrow R$$

There is a tradeoff between the complexity of C , which represents the work to be done by the machine, against G , the complexity of the rules of grammar, which represents the work and memorization to be done by the user U . From a user's point of view, G should be at the lowest level of complexity G_0 , which represents the minimal set of rules and conventions for U to remember. User experience in this respect is summarized by Sammet (69, p. 32) : "If a language has many specific and strict rules about spacing and punctuation, there is more of a tendency for error in writing the program". Nelson et al. (73) report user experience with a statistical language, STATPAC, designed "to aid users by relieving them of all unnecessary blunders". But the package is run on GE computers with a separate system language "which is quite cryptic ... Roughly half of the user problems come from errors in these cards". Software firms have found it worthwile to develop user interfaces that use "clean, easy to read language", which is then converted to IBM's JCL. Several of these (JOL, JCL-OMATIC) are listed in the ICP Software Directory (74).

We decompose the processing function C into three components, at each step producing some intermediate output, which then becomes input to the next step. The components of C are called SYN, SEM, and PR. SYN is a lexical and syntactic recognition function, SEM is a semantic interpretation function, PR is a computational function. Two storage functions D and T are also introduced. D is a dictionary of terminology, T is a template of valid processing parameter names and values defining the content for PR.

The first step is the lexical and syntactic interpretation of the input sentence I. The words used in I are recognized and identified. An output string A, consisting of a set of atoms {a} is produced by applying the function SYN to the input string I and by making use of D for the terminology. This is shown as

$$SYN\ (I,\ D) \rightarrow A$$

The next step is the semantic interpretation of A. Here an attempt is made by SEM to assign meaning and values to the atoms of A by matching them with the parameter information in the template T. The function SEM outputs the set of active parameters P, such that $P \subseteq T$. This step can be represented as

$$SEM\ (A,\ T) \rightarrow P$$

Finally the computational function PR is applied to the parameter string P ; it may also use some additional file information F to produce the results R, i.e.

$$PR\ (P,\ F) \rightarrow R$$

We have not specified here how the file data F is obtained; this could be part of the PR function, but preferably the files are handled by a separate file management package.

To summarize, we have decomposed the formulation and processing of a user-request r into four logical steps, shown in Table 1-1.

The processing function C is equivalent to the successive application of SYN, SEM, and PR, i.e.

$$PR(SEM(SYN(I,D),T),F) \rightarrow R$$

TABLE 1-1 FORMULATION AND PROCESSING OF A USER-REQUEST r

Step		Function	Responsibility of
(1)	U (r, G) → I	Formulation	User
(2)	SYN (I, D) → A	Recognition	Interface
(3)	SEM (A, T) → P	Recognition	Interface
(4)	PR (P, F) → R	Processing	Computer

The responsibilities for the steps (1) to (4) of Table 1-1 are divided between the user U and the computer. The first step, namely the decision as to what request r to formulate, is clearly the user's responsibility. Likewise the last step, doing the logic, once all the necessary parameters have been supplied, is a task for the computer.

The relative emphasis on the user or on the machine in steps (2) and (3) in between may vary considerably as to how much latitude is allowed in the terminology and the syntax, and how much of the semantic interpretation should be done by the user. They also vary widely for different application systems, depending on the ability of the implementers to produce a workable system within a limited time and budget, as well as on the intended use of the application.

Levels of user-grammars. At the lowest level of user-orientation, the user has the responsibility of adhering to strict syntax rules, such as fixed-field format, rigid delimiter structure, and exact keyword spelling and position in the sentence. He is also responsible for supplying all the parameters each time, in a prescribed sequence, with all their attributes. An example of this approach is the Job Control Language for the IBM 360, which is only understood by experienced systems programmers (Canning (71a)). The user has to assume the functions SYN and SEM and remember the contents

of the dictionary and of the template. The rules of grammar corresponding to this level of stating requests can be denoted by G_2, where

$$G_2 \equiv \text{SEM (SYN } (I_2, D), T)$$

The user formulates his request r as an input sentence I_2, written according to the rules of G_2,

$$U\ (r, G_2) \rightarrow I_2$$

and then, as before,

$$PR\ (I_2, F) \rightarrow R$$

At the highest user-oriented level, the user is free to formulate a request in almost any way ; an attempt will be made by the system to interpret his request in proper context and to supply missing information if necessary. This corresponds to a G_0 level grammar, with as few artificial rules as possible beyond the grammar rules of natural languages, such as English. Of course, all the intermediate levels between G_0 and G_2 are possible.

Granted the desirability to provide the user with G_0 type of grammar rules, it is the implementer's responsibility to supply the equivalent of the functions SYN and SEM and of the tables D and T for each application system. He has to construct a processing function PR_2 such that,

$$U\ (r, G_0) \rightarrow I \quad \text{and then} \quad PR_2\ (I, F) \rightarrow R$$

Here PR_2 is equivalent to C, i.e.

$$PR_2 \equiv C$$

Place of ULANG in the formal model. The interfacing functions SYN and SEM are basically the same for many applications, whereas the processing-logic PR is not. The purpose of ULANG is to take over the functions of SYN

and SEM - to build them into an interface, valid for many types of applications. Variability between different applications, or different procedures of a same application, is provided for by the tables D and T.

Making ULANG as part of an application-system, the user expresses his request, as before

$$U\ (r,\ G_0) \rightarrow I$$

this is accepted by the ULANG preprocessor,

$$ULANG\ (I) \rightarrow P$$

and then, as before

$$PR\ (P,F) \rightarrow R$$

which gives the equivalence relation

$$ULANG\ (I) \equiv SEM\ (SYN\ (I,D),\ T)$$

An electrical analogy. To use an electrical analogy, the ULANG system can be viewed as a plugboard, or common frame. The user sees only one side of the board, with labeled switches and buttons (elements of user-language), which he may activate. Each of the switches and buttons eventually emerges as a socket on the other side of the board. The implementer works on the other side of the plugboard, where he plugs in units of processing-logic (procedures). The prongs of the logic units are analogous to parameters; the implementer is responsible for matching them with the sockets of the board, and also for labelling the switches and buttons properly on the user's side.

The plugboard remains the same for different applications. For each application the switches and buttons are relabeled on the user's side. The implementer makes up the appropriate logic units and sets up the prong and socket correspondance. The plugboard itself has

certain built-in active logic units (ULANG system-functions, e.g. for expression processing).

Place of ULANG in system software. It helps to situate ULANG within the software levels of a system, as shown in Fig. 1-2. All processing is eventually done by the hardware at level 0, enhanced by an operating system at level 1.

A user-program (level 3) generally has access to functions of levels 0 and 1 via software at level 2, such as compilers and loaders. A user interacts with the system only at level 4, through the data and commands which the user-program at level 3 accepts (see Fig. 1-2a).

The aim of ULANG is to facilitate the work at levels 3 and 4, namely to replace parts of the user-programs by standard modules and to ease the formulation of user-commands and data (see Fig. 1-2b). It is not meant to be a substitute for an operating system or for a compiler (level 1 and 2 functions).

1.4 The user-interface: practical aspects

In section 1.1 the communication aspects were considered in general. Now we focus on specific user-needs and we describe the user-language implemented and its scope.

1.4.1 Observations about user needs

As pointed out by Halpern (67), full English text processing and understanding capabilities are not needed for most computer applications, as would be the case for, say, language translation. The area of discourse is limited to the framework of the application. Moreover the user knows that he is talking to a machine, so that he would not expect to have to use well formed literary sentences. On the contrary,

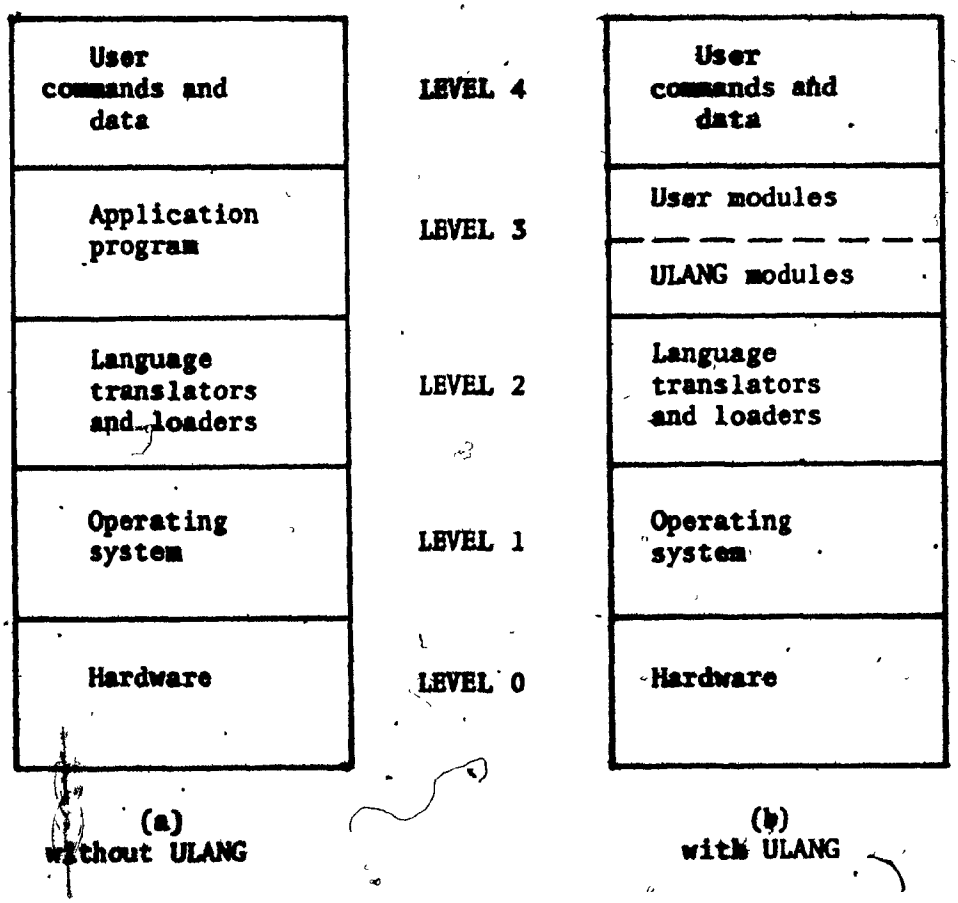


Figure 1-2. Software levels in a system.

he has the right to expect to be concise, since this represents less effort on his part, even to the point where some of the information of a request remains implicit (Joyce (72), Weiss (70)).

What the user does not like, is having artificial vocabulary and rigid syntax restrictions imposed on him, which place heavy demands on his memory. He likes to use and define his own "active" language subset, appropriate to the application. That means that the user has to be able to define his own vocabulary and to make requests in a reasonably natural way (Halpern (67), Joyce (72)).

The user may expect the machine to "memorize" for him a certain amount of information about a request, so that if he does not supply it explicitly, then the appropriate processing context is supplied automatically by default. To avoid uncertainty, the machine has to echo back its explicit interpretation of the request, if asked to do so (Joyce (72)).

For most applications, the user wants the computer to carry out for him some activity which he knows to be within its capabilities. For this, the imperative mode, or the equivalent query mode, are adequate. The declarative mode can be restricted to combinations of previously defined items, as for arithmetic and logical expressions, or for limits and conditions. The users are not expected to define their own applications and data-bases in natural language and then make unpredictable requests. It is assumed that they will be satisfied with being able to make specific requests in a reasonably natural way of an application which has been previously set up by programmers (Weiss (70)).

It is further observed that the largest number of current applications are of the "command-language" type, where different acti-

vities have to be carried out in a sequence specified by the user - requests. The canonical form of all user-requests can be reduced to :

COMMAND, $PARAMETER_1$, $PARAMETER_2$, ..., $PARAMETER_n$

which agrees with the conclusion reached by Halpern (68). This does not mean that the command has to be the first word of a sentence, nor that it has to be part of every sentence. Any generalized interface has to be able to satisfy the above user-language needs.

1.4.2 The user-language implemented

In this work a user-language has been implemented, attempting to answer the user needs of the previous section, and to demonstrate the scope of the system. A sentence of this language consists of a command-word, and of a number of parameters, that are necessary to define the context for the task specified by the command. Typical parameters could be data-names, values, expressions, ranges for values, or conditions. The language is described in more detail in chapter 3. This user-language is of general usefulness, since it is not limited to any single application, but is valid for a whole range of applications, as outlined in the next section.

The user-language implemented can be used as a front-end to applications where activities have to be performed in sequence. Each activity is initiated by a keyword, usually a command, followed by data, or parameters, which are transmitted to the problem-solving procedures selected by the command. These procedures are either written by the implementer, or else provided by the ULANG system. There is no predefined vocabulary ; the basic terminology is defined by the implementer when the application is set up. Users may later add their own synonyms.

Provision has been made for two modes of usage :

a) An immediate mode, where each request is immediately executed. This is most appropriate for an interactive environment.

b) A delayed mode for building up more complex requests. In this mode all commands are analysed and stored as successive steps in the Command Table. They would be only executed when the EXECUTE command is given.

For applications, where the user-requests cannot be reduced to the form COMMAND, $PARAMETER_1$, ..., $PARAMETER_n$ different systax analysers have to be provided. It may even be necessary for several analysers, of varying capabilities, to co-exist, each handling a different sublanguage of an application.

This does not in any way lessen the usefulness of the ULANG concept. The lexical and syntactical analysis modules each account for only 10 % of the Fortran statements constituting the ULANG system. The remaining 80 % of the statements make up the modules for data structure organisation, for input semantics, and for value-functions. The different facets of the system are summarized in section 1.5.2 below. The syntax analyser implemented is described in section 5.4.3. A comparison of ULANG with translator writing systems is given in section 2.3.2.

1.4.3 Applicability of the user-language implemented

The type of user-language described in the previous section is valid for many types of applications.

A very large proportion of all computerized applications involve queries, updating, and reports from data-banks of various kinds. ULANG is well suited for this type of application, particulary in an interactive mode. Here the terminology and the actual data used are

different, but the underlying processing and the nature of the requests are similar. Some examples : customer accounts, policyholder accounts, medical histories ; inventories, skills catalogues, securities positions ; student records, medical records, accounting records.

The user-language could also be used for applications which require a flexible user interface, with or without data-banks, as for parametric studies and simulations for decision making, performance calculation studies, optimization studies, cash flow studies, investment policy decisions, engineering design, statistical packages, or systems for solving numerical and scientific problems.

In certain situations there would be little advantage in using a generalized interface. These include one-time programs, applications not requiring much interaction with the user (i. e. where the amount of input is small, or where there is no need for flexible input), and situations where there is not much variation in the possible interaction. It is also not suitable for applications where full natural-language processing capabilities are required, as in language translation, or in deductive question-answering systems.

1.5 The implementation system : practical aspects

In section 1.2 the implementation problem was discussed in general terms. Now we look at specific needs of the implementers of application packages and we describe the functions assumed by the ULANG implementation system.

1.5.1 Observations about programmer needs

First, we assume that a programmer is still needed to set up

an application and to define the types of requests which the users will be able to make. The programmer is also responsible for providing the processing logic for the application, either by coding it,or else by assembling the appropriate subroutines from a standard subroutine library.

A study of business application package utilisation in 100 French firms was conducted by a team from the Department of Computer Science of the University of Montpellier for the French Ministry of Industry & Research (Etude (74)). One of their conclusions was that an application package will be succesful within a company only if it has the support and cooperation of the programming staff. In fact, the most succesful packages have been those of the programmer-aid type, such as automatic flowcharters, or report generators. This conclusion is corroborated by an EDP Analyser survey about application package usage in 44 U.S. firms (Canning (71)). The French study also concluded that the firms would rather use software, which helps their programmers to implement packages, tailor-made to their specific requirements, as opposed to generalized packages, which require standardisation and modifications in their operating procedures.

It will help the implementers to have a programming aid which relieves them of the details of interfacing the problem logic with the users and which looks after the lexical and the syntax analysis of user-requests. That task could in theory be also handled by other language-extension tools, such as macro-processors or compiler-compilers, but, in practice these are ill suited and rarely, if ever, used for application programming (Sammet (72)). They have been designed to implement known programming-languages rather than as a tool for developing new user-languages.

Lexical and syntactical handling of input, however, represents only part of the interfacing job. The other part is semantic handling, namely assigning of meaning to the tokens (words, numbers, operators), which make up the input string. The semantic handling is a matching process of the input tokens with the possible parameters for the corresponding processing-logic routines. In an English-like interface this presents a difficult task, since the tokens do not have to be in any particular order. Some of the required processing parameters may even be omitted on input, in which case default values have to be found. The validity of the input tokens has to be checked, for instance to see if numeric values are within allowable ranges. Mode conversion may also be necessary.

Few programming tools have been reported which do the input semantics for application packages in a systematic way. The GIS data base management system checks the validity ranges of updating information against those specified with file definitions (GIS (68)). Recently, in the programming language PASCAL, variables such as integers can be given certain ranges. However for applications not using GIS or PASCAL the programmer has to incorporate these tasks as part of his logic routines, together with default values of parameters, and with the logic for parameter matching. For most applications, documentation about parameter structures and values is sketchy and any changes in the structure and values are difficult to make (Ross (67)).

From an overall system point of view, a general framework would be helpful to the implementer, to coordinate and to call into action the appropriate problem-solving procedures and to transfer infor-

mation between them. Such a central program would be the main-program of an application, and would look after the branching off to the appropriate processing routines. The programmer could build up the application in a modular way, by adding successive logic modules, corresponding to the various user-requests.

In order to be used by implementers, a system has to be simple enough to use and to understand. The specifications for setting up an application and for accessing parameters within the logic procedures written by the programmer have to be clear and concise. There is a certain reluctance among application programmers to use "super-tools" claiming many capabilities. By experience, such tools are too complex to understand and to adapt and they require too much of the machine's resources to be of practical value (Etude (74), Canning (71b)).

For reasons of economy, the interface cannot be too demanding on core memory and processing time. Both of these systems resources are expensive, and moreover all user-programs compete for them. Usually programs with high memory and processing time requirements receive a low priority, which would result in an unsatisfactory response time for the users of such an application. This means that the interface cannot be embedded in a large complex system, but it has to be modular and self-contained.

A serious problem is the transportability of applications from one machine to another. For organizations which buy computer services from computer utilities, there is a great variety of competitive services available, and machine independence makes it possible to use the service which offers the lowest cost. For organizations

using in-house computers, the transportability problem still exists, since the average life span of a computer system, before its replacement by another one is less than four years.

1.5.2 The system implemented

In this thesis a pragmatic approach has been taken. A system has been implemented that is reasonably simple and structured to begin with, but can be easily modified and extended. A number of important features are at the disposal of the application programmer:

1) A lexical analyzer, based on finite state automata techniques. The meaning of input characters is localized within one table and can be easily redefined. The finite state approach permits the removal or addition of lexical processing states.

2) The vocabulary of the application is kept in a user-dictionary, which is automatically created when the application is defined. The vocabulary can be augmented and synonyms can be added by simple ULANG system commands.

3) A syntax analyzer, valid for the user-language described earlier. For languages with a different syntax, this analyzer can be modified or replaced.

4) An input-semantics module checks, matches and converts modes of input parameters, and supplies default values. It sets up the proper context for processing the user-requests.

5) An underlying data structure for the parameters, capable of accommodating various application and data types. A ULANG setup-language allows the implementer to easily define this structure.

6) Table building routines which assemble the active parameters of a user-request, and functions which transmit their values to problem-solving procedures. The value-functions can be customized by the implementer and their use simplifies the program logic considerably.

7) A control framework that coordinates the processing sequence and facilitates the addition of new operations.

The ULANG system was first implemented on the CDC 6600 in FORTRAN IV, by taking special care to parametrize the system to make it easily transferable to other systems. It has proven to be efficient in core memory (10500 words) and in processing time - it costs only one cent to process an average user-request. Several actual application examples are presented in chapter 6.

The transportability of the ULANG system has been demonstrated by transferring it to IBM 360 computers. The only changes necessary were due to differences in internal character representation (BCD vs. EBCDIC), and in direct-access procedures. The required changes are well isolated within a few tables and subroutines.

CHAPTER 2 - OTHER SYSTEMS

2. 1 Other interface systems

Since ULANG is designed with the capability of supporting user-languages that use a subset of English, other systems having user-interfaces with similar characteristics are surveyed next. These include systems which communicate wich data-files in natural language, and command lang-uages. ULANG draws on the experience gained with such systems.

Natural language question-answering systems. Systems of this type take English sentences as input, subject to various constraints, and attempt to answer questions from a data-base, which is stored in form of a list structure. Developments in this area have been reviewed by Simmons (65, 70). The ambitions of the question-answering systems are set high, namely to provide a general framework for storing, "understanding", and intelligently retrieving unpredictable information.Their organization is correspondingly complex ; at this stage they remain as large and slow experimental systems, dependent on LISP for the associative storage needed for the syntactic and semantic information they use.

In practice, it is difficult to say how well these systems would perform beyond the very limited areas of discourse and subsets of English on which they are based and which hve been reported in the literature. The grammar of the language subset corresponds more to the data structures chosen than to English. For instance the BASEBALL system of Green (63) was limited to the description of specific baseball games, and the SIR system by Raphael (68) was limited to the recognition of a small number of family relationships. The ELIZA program of Weizenbaum (66) imitates

conversation with a psychotherapist. It operates on the basis of recognizing certain keyword patterns.

Although doubts have been expressed by Guilliano (65) and Kellogg (68), as to whether these experimental systems can be generalized to other applications of significantly different nature, they have nonetheless proven that a limited English subset interface can be succesful within a given application. This fact is used for defining the ULANG interface.

Kellogg (68) has developed an on-line data base management system which, among other things, accepts a less restrictive English subset than the previous systems. Kellogg has added an important feature : an anomalous input sentence, lying outside of the acceptable English subset, causes the compiler to construct an appropriate feedback message to the user. This makes the user aware of the current limitations of the subset. User feedback is an important aspect of the ULANG interface.

All of the above natural-language systems have been able to handle only subsets of English because of the complexity of the full grammar and lack of adequate machine representation for it. If an interface is to be kept reasonably simple, it has to recognize this difficulty and content itself with a subset of the language, chosen in a suitable way:this is done for ULANG.

More recently, several systems are in the experimental stages, attempting to deal with the full complexity of the language. Thompson (69) and Dostert (71) at Caltech are developing the REL system to facilitate conversational interaction with highly interrelated data-bases, used by social scientists. It has powerful English grammar facilities, permitting individuals to communicate with the computer in a fairly

natural language. It has its own operating system closely integrated with a single language processor. Woods (70) and Winograd (71) are focusing their efforts on the representation of the English grammar as transition networks and as procedures, respectively.

A common characteristic of all the natural-language systems is that, because of their aims of a high level of generality at being able to accept and store declarative information, they remain rather complex, list-language embedded systems. Since the user of an average application has no need to significantly alter the data-base structure, most of the semantic and data-base organization problems can be avoided and a much simpler implementation can be achieved, as shown in this work.

Data-base inquiry systems. Interactive data-base inquiry systems aim at providing a system for creation and maintenance of master files and an inquiry language for interrogating their contents. A number of data-base systems are in existence, for instance ASAP (70), or the inquiry system described by Holland (70). The user-interfaces of such systems are usually natural-language oriented, usable by a clerk or a manager.

The type of inquiry allowed is simple and restricted to operations on the contents of the master file. Some definitional facilities are provided for naming fields and records of the file, or for performing arithmetic operations on the items of the file, prior to displaying them.

The user-interface is an integral part of these systems, and it has been specifically developed for it. By using a generalized interface, the implementation of such data-base inquiry systems would be greatly facilitated.

Numerical-problem stating languages. User-languages have been proposed for stating scientific and numerical problems in standard mathematical notation for formulas and equations, and in plain English for organizational instructions. Examples of such systems are: the MIRFAC scientific compiler proposed by Gawlik (63), the numerical problem description languages NAPSS by Rice & Rosen (66), and POSE by Schlesinger & Sashkin (67). Such systems have been called "non-procedural" since the user does not have to specify a method for their solutions, but the system itself supplies some general purpose method from a subroutine library.

Here again, a generalized user-interface could be used, which the implementer could tie to a numerical method subroutine library or to a statistical library.

Command-languages for specific applications. A number of command-languages have been implemented in specific areas with natural-language like user-interfaces, such as AMECO (70) for structural engineering, or Synthesis & Analysis by Samuel (69) for debugging in a time-shared environment.

ULANG could be used for these and other applications to provide the user-interface. At the same time this would speed up the implementation process and would insure that future growth and changes in the language can be more easily handled.

2.2 Other methods of implementation

Presently a number of methods exist for implementing application packages. It is appropriate to compare them to the ULANG approach, keeping in mind the place of ULANG in the system software hierarchy, as shown in Fig. 1-2(b).

Interactive systems. Conversational systems such as APL, ITF PL/I, or BASIC, may be used to create interactive applications. In the same sense any programming language can be used to create batch applications. If this approach is taken, then the programmer has to figure out and implement for each application the interfacing, data organizing and context checking, and control functions assumed by ULANG.

Of course, some systems make these tasks easier than others, e.g. APL, nonetheless the logic has to be figured out and tested for each application. The principal aim of this work is to avoid this continuous duplication of effort. ULANG itself could be coded in any of these interactive languages.

System command-languages. Most of the present operating systems have command-languages of the form

< command >< param1 > < param2 > ...

This is also the form of user-commands. Examples of system command-languages are the Job Control Language (JCL) for OS/360 and the Cambridge Monitor System (CMS) for IBM 360/67 (IBM(69)).

CMS permits setting up of a series of other CMS commands and then filing them under a < filename >, which can be subsequently all executed by simply typing

EXEC < filename >, < arguments >

The CMS commands may have symbolic arguments, designated as &1, &2, etc., for which the real arguments of the EXEC command are substituted at run-time. This is somewhat similar to the REQUEST mode operation of ULANG, with input value substitution for the default values of the parameters.

Theoretically, it should be possible to adapt, say, the JCL command processor to handle new user-commands for applications, obeying

the same syntax. In practice this is almost never done for several reasons. It would be extremely difficult for an applications programmer to obtain sufficient documentation and familiarity with the operating system modules in order to be able to incorporate the necessary changes to handle the additional commands. Also most installations are extremely reluctant in allowing modifications to their operating systems software (level 1) in a production environment.

However, even assuming that these practical difficulties could be overcome, the language facilities available are far below the level of flexibility desirable in a user-language. The parameter sequence is rigid, missing parameters have to be accounted for, and punctuation is usually critical. It is a well-known fact that job control languages are difficult to master and lead to frequent syntax errors. This work tries to overcome precisely some of these deficiencies. In fact ULANG could be used to advantage to implement more easily useable system control-languages than some of the existing ones.

Another disadvantage of tying an application into some control language processor, such as CMS, is its reduced mobility. Whereas a level 3 program (see Fig. 1-2) is easily transportable, a level 1 system is very closely tied to its level 0 hardware.

Systems using subroutine calls. In some application systems the user himself states his requests in a programming language, say FORTRAN, as a series of subroutine calls with predefined arguments, of the form

CALL < command > (< arg1 >,< arg2 >,...)

An example of such a system is the Investment Analysis Language (IAL) described by Dmytryshak (72), which has been used in forecasting

and corporate planning applications. IAL is a FORTRAN subroutine package for Management Science and Financial Analysis, to be used by analysts having knowledge of programming as well as of their special field.

While this approach may be advantageous to a fairly sophisticated user who knows programming and who can take advantage of the capabilities offered through the programming language, it has several disadvantages to the average, less knowledgeable user. First, he would need to master some programming concepts, and to obtain some familiarity with the compiler. Moreover, he would be restricted to a rigid command and parameter structure, subject to the criticisms given for system command languages.

As Sammet (72) points out, while subroutines undoubtedly serve a very useful purpose in making additional functional capability available, they are no substitutes for a language. She gives examples of application areas, where subroutines for specific tasks had existed for a while, but people did not "rush to use the computer" until a language became available (Sammet (69, p. 733).

2.3 Systems for language extension

One of the problems of computerizing applications is the development of specialized languages tailored to particular users. Theoretically, these languages could be implemented by extending programming languages to accommodate the special requirements of each application, although no one seems to be doing this in practice (Sammet (72)). Programming tools, described in the literature, which claim this capability, are macro-processors, translator writing systems, and extensible languages.

2.3.1 Macro-processors

Brown (69) defines a macro-processor as "a piece of software designed to allow the user to add new facilites of his own design to

existing software". The purpose of macro-processors is to provide the programmers with a precompiler, which recognizes certain extensions to the base-language and replaces them by some previously defined replacement text in the base-language. The modified text, together with portions of intermixed base-language text, becomes input to the base-language compiler.

The emphasis of macro-processors is to simplify and to expand the existing programming facilities, e.g. a shorthand notation for COBOL, or high-level loop control statement in assembly language, or mathematical function definition, such as factorial (n).

ULANG is different from macro-processors in that it is not a precompile facility, but rather a run-time preprocessor and controller. Its emphasis is not on simplifying base-language programming by text replacement, but rather on providing the run-time procedures with all necessary parameters and values.

In macro-processors, the macro-definition and macro-call facilities can be intermixed and nested within each other, together with base-language text, which results in a complex system. In ULANG, user-commands are the equivalent of macro-calls. These are issued by the end-user of an application at run-time only. The equivalent of macro-definitions for ULANG is the set-up subsystem, which is used by the implementer only. This allows the separation of the definition and of the call functions into two separate and simpler modules.

From the definition point of view, ULANG is different from macro-processors in that, instead of having to define replacement instruction text, the equivalent of macro "code body" has to consist of parameter patterns and their values, which have to be matched against input supplied by the users at run-time. In other words, the emphasis is on data structures

rather than instruction text.

2.3.2 Translator writing systems

During the past decade, development of programming languages has received a great deal of attention, in particular syntax-directed compilation techniques. In order to automate the compiler writer's task, a number of automatic translator-writing systems (TWS), or compiler-compilers, have been introduced. The state-of-art of compiler-compilers has been surveyed by Feldman & Gries (68), and by Cheatham & Standish (70).

Compiler-compilers and ULANG are programming tools with similar aims : to automate the programmer's job ; in one case the implementation of programming systems and in the other case the implementation of application systems. Similarities can be expected to exist between the two ; these are examined in this section.

While earlier TWS were constructed in an ad-hoc fashion, lately the compiler writing task has become more formalized, and its different aspects have been isolated and studied. Some of the concepts used in compiler-compilers and the lessons learned are valid for other, non-translator, tasks and for user-languages in particular. Although most compiler-compilers have been built specifically for programming languages, some systems were designed at the outset to be general purpose syntax-directed symbol processors, such as the AED system, described by Ross (67), and the APAREL system of Balzer & Farber (69).

Compiler-writing can be considered simply as another computer application, consisting of a syntax phase, a semantics phase, and a processing-logic phase PR. The PR phase is in this case code-generation for some particular machine. In ULANG the PR phase is to be done by the implementer, since it is different for each application. On the other hand, if the application is restricted to compiler writing, it is possible to

provide a generalized "semantic-language" for describing code-generation, compile-time data structures, tables, and optimization. Such a semantic-language ties in closely with the syntax phase, and distinguishes compiler-compilers, such as the one of Brooker et al. (63), or FSL of Feldman (66), from other syntax-directed symbol processors. This semantic-language makes compiler-compilers unsuitable for non-translator applications.

Lexical analysis. The syntax phase can be broken up into two parts : a lexical analyser, and a syntax analyser proper. In early systems the lexical processing had been incorporated as part of the syntactic-analysis programs. The advantages of separating the two functions have been pointed out by Johnson et al. (68), and by Cheatham & Standish (70). Experience has shown that a large portion of time of any symbol processor is spent in lexical analysis of the input strings. Separating out that function allows this problem to be attacked more effectively. It also allows the investigation of lexical properties of languages per se, and the development of systems for efficient lexical analysis. Separate lexical recognizers allow to accept input from different devices (terminals, graphics, voice) and to produce the same token string for the syntax parser.

ULANG draws on the experience gained with lexical analysers in translator systems. It has a separate lexical phase, which is based on finite automata and makes use of the hardware representation of a character as an index in a class membership table, as advocated by Cheatham & Standish (70), and Gries (71).

Some systems, such as AED, have specialized languages for specifying regular expressions and class membership of characters. ULANG does not have such a meta-language at this stage, but class membership of characters can be easily accessed and changed directly within a table of the scanner phase. This is less general but more efficient (Lecarme (73)).

In ULANG, the lexical analysis has been extended by adding a post-scan phase, which does a Dictionary lookup for names of the input string. This approach allows to look after synonyms, noise-words, spelling errors, features not essential for programming languages, but important to user languages. The post-scan phase could also handle text expansion and replacement, e.g. the word ALL might be expanded to FROM FIRST TO LAST BY 1. This is somewhat similar to the macro-processing phase, between the lexical and syntax phases, of the AED system (Ross (67)).

Syntax analysis. Syntax recognition has been the most extensively studied aspect of TWS. One of the characteristics of such systems is the presence of a meta-language by which the syntax of the language to be developed, called target-language, can be described. Usually the target-language description is in a format close to the Backus-Naur Form (BNF). The BNF-like description is input into a constructor phase of the compiler-compiler, which produces either a parsing algorithm in the form of a program to be executed, or else a set of tables which are combined with the basic parsing routines of the constructor to form the syntax phase for the target-language processor.

This approach has the advantage of providing a ready-made syntax phase for the target-language processor. It also permits experimentation with the grammar of the proposed target-language and removal of ambiguous forms. Such features are of considerable interest to user-languages as well as to programming-languages.

In spite of these advantages, automatically constructed syntax recognizers have not, until recently, enjoyed widerspread popularity, mostly because of difficulties in describing the full syntax of the target-language (Feldman & Gries (68)) and because of inefficiency.

Recent advances in formal language theory have permitted efficient parsers, based on deterministic pushdown automata techniques, to be constructed. DeRemer (71) has devised a method to produce LR parsers of practical size. Lalonde, Lee, and Horning (71) have built a parser generator based on these principles. McKeeman, Horning, and Wortman (70) have implemented the XPL parser generator for simple mixed-strategy precedence grammars. Lecarme (72) has described a generator for uniquely invertible weak precedence grammars.

Since different syntax analysers may be required for different user-languages, it appears advantageous to use a TWS to automatically generate parsers for them. This may be possible in some cases, but with severe restrictions, and at the cost of accepting, what McLure (72) calls, a "stiff, prescribed mold".

First, the grammar of the user-language to be parsed must be cast in the form required by the TWS, which may be a difficult task in practice, and requires someone trained in formal language theory to do it. Even so, the resulting grammar may be awkward to use, since this seems to happen even with programming languages (Aho & Ullman (72)).

Next, a single parsing strategy is mandatory for the entire grammar, whereas it may be better to partition it into smaller subgrammars, for which more efficient parsers can be built and then merged (Aho & Ulman (73, p. 632)). McLure (72) reports that grammar splitting for improved efficiency is current practice in a number of actual compilers. The Automatic Translation Group at the University of Grenoble have also found it more practical to decompose natural-language grammars into networks of elementary grammars, each parsable by the parser of the lowest power possible (Chauché (74)).

The single strategy of the TWS may thus be too powerful and

wasteful for cases where finite-automata would suffice, and on the other hand it may be insufficient for parts of the language that are context-sensitive. For ULANG, a mixed parsing strategy is also used, based on finite-state automata for the most part, except for expressions, when operator-precedence is used.

Then, usually the TWS uses a "bottom-up" parsing scheme, which looks for a limited class of primitive tokens, and uses tables to determine whether or not arrangements of tokens are legal. For many applications, the tokens fit together in a limited number of ways and the most efficient parsing scheme is "top-down" - having recognized the command, look for the appropriate parameter values. With top-down parsing, calls to semantic routines can be made much earlier (Aho & Ullman (73)). Another disadvantage with bottom-up parsing for applications is the difficulty to vary the scanning rules as the context varies (Martin & Guertin (73)).

Finally, the newer, more efficient TWS, mentioned above, are still in the experimental stages and are tied to the systems on which they were developed. XPL of McKeeman, Horning, and Wortman is destined for PL/I like languages on the IBM 360. The TWS of Lecarme (72) requires the target-language compiler to be implemented in PASCAL.

The current situation of TWS with respect to user-languages has been aptly summarized by Sammet (72) :

> "In theory, any compiler-compiler, meta-compiler or similarly designated system could be used for this purpose. However, there is a different emphasis in most of those developed to date. They have been designed primarily to provide an easy means of implementing known and widely used languages rather than as a tool for the development of new languages with uncommon requirements, and their processors. Thus the major considerations have pertained to efficiency of the resulting compiler, with an easy way to make minor changes in the syntax".

Semantic preprocessing. There is no direct equivalent in TWS to the next phase of ULANG, the semantic preprocessor, which checks the user-supplied input information for validity in a given context, and supplies default values if the user has not done so. This phase also rearranges the input information into the rigid sequence required by the processing-logic phase PR.

In other syntax-directed symbol processors, such as APAREL, the programmer simply has access to the raw output of the syntax phase in the form of strings and some indicators about the outcome of the parse. In the AED system the parsed output is placed into global "plex" structures, which are data structures whose elements can be links as well as data. The processing and construction of the plexes are accomplished with macro-routines.

2.3.3 Extensible Languages

Extensibility means definition of new language facilities in terms of existing ones. In general, extensions could occur along four directions : syntax, data types, operations, and control, as defined by Wegbreit (71) for the ECL system, now under development. Usually the extension capabilities are limited to one or two of these aspects. Notley (71) also remarks that the "cut" at which the extension is to take place, whether by user or by programmer, is to be noted. For instance in the REL system, Dostert (71) defines extensibility as the ability of the user to define terms which modify relationships in the data. It is too early to say if any of the proposed extensible languages will be implemented and used for applications, but "there is not much evidence of practical systems or significant usage as yet" (Sammet (72)).

In ULANG, extension of vocabulary can take place at the user level. Also existing parameters or previously defined expressions can be combined into new expressions. At the moment, only arithmetic, relational, and logical operators have been implemented, but others can be added. The LINK and CLASS facilities of the setup language are also potentially powerful data type extension facilities for the user.

2.4 Criteria for language design

Criteria for programming language design have been formulated by McKeeman (66). Although McKeeman addresses himself to algorithmic languages, an extrapolation can be made to user-languages.

First, he points out that the major responsibility for computer language design should rest with the language user, i.e. the programmer. This is in agreement with the objectives of this work, namely that the responsibility for an application-language should rest with the end-user, who should define his requirements and his vocabulary jointly with the implementer.

McKeeman aims to provide a tool, which would reduce the compiler writer's task to manageable proportions, in his case an "extendable compiler", which will do text scanning, syntactic analysis, and will allow the programmer to concentrate on translating the syntax to semantics. ULANG in turn aims to provide a tool which will reduce the application writer's task to manageable proportions, by providing him with a tool described herein.

McKeeman postulates a set of basic algorithmic concepts, common to all computing tasks, via a "kernel language", more powerful than EULER, as defined by Wirth and Weber (66). Since the form of a language is deter-

mined by its grammar, Mc Keeman investigates the possibilities of automatically constructing parsers for some classes of deterministic context-free grammars, in particular for the (1, 1) and (2, 1), (1, 2) bounded-context grammars of Floyd (63) and of Wirth & Weber (66). He concludes that, although the grammars are sufficient for describing existing context-free languages, the tables generated are too large for practical purposes (p. 50).

A compiler generator based on Mc Keeman's work on bounded-context grammars has been implemented for the IBM 360 and is known as the XPL compiler writing system. It is documented in the book A Compiler Generator (1970) by McKeeman, Horning, and Wortman. Il should be noted that XPL was not implemented in the kernel language proposed in McKeeman's thesis, but in a dialect of PL/I, and in turn generates PL/I based languages.

In comparison, the purpose of ULANG is not to devise a new algorithmic kernel language, but to provide means of using existing algorithmic languages more effectively. By analogy, a kernel user-language is defined, together with some basic operators and functions for applications.

In conclusion, whereas McKeeman has defined programming language design criteria, in this thesis criteria for user-language design have been postulated.

CHAPTER 3 - A USER-LANGUAGE

3.1 Introduction

In this chapter, the user-language aspects are discussed. General guidelines for the formulation of user-requests are given. In this work, the syntax of a commonly encountered type of language for specialized applications is presented. The user-requests in this language are of the form

COMMAND, $PARAMETER_1$, ... , $PARAMETER_n$

The range of applications for a language having this syntax is outlined in section 1.4.3.

Some examples follow, showing other types of application languages, which have a different syntax :

a) Graphical applications. In the GRAF language, described by Hurwitz and Citron (67), the statement

K72A(2,1,3) = PLACE(0,1) + PRINT 13,(YY(I), I=1,8)

indicates that the value of point K72A is obtained by taking each of the graphic orders PLACE and PRINT in turn. The + sign indicates sequencing.

b) Declarative languages. A typical statement of DATA-TEXT (67) for social science research might be

VAR(1) = SEX OF SUBJECT (MALE/FEMALE)

c) A movie scanner language, described by Knowlton (64), has statements like IF ANY (B,C,10), (B,A,C) ,(A,E,7),...

When stating user-requests, rules of formation of syntactic entities on several levels have to be considered. First there is the

lexical level : the meaning of characters, and rules of formation of words, or tokens. Next, the sentence level : the meaning of words and tokens, and the rules of formation of sentences. Finally there is the request level, or grouping of sentences into paragraphs.. This is analogous to rules of grammar in natural-language.

For illustrative purposes, some examples of sentences have been taken from applications described in chapter 6. Capital letters are used for actual examples and for ULANG keywords. To designate syntactic entities, the usual metalinguistic notation with angular brackets is used, e.g. <integer> designates any integer.

First, the general characteristics of the implemented user-language are given, followed by the rules for formation of words, sentences, and requests. A metalinguistic description of the user-language is presented in Appendix A.

3.2 General specifications for a user-language

In the language implemented, the user converses in an English language subset, limited to the framework of his application. The underlying assumptions about language needs have been stated in section 1.4.1. The user is to take an active role in stating requests. The input word order is not important, except for certain keywords. Extraneous words and prepositions may be used at will to improve the readibility of input. If deemed desirable, input spelling mistakes may be corrected (this aspect is not implemented here).

The user, in collaboration with the implementer, defines his own terminology. Synonyms and abbreviations are allowed. As Jean Sammet points out (69, p. 728), there are almost as many ways to say "ADD A TO B" as there are people capable of saying it. Each user's terminology is kept in the user-dictionary, which can be updated at any time. As a side benefit, this approach permits one to easily adapt the same application to different natural languages, say Franch, or German.

To relieve the user of repetitive and lengthy input specifications, liberal use is made of default options for the unspecified parameters. A command and its associated parameters remain in effect until changed by a subsequent command.

As in an incremental compiler, the user-language analyzer attempts to interpret a string of information from all available context, namely the string itself, default values, and previous commands. If the input is still undecidable or inadequate, additional information may be requested from the user in an interactive system. This is similar to "incremental computation", as defined by Lombardi & Raphael (66).

Although ULANG is not meant to be a general problem-solving

system, certain language processing facilities of general usefulness are provided. These include specifications for conditional execution of commands, restrictions on the range of values of items, use of arithmetic expressions, and grouping of data-names into classes.

3.3 Lexical elements

Characters. The smallest unit of input is the character. For a given application, the set of characters used will depend on the internal organization of the machine (BCD, EBCDIC, ASCII), as well as on the symbols available on the keyboard of the terminal device used for input. Whatever the external form of a character may be, its hardware representation is used as an index in a table to obtain the corresponding internal lexical class. In this way any number of different character representations may be handled with appropriate tables.

Atoms. Groups of characters having some meaning are called tokens, or atoms, similar to the usage established by LISP (62). An atom consisting of letters only is synonymous with "word" in the usual sense, e.g. PRICE, PRINT, are words, or atoms. However "atom" is more general then "word", since it also includes numbers, and special-character combinations, such as NOT= . Atoms may be composed of several characters : names are composed of letters, and numbers are composed of digits; an atom may also consist of a single character, such as an arithmetic operator, + , or *.

Atoms in the input string must be separated from each other by at least one delimiter, such as a space, comma, etc. In general, alphabetic and numeric characters are not to be intermixed in the same atom.

From key-driven terminals, the input scanner reads successive characters of the input buffer and assembles them into atoms, until a

delimiter symbol, such as a semicolon (;), is encountered. Alternatively, from graphical, voice, or function-button driven input devices, the input may be in atomic form already, thus simplifying the job of the scanner.

Lexical classes. In this implementation, the scanner classifies an input character into one of the following lexical classes: letter, token-separator, minus sign, operator, digit, decimal-point, string delimiter(quote),label delimiter(colon),end-of-sentence delimiter (semi-colon).

The assembled atoms are in turn assigned to one of the following parameter categories:

i) Numeric constants; integers, e.g. 37, -150, or decimal numbers, e.g. 123.46, .5, -3.0.

ii) Operators, e.g. +, (, <.

iii) Strings of any available characters, enclosed between quotes; usually used for names, or identifiers, such as 'XEROX'.

iv) Names, consisting of alphabetic characters; these are looked up in the user-dictionary by the post-scan phase.

v) Line numbers, or labels, that is numbers followed by a colon (:).

Atoms which have been tagged as "names" by the scanner are further classified by a post-scanner with the help of the user-dictionary.

The user-dictionary. The Dictionary contains all the appropriate terminology for an application. The basic vocabulary, compatible with the processing logic, is defined at set-up time. Later, the users may add their own terms with the SYNONYM command.

There is no inherent meaning to a given word, its usage is strictly determined by its category in the Dictionary. The word REPORT could be used as a command, as in

REPORT [illegible] ACCOUNT 123;

with the meaning "produce a report". Alternatively REPORT could be defined as an item-name, as in

PRINT TRANSACTION REPORT;

Words such as FROM, TO may be defined as limit-descriptors as in

DISPLAY STOCKS FROM 'A' TO 'GM';

However if they have not been defined in the Dictionary, they will be simply ignored.

A Dictionary entry can be defined as belonging to one of the following basic categories:

1. Item-name, or class-name.
2. Value (descriptive-constant).
3. Command, keyword.
4. Descriptor: of limits, of conditions, marker, etc.
5. Operator: logical, relational, arithmetic.

An input-name is looked up in the Dictionary and is replaced by its category designation and by a standard keyword. If the input word is not found in the Dictionary, it is classified as a "noise-word" and is given the category zero. Noise-words may be used freely to make the input more readable. For instance, if in the sentence

DISPLAY THE LATEST MARKET PRICE FOR STOCK 'ABC' ;

only the underlined words are relevant, then it is equivalent to the more concise form

DISPLAY STOCK 'ABC' PRICE ;

Generally, an inexperienced user tends to be more verbose, whereas the experienced user wants a concise input.

Specific semantic actions may be taken for certain noise-words by the syntax analyzers. In this way, some words may be considered as noise-words in general, except for certain commands, where they have some

specific meaning. Examples of user-dictionary entries are given in section 6.1.

Although not done in the present implementation, for some applications it may be useful to attempt probabilistic matching of input words with the closest Dictionary entries. This would be useful for recognizing misspelled and incompletely formed words. For instance, words such as DIVS, DAVE (misspelled), DVD, could all be recognized to mean DIVIDENDS, and be replaced by the same standard entry DIVI.

Text replacement could also be handled via the Dictionary. For instance, the descriptor ALL might be replaced by the equivalent expanded string FROM FIRST TO LAST BY 1.

3.4 Elements of sentences

The basic atomic components, described above, are combined into syntactic units, or sentences. Sentences are made up of a keyword, usually a command, and of some other parameters necessary for the execution of the task specified by the keyword.

The parameters may be data-names, values, arithmetic or Boolean expressions, limits, or conditions. These elements of sentences will now be described in more detail.

3.4.1 Keywords

A keyword is defined to be an entry of type 3 in the Dictionary. It is usually a command-verb. Its primary function is to activate the appropriate processing module. A keyword remains active until its effects are changed by a subsequent keyword.

A keyword may appear anywhere in the sentence, or not at all, if it has been given in a previous sentence. Usually the keyword is accompanied by some other data within the same sentence.

3.4.2 Data-names

Data-names are names of parameters of different types. Item-names are names which have been entered into category 1 of the Dictionary, defined at set-up time. Values of these items are to be found in data-files, or in the templates, or else they follow the item-name in the input string. Expression-names are names of arithmetic or Boolean expressions, defined by the user at run-time. The value of such expressions is evaluated and transmitted to the processing-logic module. Class-names are names for collections of items. This is a shorthand notation, which enables the user to refer to several quantities with one name. The preprocessor expands a class-name into its individual components.

3.4.3 Values

In general, there are one or more values associated with a data-name. Values can be of three different types: numeric constants, strings, and descriptive-constants.

Item-names in a sentence may be followed by values, e.g.

CHANGE PRICE TO 23.5 ;

which means that the value of the item PRICE is to be set to 23.5. An item may have multiple values, e.g.

DISPLAY NAMES OF STOCKS 'A', 'MN' and 'TX' ;

which refers to three values of the item STOCK, namely to 'A', to 'MN' and to 'TX'.

In some cases, where no ambiguity is likely, an item-name may be implied, as in

PRINT 'GM' PRICE ;

which really means

PRINT STOCK = 'GM' PRICE ;

An item-name may be implied if the item is a required input-parameter and not an optional one. The implied name is established by the semantic routine SEMIMPL by considering the value in context with other atoms in the string, the parameter sequence in the template, and the type of the value (numeric, alphabetic, or descriptive-constant).

Implied item-names allow for a shorter input string, and may make it more readable. For instance, let us consider the sentence

DESIGN CONCRETE BEAMS ;

Here CONCRETE is descriptive-constant for the item-name MATERIAL which is implied. To a user this is more readable and preferable to a more rigid syntactic form, such as

DESIGN BEAMS, MATERIAL CONCRETE ;

or even

DESIGN BEAMS, MATERIAL = 1 ;

where 1 would mean CONCRETE, 2 STEEL, etc..

3.4.4. Qualification of names and subscripting

Multiple values of items may be thought of as entries in a table of one or more dimensions. Depending on the size of the table, it may be physically an array in core or a data file. A specific value can then be selected by specifying all the appropriate coordinates, or subscripts. Such a data-name has to be qualified by its subscripts.

As an example, the item PRICE may be considered as a two-dimensional table, with stock symbols along one dimension, and the time periods (weeks) along the other, as shown in Fig. 3-1. A unique value of PRICE is specified as

PRICE OF STOCK = i, OF WEEK = j ,

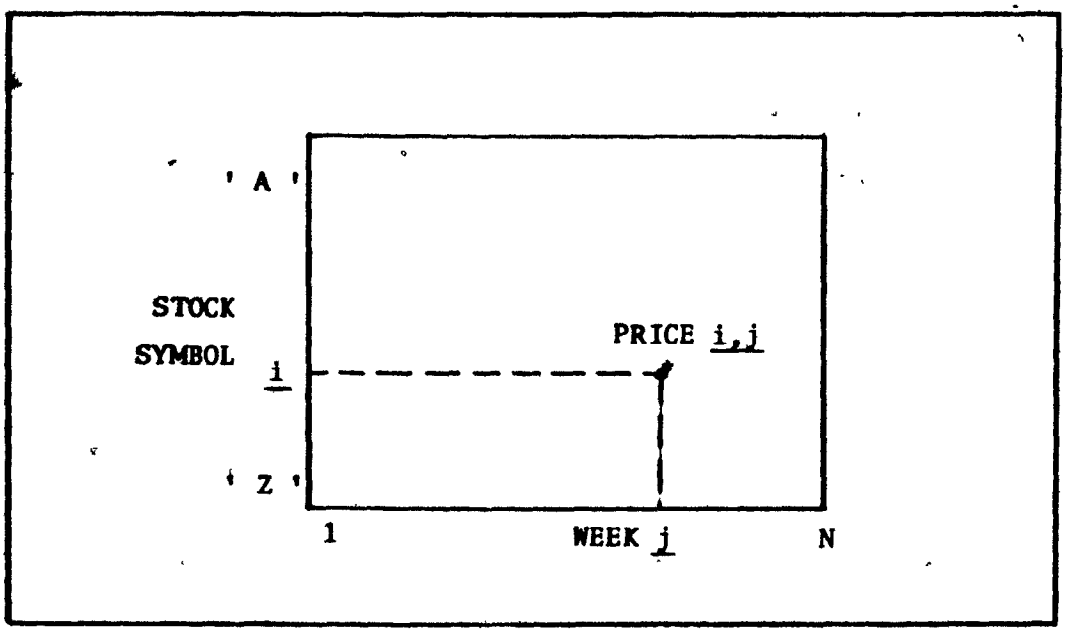


Figure 3-1. Table of stock prices for N weeks.

In order to access a particular PRICE, the values i and j have to be known. To insure this, when the application is set up, the implementer must specify that STOCK and WEEK are required parameters, whenever the item PRICE is mentioned in a request.

The user does not have to follow any specific syntax in his request to indicate that PRICE is a subscripted variable when he wants to access a particular value of PRICE. The subscripts STOCK and WEEK may appear anywhwere in the sentence, and in any order. They are recognized by name, and the usual rules for names and values apply. In this example, the item-name STOCK and the item-name and value of WEEK may be implied, in which case the default value of the current week would be used by USERSEM. Valid requests might be:

```
        DISPLAY PRICE OF 'CH' ;
or      FOR STOCK 'CH', TYPE THE PRICE OF WEEK 2;.
```

If the command

```
        DISPLAY PRICE;
```

was given, the user might be asked to supply the stock-symbol, unless it can be otherwise determined.

Instead of referring to a specific subscript, such as WEEK = 2, the subscript may be variable, in which case certain rules of syntax must be followed.

The name of the subscript may be followed by a position-marker, such as THIS, NEXT, PREVIOUS, optionally modified by a signed integer, e.g. THIS-2, NEXT + 1, or the name of the subscript may be preceded by the markers THIS, NEXT, PREVIOUS only. Alternatively, the name of the subscript may be preceded by an unsigned integer and followed by a direction-marker, such as AGO, HENCE. Some examples of variable subcripting are shown in Table 3-1.

TABLE 3-1 EXAMPLES OF VARIABLE SUBSCRIPTING

	Effective value
WEEK NEXT + 1	T + 2
WEEK THIS - 5	T - 5
THIS WEEK	T
PREVIOUS WEEK	T - 1
3 WEEKS AGO	T - 3
2 WEEKS HENCE	T + 2

Instead of the suggested position-marker THIS to stand for the current value, the user may of course use other synonyms, say T, or I. Before the subscript can be evaluated, the position-marker must either be given a value, or else a default value is used.

When using a qualified item-name in arithmetic expressions, the item-name and the subscript must be adjacent and connected with

the keyword OF, as in

DISPLAY = PRICE OF THIS WEEK / PRICE OF PREVIOUS WEEK * 100;

(not implemented at present).

If name qualification is to be used, then the position-markers and direction-markers have to be entered in the user-dictionary with a UDICT command.

3.4.5 Limit-loops

Limit-loops are used to indicate the range of validity of the command for the items specified, e.g.

DISPLAY STOCK PRICES FROM 'CCM' TO 'GM', 'MB', FROM 'X' TO 'Z';

which would limit the range of stocks for which prices are to be displayed for those with stock symbols between 'CCM' and 'GM', then 'MB' by itself, and finally those between 'X' and 'Z'.

Another example :

RESTRICT DIAMETER TO MIN 5 MAX 10;

which places minimum and maximum limits for the item called DIAMETER.

For numeric values, an increment can also be specified, e.g. FROM -2 TO 75.3 BY 2.58.

3.4.6 Expressions

Arithmetic of Boolean expressions may be used in a sentence, in the general format

<expr-name> = <expression>

Example :

DISPLAY RATIO = (PRICE + DIVIDENDS) / COST * 100 FOR 'MB';

arithmetic expression

The expression-name is entered into a special-name section.

Expressions are composed of item-names and of constants, separated by operators and by parentheses. The operators implemented are given by entries of category 5 in Table 4-1. Boolean expressions are composed of predicates connected by logical operators OR, AND, NOT. A predicate is a relation consisting of two arithmetic expressions connected by a relational operator, such as GREATER, =, LESS. The complete syntax for arithmetic and Boolean expressions is given in Appendix A.3.

Expressions are evaluated from left to right in the sequence implied by the priorities of operators. In decreasing order, the operator priorities are : exponentiation, multiplication/division, addition/subtraction, relational operations (such as $=, <, >$), logical NOT, logical AND, logical OR. The implied priorities of operations can be explicitly modified by the use of parentheses.

3.4.7 Conditions

Conditional sentences are used to make the execution of a command dependent upon certain relations that must hold, otherwise that command is skipped, e.g.

PRINT NAME FOR STOCK 'AB' IF PRICE / COST $>$ 1 $\wedge$ DIVS $>$ 0;

A condition is composed of the keywords IF or WHILE, followed by a Boolean expression. The Boolean expression is evaluated at run - time, and if it is true then the command is executed, otherwise it is skipped.

The WHILE condition is similar, but it also acts as a limit based on some condition being true, e.g.

PRINT ALL STOCK PRICES WHILE STOCK IS NOT EQUAL TO 'M';

which would print all the stock prices until the stock 'M' was encountered, at which time the command would become inactive.

The end of a condition is recognized in a sentence either by the final semicolon, by the keywords THEN or DO, or else by the command-verb, as in

FOR ALL EMPLOYEES; IF SALARY IS LESS THAN $ 1500, WRITE REPORT;

If a condition is part of a sentence, then the special name ULCOND is entered into the special-name section for that sentence.

3.5 Requests

For a given application, the complete specification of a prob-.lem to be solved is a request. For example, in a structural engineering application, a request may be as simple as to display some information, or to do some simple calculations, or as complex as to design a complete building.

If the request is a simple one, it may consist of a single sentence, e.g.

TYPE X = PRICE / EARNINGS FOR 'ABX';

The command is executed immediately after the delimiter (;) is encountered. The control after each command reverts to the user. This is called the immediate mode, and is most appropriate for interactive use. Unless otherwise indicated, this is the normal mode of operation.

If the user wishes to formulate more complex requests, he may enter a delayed request mode. It is signaled by the keyword REQUEST ; followed by one or more sentences, as needed, to formulate the request. It is terminated by the keyword EXECUTE;. In this mode the commands are scanned and analyzed but they are not immediately executed. They are simply stacked for execution subsequently. The delayed request mode has not been implemented at the time of writing.

CHAPTER 4 - IMPLEMENTATION OF APPLICATIONS

4.1 Introduction

In this chapter, the ways of using ULANG for the implementation of application-systems are presented. First, the steps of the implementation procedure are examined, in the context of the ULANG system. An example is used to illustrate the implementation procedure and the resulting templates and user-requests. Additional examples can be found in chapter 6.

The setup-language is then presented, together with the reasons for the choice of the setup commands. The representation of the data-structures of the templates, Command Table steps, and user-dictionary is explained. Finally, the methods for accessing values and other attributes of parameters from the problem-solving procedures are given.

The use of ULANG results in better-defined applications and avoids misunderstandings between users and implementers. The setup facilities require them to work together right from the early stages to define the area of discourse of the application and its problem-solving capabilities. The implementer can immediately concentrate on the problem-solving aspects, without being sidetracked by user-language specifications, since a language is already available as part of the ULANG system.

4.2 Implementation steps for an application

The implementation procedure of an applications package with the assistance of ULANG can be analysed into several distinct steps, shown in Fig. 4-1:

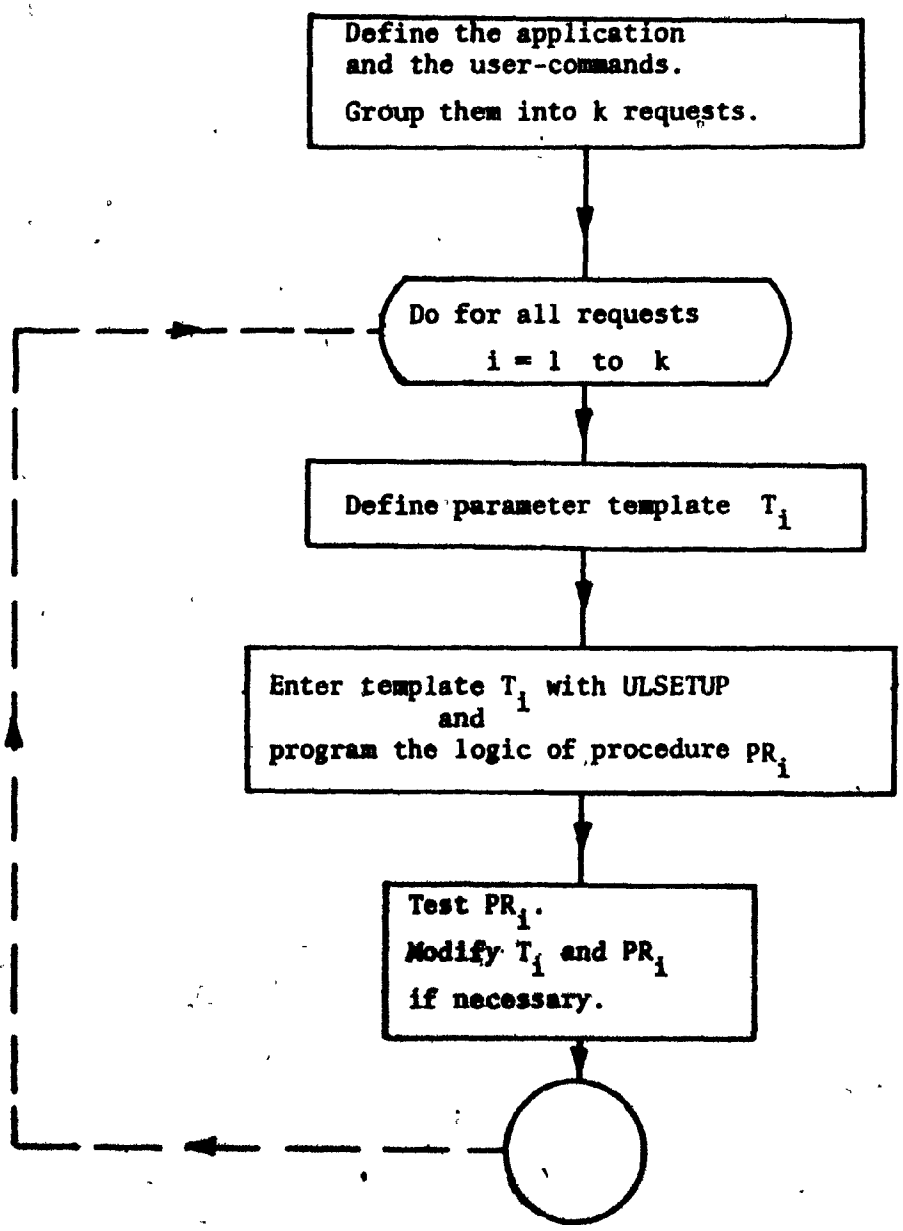


Figure 4-1. The implementation procedure for an application.

i) First there is a planning phase, where the scope of the application and the needs of the users are examined. From this, the type and number of user-requests is determined. A user-request is a certain problem solving activity, that a user may ask the system to carry out. It is initiated by a command. Each user-request i has to be supported by a problem-solving procedure PR_i.

ii) The next step is to define in more detail the processing parameter requirements for each PR_i and to define a parameter template T_i correspondingly.

iii) The parameter template T_i is entered into the system by the setup-language. At the same time the user-dictionary is built-up with the terminology of the application.

iv) The problem-solving procedure PR_i is programmed or is obtained from a subroutine library. The processing parameters are used directly through special ULANG system functions.

Steps ii) to iv) are repeated for all user-requests.

Defining the parameter templates. Each template and its corresponding problem-solving procedure PR_i is identified by a keyword. In addition, each PR_i requires parameters, some mandatory, some optional. The implementer has to decide which parameters are required in order to properly interpret all user-requests.

How this is done, is best illustrated by an example. Additional examples can be found in chapter 6. Let us consider a financial application, based on some data about securities. Let us assume that a user-request consists of predicting the price, earnings, or dividends for a stock or for an industry, by an extrapolation technique, such as regression or moving-averages, over a number of time-periods.

The required parameters might be:

i) the entity for which the extrapolation is to be done, either a stock, or an industry,

ii) the name of the entity, either the stock symbol, or the name of the industry,

iii) the item on which the extrapolation is to be done, either price, earnings, or dividends,

iv) the method of extrapolation to be used, either regression, moving-averages, or exponential-smoothing. It is decided to restrict exponential-smoothing to dividends only,

v) the limits on the time-periods to be encompassed in the extrapolation, namely the first point and the last point.

In this case, six parameter classes are required to define the parameters needed for a PREDICT request.

Having thus decided on all possible input parameters, the implementer enters their description into the template PRED by the setup language statements shown in Fig. 4-2. The resulting contents of the PRED template are displayed in Fig. 4-3. The format of the display is explained in more detail in chapter 6.

The implementer also proceeds to write a procedure which will carry out the extrapolation, using the active parameters provided by the ULANG interface. To obtain the values of these parameters, he makes use of certain run-time functions, described in section 4.5. Typical user-requests are shown (underlined) in Fig. 4-4, with the corresponding active parameters and informative warning messages.

```
LINK KWD TO ENTITY, METHOD TO LIMITS*2
LINK ENTITY TO C-NAME, ITEM
DCLASS KWD IS PREDICT;
 ENTITY= STOCK,INDUSTRY
 ITEM= PRICE FOR STOCK,EARNINGS, DIVIDENDS
 METHOD=(REGRESSION,MOVING-AVG) FOR (PRICE,EARNINGS)
 METHOD IS EXP-SMOOTHING FOR DIVIDENDS
CLASS C-NAME= SYMBOL*A FOR STOCK,NAME*A FOR INDUSTRY
CLASS LIMITS FP,LP
CATEGORY I=FP,LP;
VALUE FP 1 LL 1 UL 15
VALUE LP 15 LL 1 UL 20
UDICT BEGIN 431 END 432 OPTION 373;
DISPLAY
```

Figure 4-2. Setup statements for PREDICT command.

```
TEMPLATE PRED

C =   9 KWD          NPR =    1 S = ENTI        B =
    M =  15 K PRED             IV =  1

C =   4 ENTI         NPR =    1 S = CNAME       B = METH
    M =  18 D STOC             IV =  1
    M =   7 D INDU             IV =  2

C =   1 CNAME        NPR =    1 S =             B = ITEM
    M =  19 A SYMB             IV =  0 FOR STOC
    M =  14 A NAME             IV =  0 FOR INDU

C =   8 ITEM         NPR =    1 S =             B =
    M =  16 D PRIC             IV =  1 FOR STOC
    M =   3 D EARN             IV =  2
    M =   2 D DIVI             IV =  3

C =  12 METH         NPR =    1 S = LIMI        B =
    M =  17 D REGR             IV =  1 FOR PRIC      EARN
    M =  13 D MOVIAV           IV =  2 FOR PRIC      EARN
    M =   5 D EXPSMO           IV =  3 FOR DIVI

C =  10 LIMI         NPR =    2 S =             B =
    M =   6 I FP               IV =  0
      V =          1                    LL =          1 UL =          15

    M =  11 I LP               IV =  0
      V =         15                    LL =          1 UL =          20

                                                .299 SEC.
```

Figure 4-3. Template for PREDICT command.

```
OPTIONS ECHO-OUT;

PREDICT ≠ABC≠;

        ACTIVE PARAMETERS
NAME     TYPE  VALUE     MEANING

KWD        K             1 PRED
ENTI       C             1 STOC
SYMB       A ABC
ITEM       C             1 PRIC
METH       C             1 REGR
FP         I             1
LP         I            15

EARNINGS INDUSTRY NAME=≠FOOD-PROCESSING≠ BY EXP-SMOOTH,FROM 3 TO 10
    EXPSMO   USED IN WRONG CONTEXT

        ACTIVE PARAMETERS
NAME     TYPE  VALUE     MEANING

KWD        K             1 PRED
ENTI       C             2 INDU
NAME       A FOOD-PROCE
           A SSING
ITEM       C             2 EARN
METH       C             3 EXPSMO
FP         I             3
LP         I            10

PREDICT PRICES FOR STOCKS BEGIN ≠MAL≠ END ≠QTL≠ WITH MOVING-AVG

        ACTIVE PARAMETERS
NAME     TYPE  VALUE     MEANING

KWD        K             1 PRED
ENTI       C             1 STOC
SYMB       D FROM
           A MAL
           D TO
           A QTL
ITEM       C             1 PRIC
METH       C             2 MOVIAV
FP         I             1
LP         I            15

                                    .121 SEC.
```

Figure 4-4. Typical PREDICT requests and active parameters.

4.3 The setup language

4.3.1 Introduction

In order to make use of the facilities available through ULANG, the implementer must first set up his application using the subsystem ULSETUP.

The setup phase consists in defining the contents of two types of tables :

a) The basic terminology for the application, to which the users may later add their own synonyms. This terminology is used to build up the Dictionary D.

b) A template T_i for each procedure PR_i , i= 1, 2, ..., k. In general there would be a PR_i for each type of user request r_i. Each template T_i has the names, values, and relationships of all the parameters $\{p\}_i$, which are necessary to be able to interpret and to process the request r_i.

The implementer is given some basic ULANG commands, which enable him to do this setup, such as DEFINE, LINK, CLASS, VALUE. These commands are used when first defining the application, or later, when adding capabilities to it, or modifying it. They should not be confused with user-commands, which are commands that allow a user to formulate his requests at run-time.

4.3.2 Relationships between parameters

There are two ways in which relationships between parameters can exist:

- they may posess a common property, in which case they can be grouped in an array, or a table;

- they may posess a hierarchical relationship, which can be expressed by a tree structure.

In general, both types of relationships are present, making up a compound data structure, which has to be built up at set-up time and accessed at run-time. The data structure has to be retrieval oriented, since that is by far the most frequent mode of usage.

An additional requirement is that the parameter structures should be easily modifiable by the implementer, in order to add new commands, or to change or to add capabilities to an existing command. However the efficiency in processing time for this activity is of lesser importance, since it is relatively infrequent as opposed to run-time use of the parameter structure.

If two parameters XA and YA are dependent on each other in some way, in ULANG this dependency is shown by linking together the classes to which they belong, say CA and CB, by LINK CA TO CB. All relations between parameters are expressed by assigning them into classes and then by linking these classes together.

The parameters are referred to as members of the classes, i.e. XA is a member of CA and YA is a member of CB. Parameters posessing some common property, or different alternatives of the same parameter can be grouped as members XA, XB of the same class CA, by the statement CLASS CA = XA, XB;.

Unless otherwise specified, the members of a given class are dependent on all the members of higher classes to which they are linked. If this is not so, the restriction in dependency can be shown by qualifying the member name with FOR<restriction>; e.g.

DCLASS CB = YA FOR XA, (YB, YC) FOR XB;

which means that YA relates to XA only, and YB, YC to XB only.

Each member or parameter posesses a value. For descriptive-constants this value is only a switch-value, 1, 2, 3, ... representing the different alternatives. Descriptive-constant classes are identified by the keyword DCLASS. For items, the values can be numbers or strings. To assign the value 17.5 to XA, with lower limit 1, and upper limit 20, the statement VALUE XA = 17.5, LL = 1, UL = 20; could be used.

Within a template, the classes (also called families), members, restrictions, and values are represented by F-cells, M-cells, R-cells, and V-cells, respectively.

Next, the important setup commands are described in more detail, followed by a description of the template representation in storage. A complete description of the syntax of setup statements in metalinguistic BNF notation is given in Appendix A.

4.3.3. LINK

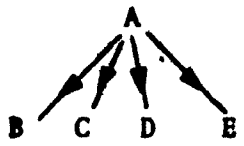
The general purpose of this system command is to define all the names and hierarchical relations between classes (or families) of parameters. All the possible parameters of a command have to be assigned into classes, and the hierarchy of classes must be shown with the LINK commands, of the general form:

LINK<class-name-1> TO <class-name-2> [#< npr >]....

which indicates that parameters of class-2 are dependent on parameters of class-1 ,optionally followed by the number of parameters required.

The highest class of the hierarchy is the keyword (KWD) class. It does not have to be specifically mentioned. Any other class, which is not explicitly linked to a higher class, is assumed to be linked directly to the KWD class.

One LINK command can show either horizontal or vertical relations between parameter classes; for instance the following class structure



can be expressed by the statement

LINK A TO B, C, D, E ;

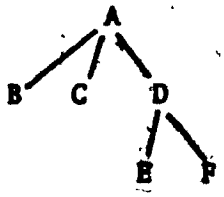
The relation

A ⟶ B ⟶ C ⟶ D

can be expressed by

LINK A TO B TO C TO D ;

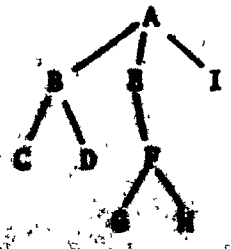
The relation



can be shown by

LINK A TO B, C, D TO E, F ;

Given a more complex relation, it can be expressed by several LINK clauses, e.g.



This can be related with three LINK statements,

LINK A TO B, E, I; LINK B TO C, D; LINK E TO F TO G, H;

For reasons of simplicity of input, it was decided not to allow a general nesting of classes, to any level, with a single LINK clause. This would require another delimiter, say END, in addition to TO, or else a parentheses notation as in LISP, which would make the input hard to read. For example the last structure would become

LINK A TO B TO C, D END, E TO F TO G, H END, END, I ;

or LINK A(B(C,D), E(F(G,H)),I);

Either of these forms is harder to read than the three simple LINK statements above.

Sammet (69, p.406) reports user experience with the LISP notation:

"It is extremely difficult to read and write because of the existence of large number of parentheses and the problem of matching them; as a result it is extremely error prone."

This view is supported by Harrison (70).

To summarize, class relations on the same level (brothers) are shown by listing their names together, following the TO. Relations in depth are shown by nesting the TO's. Only one class-name is allowed between LINK and the first TO.

In addition to the class relations, the LINK statement allows one to specify the number of required parameters of each class, shown as <class-name-2>#<npr>. The indicator <npr> can be an integer, e.g; #1, or some <class-name-3> which has been defined previously. This means that the number of required parameters of class-2 has to be the same as the number of parameters of class-3. If no <npr> is specified, it is assumed that one parameter of that class is required.

4.3.4 CLASS

The parameters making up a given class, which has been previously

defined with a LINK statement, are defined by the CLASS and DCLASS commands, having the following syntax:

CLASS < class-name > = < member-name1 > [#<cat>][FOR < member-name 2 >]...

or DCLASS < class-name > = < member-name1 > [FOR < member-name2 >]...

Unless otherwise specified, the members of a given class relate to all the members of higher classes to which they are linked. If this is not the case, then the restrictions can be shown by qualifying the member-name by the keyword FOR followed by the class or member names to which the restriction applies. A FOR, with a restriction following it, is assumed to apply to the member preceding the FOR. If the qualification applies to more than one preceding member, parentheses may be used to indicate the proper grouping.

For example, assume that in the PREDICT request, regression and moving-averages can only be used for prices and earnings predictions, and exponential-smoothing only with dividends; this is shown as,

```
DCLASS METHOD = (REGRESSION, MOVING_AVG) FOR (PRICE, EARNINGS),
                EXP_SMOOTHING FOR DIVIDENDS;
```

The members of a class can be of three types: commands, descriptive-constants , or item-names. All the members of a given class must be of the same type. DCLASS identifies descriptive-constant and keyword classes. Item-names are defined with the CLASS command. Items can be integers, floating-point numbers, or strings, corresponding respectively to the optional #<cat> specifications I, F, or A. F is the category by default.

4.3.3 VALUE

While commands and descriptive-constants have no inherent

value, except as switches, item-names do have actual values which can be numeric, or alphabetic. These values are either in some data-files, accessible through a file-management system, or they are provided by the user on input, or they can be stored in the V-cells of the template, if the number of values to be saved in the template is not too large.

To enter the values in the template, the implementer uses the VALUE clause, having the following form:

VALUE < item-name > [FOR < restriction >] =<value1> [LL<value 2>] [UL<value 3>]

The item < item-name > must be the name of a previously defined member. The values can be numbers or strings or else they can be procedure names, where the values would be computed, looked-up, or otherwise arrived at. Repetitive specification of the same value can be shown by enclosing the value to be repeated in parentheses and preceding it with a numeric repetition-factor, e.g.

3(150) means 150, 150, 150.

If the user is going to be responsible for providing values of some items on input, the implementer should provide default values for these items whenever possible. At the same time validity range for the input values can be specified by following the value specification with the keywords LL and UL, for lower limit and upper limit, respectively. Example:

VALUE WEEK = 1, LL 1, UL 15;

means that the default value of WEEK is 1, which is also the lower limit; the upper limit is 15. A user-supplied week is checked to see if it is a number between 1 and 15. If not, a warning message is printed and the limiting value is used. If none is supplied by the user, then 1 is used.

Instead of being constants, the limits could also be procedure names, where some more involved lookup or computation could be done to check if a given value is within the acceptable range. This option has not been implemented in the present version. To restrict values to certain items only, the names of the qualifier items (grouped within parentheses if more than one) are given, following the keyword FOR.

4.3.6 Other setup commands

CATEGORY. This command is used to assign or to reassign categories to previously defined items. The format of the command is

< cat-spec > = < item-name1 >,< item-name2 >, ...

where < cat-spec > is I for integer, F for floating-point, and A for alphabetic string. The default category of an item is F. Categories for individual items can also be specified in the CLASS clause.

DEFINE. To open a template the command DEFINE <keyword > is used. If the template < keyword > exists, it is brought in from disk storage and opened. If the template does not yet exist, it is created. If another template is open at the time, it is first closed, before opening the new template.

DISPLAY is used to list all classes of an open template in family-order, with their members and values. The format of the displayed template is described in chapter 6.

DUMP-T is used to dump in numeric form all the subfields of all the cells of an open template. This command is used primarily for debugging.

The next three commands can be useful to the implementer as well as to the users.

OPTIONS. Intermediate printouts can be activated and deactivated with this command. The following keywords can be used in conjuction with

OPTIONS: ECHO-IN, ECHO-OUT and DUMP-P.

ECHO-IN is designed to provide an optional echo-print of an input sentence at the end of the lexical analysis phase. The atom and category strings are displayed.

ECHO-OUT provides a printout of the active parameters in the CT-step after the ULANG syntax and semantic analysis phases. The name, category, and value are displayed for all active parameters.

DUMP-P is used to dump in numeric form all the subfields of the active parameter cells in a CT-step.

By default, all three above options are deactivated. They can also be specifically deactivated by specifying NOECHO-IN, NOECHO-OUT, or NODUMP-P.

SYNONYM has the format:

SYNONYM < name > = < synonym1 >, <synonym2 > ...

It is used to add synonyms to an existing <name> in the user-dictionary.

UDICT has the format:

UDICT < name1 >, < cat1 >,< name2 >,< cat2 > ...

It is used to enter new words in the user-dictionary. These words are not entered in any template. Similarily the DICT command can be used to enter new words in the setup-dictionary.

Although it is an important aspect for an application system, no passwords or other security aspects have been incorporated at this time. This is an intricate problem, which is being studied by others (Conway (72)). Security has to be provided at different levels, from system modification, dictionary and template updating, to the ability of using different user-commands by different user classes, to different users being able to access different types of data if a data-bank is part of the application.

4.4 Representation of parameter structures in storage

4.4.1 Lists and rings

Data structures have been expressed as lists, processable by languages such as LISP (62), if they can be grouped in classes having some common property and if relations between members of different lists can be expressed through their class-header functions. Lists can be traversed in one direction.

This proved to be insufficient for computer-aided design and graphic-display applications, where relations between objects of different classes have to be expressed without passing through the headers. The general data structure in such systems is in the form of rings, where objects having a common property are placed on a ring, which is essentially a two-way list. Many rings may pass through the same element, which implies a variable size cell for each element, and consequently a more complex storage structure. This approach allows a data structure to be entered at any point as well as an unbounded number of relations to be expressed between objects. Examples of ring structured data organization schemes are SKETCHPAD, CORAL, APL at GM Research, and ASP systems. These have been reviewed by Gray (67).

The full capabilities of ring-structure systems are not required in ULANG, because the accessing of parameters is carried out in systematic way. List processing languages could be used for implementing the template structure. However this would remove some of its usefulness and generality from ULANG, since most host-languages, such as Fortran or Cobol, do not support list processing, and even in PL/I its use cannot be generally recommended, because of the additional storage and processing overheads incurred. For instance, pointers in PL/I are one word long, whereas one byte

pointers are sufficient for the templates.

4.4.2 Template organization

One of the most difficult problems of implementation has been the structuring of the templates, to satisfy the following criteria:

- ability of accommodating an unpredictable number of classes, members, restrictions, and values,
- efficient retrieval of information at run-time,
- compact representation in storage,
- simplicity of access mechanisms to insure portability and independence from the peculiarities of any specific system.

The key to accessing a class or a member is its name. The class-names and the member-names are stored together in TNAME. The index of a name in TNAME immediately identifies its corresponding F-cell or M-cell (see Fig. 4-5). The type of cell is indicated by the TYP subfield of a cell.

In this version the names are arranged alphabetically and TNAME is accessed by binary-search, which insures efficient retrieval at run-time. An alternative would be to hash the names. This could be easily done, only the routines FIND and INSERT would have to be modified.

The F-cells form a one-way list through all the parameter families. The M-cells form two-way lists through all the members of a family. Each M-cell points back to its family. There is one list of M-cells for each F-cell. Each M-cell in turn may point to variable length sublists of value-cells (V-cells) and restriction-cells (R-cells).

Corresponding to a name in TNAME there is only one F-cell or M-cell, but there may be none or several R-cells and V-cells. A template consists then of a fixed-size section of length MAXM, containing the F-cells

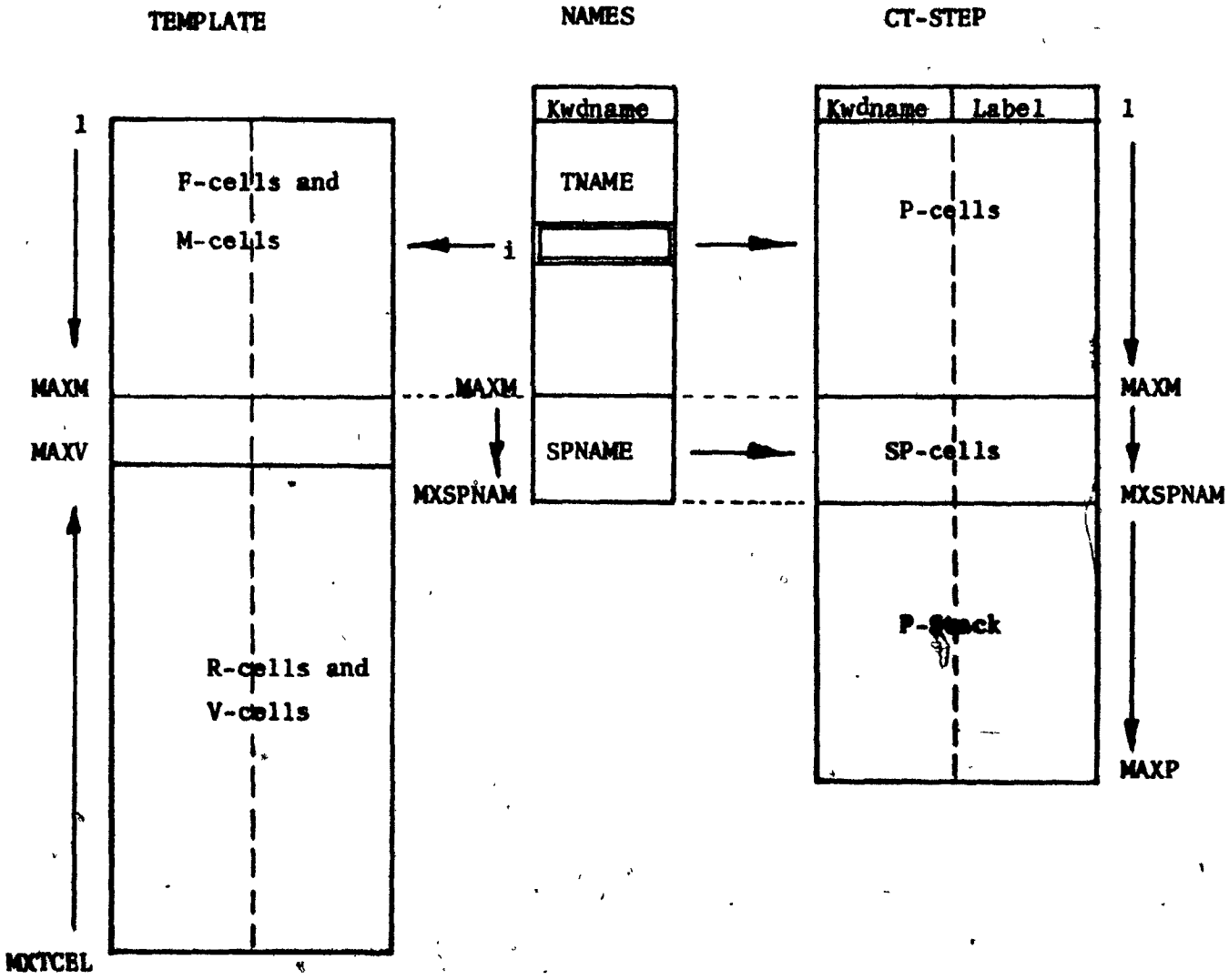


Figure 4-5. Parameter structures.

and M-cells, plus a variable-size section of length MAXV, containing the R-cells and V-cells. All cells are two machine words long and have between 2 and 8 subfields. A subfield is either 8 bits long, or else one machine word long. In this way all cells can be uniformly accessed by the same functions T and TSET.

A template is built up from both ends. In this way only a single area of size MXTCEL has to be reserved to take care of the variable storage needs for classes, members, restrictions, and values.

Each template is saved on disk, together with TNAME and a KWDNAME that identifies it. Subsequently the template can be reopened with a DEFINE command and information can be added to it.

No facilities for modifying or deleting information from a template have been provided at this time. It has been found simpler to modify the original setup-language statements and to re-create the template. The creation of a template and of the user-dictionary, including display of their contents, takes only between 0.2 and 1.0 cpu-seconds on the CDC 6600.

Examples of template contents can be found with the examples in chapter 6.

4.4.3 Command Table organisation

At run-time, a user's input line triggers the consultation of a template and the creation of a step in the Command-Table CT. Each step of a request is identified by a LABEL. Each parameter occupies one or more P-cells within a CT-step.

The P-cells have to satisfy the following criteria:

- rapid creation by merging information from the user's input and from a template,

- efficient retrieval of information by the problem-solving procedure PR, during the execution phase,
- compact representation in storage,
- ability of accommodating multiple-values of parameters of different types,
- ability of accepting previously undefined parameters, such as expression - names,
- simple mechanisms, for machine independence.

The first requirement implies that no time can be wasted on parameter positioning in the CT-step. The index of a parameter-name in TNAME immediately determines its position in the CT-step (see Fig. 4-5). Each P-cell is two machine words long. One word contains the value of the parameter, and the other word has type, status and pointer information. For a single-valued parameter only one P-cell is used. For multiple-valued parameters, or for strings several words long, as many additional cells are allocated in the variable P-stack section as needed.

To allow for user-defined parameters, not in TNAME, a special-name section SPNAME is provided. A maximum of MXSPNAM special-names are allowed, 10 by default. To each SPNAME also corresponds a P-cell.

In REQUEST mode, the successive CT-steps are saved on disk. The information kept for each CT-step consists of its identifying KWDNAME and LABEL, SPNAMES, P-cells, and P-stack, for a maximum of 512 words (in this implementation). The latest CT-step remains in core, ready to be used by the PR procedures.

During the preprocessing phase, one template and one CT-step are in core simultaneously, with TNAME being shared between them (see Fig. 4-5). During the execution phase, the template is no longer used.

Its space may be used by another CT-step, allowing information transfers between two CT-steps.

Examples of CT-step contents, for various user-requests, can be found in chapter 6. The preprocessing and displaying of a user-request varies between 30 and 45 cpu-msec. on the CDC 6600.

4.4.4 Dictionary organisation

The Dictionary contains the vocabulary used by the users when formulating requests. The basic terminology of an application is defined by the implementer, since all class-names, item-names, decriptive-constants and keywords are automatically entered into the user-dictionary at the same time as the template is being built up. In addition, descriptors and operators may be entered separately with the UDICT command. A user may add his own synonyms to the existing terminology with the SYNONYM command (not yet implemented).

In the current implementation, a dictionary entry consists of two machine words. The first word, DNAME, contains the name, reduced to standard-form by SPELL. The second word, DCAT, contains the type of the entry, which is one of the types described in Table 4-1. Additional information could be added to the second word, for instance some information about the item if it is in some file, such as pointer to the file, file-type, position of the item within a record, etc. In that case, a dictionary-cell (D-cell) could be accessed in a way similar to the T-cells and P-cells, with D-functions.

The user-dictionary /UDICT/ is kept on disk. The setup-dictionary is part of BLOCKDATA CONSET. It has 22 entries, which have been defined in the labeled COMMON /ULDAT3/.

4.5 Facilities for accessing parameters

The Command-Table steps contain values and other information of parameters which the application-programmer needs when writing the PR procedures. This information is easily accessible within the framework or the host-language in which the application is coded. In this section methods are outlined for obtaining this information.

4.5.1 Access to values by global variables

If it is known that all parameters are single-valued and of known types, and if no special-names or expressions are permitted, then only MAXM P-cells are used in a CT-step. The implementer could then simply include the global data-area /CTSTEP/ in his procedures. The value of the Ith parameter is at the address PCELL(2*I-1). In this way the values of all parameters can be accessed directly from the PR procedures.

The index I of a parameter in the P-cells may be obtained by displaying the template. It can also be obtained in the PR procedure by calling the integer function FINDP as

I = FINDP(' < param-name > ', STATUS)

STATUS is returned as 1 if the parameter-name was found.

The programmer can EQUIVALENCE the variables used in the PR procedures and the parameters in PCELL, if he wishes.

4.5.2 Access to values by value-functions

Access to values by global variables permits single-valued parameters only, and consequently, a simple user-language. To allow for a richer user-language, the value-functions have been provided. They obtain successive values of a multiple-valued parameter, or successive

words of a string. Ranges of parameters and expressions are also handled.

The value-functions (V-functions) are used in the host language in the same way as any other function. Their usage in FORTRAN is shown here.

The V function. This function returns the current value of a parameter, either a number or one word of a string. V is a one-parameter function, V(< param >), where < param > is either the parameter-name or its index in the template.

To allow for both integer and real values of parameters, an alternate entry point IV is provided. The usage of V and IV functions is illustrated in section 6.1.

If a parameter has multiple values, these are stored in P-stack, and successive references to V obtain them all. Suppose that a user has specified the following input values for some parameter X:

7, FROM 2 TO 15 BY 3, 27

Successive references to V('X') return the values 7, 2, 5, 8, 11, 14, 27, EOV to the calling procedure. EOV is a special end-of-value marker, similar in concept to an end-of-file marker.

To obtain the previous value of a parameter, a special indicator VPREV is used as argument, e.g.

```
IF(V('X').EQ.EOV) X = V(VPREV)
```

keeps returning 27 after all the values are used up. Similarly the special indicator VFIRST always returns the first value, e.g. X = V(VFIRST) returns 7, in the above example. VPREV and VFIRST do not alter the internal current-value pointer CV. They are part of the labeled COMMON /RUNDAT/.

The VM function. To obtain at once all the values of a parameter, or all the words of a string, the VM function can be used. If the calling

procedure has reserved an array of N words, then the call to VM(< param >, < array > ,N) will transfer all the values of < param > to < array > and VM contains the number of words transferred. For instance, the call to VM('X', A1, 10) transfers all the values of argument X to the array A1(10).

The VP function. Upon each reference to the value stack, the next value is given to the calling procedure, until all values become exhausted. Internally a pointer keeps track of the current value, CV, for each parameter.

If it becomes necessary to go through the sequence of values of a parameter more than once, this internal pointer may be reset to a previous value again by invoking the function VP(< param >,< index >). For instance, CALL VP('X', 1) will point again to the first value 7 in the above example, so that the next reference to V('X') returns 7.

The VS function. Sometimes the programmer may want to change the values of a parameter from one of his procedures. This can be done by the function VS(< param >,< value >). For instance, VS('X', 53) would change the current value of X to 53. The programmer should be careful to adjust the current-value pointer with the VP function, prior to affecting the value.

If a parameter is used in a user-defined arithmetic or Boolean expression, the VS function may have to be used to set its current value from the program, prior to the evaluation of the expression.

The VXA and VXB functions. To obtain the values of user-defined arithmetic or Boolean expressions, the functions VXA(< index >) and VXB(< index >) are invoked, respectively. The SPNAME section is consulted, and the first, second, etc., expression or condition is evaluated, and the value is returned to the calling program.

It is the calling program's responsibility to ascertain that any parameters used in such expressions have the appropriate values in them. They may have to be set first by the VS function, described above. The status of a parameter can be tested to see whether it is used in an expression by the user.

Other V-functions. It should be noted that the V-functions described here do by no means exhaust all the possible forms of value-functions which may prove useful for certain applications. The applications-programmer can easily extend himself the existing value-functions provided.

For instance, where the host language permits it, a two-parameter function VL(< param >,< label >) may be implemented, where branch to< label> would be taken if the values of <param > have become exhausted. This would eliminate the need for checking the EOV marker status.

4.5.3 Access to other attributes of parameters

Besides values, the programmer may need some additional attributes of parameters. To obtain additional information about parameters, the index I of the parameter has to be known first. The index can be obtained by calling on the integer function FINP as

I = FINDP('< param-name >', STATUS)

STATUS is returned as 1 if the parameter was found.

Parameter name. TNAME(I) is the name of a parameter that has been defined in a template. SPNAME(I) is the name of a parameter defined by the user at run-time, such as an expression-name. To access these names, the global areas /TEMPL/ and /CTSTEP/ have to be included in the calling procedures.

Parameter status. By invoking P(I, STAT) the status of a parameter I in the current request can be obtained. The meanings of the value of P returned are as follows:

0 = inactive

1 = active

2 = used in arithmetic expression

4 = used in Boolean expression

8 = value of parameter has been set by the VS function from a PR procedure.

The status settings are additive.

Parameter type. By calling on the integer functions P(I,TYP) and P(I, TYP2) the major and minor categories of a parameter I can be obtained. The meaning of the TYP and TYP2 subfields is shown in Table 4-1.

To obtain the proper values for the subfield indicators STAT, TYP, TYP2, the labeled COMMON /RUNDAT/ EOV, TYP, NXT, CV, TYP, TYP2, VALUE, VPREV, VFIRST should be included in the calling procedure.

TABLE 4-1 DESCRIPTION OF PARAMETER TYPES

Major category			Minor category	
TYP	ID	Meaning	TYP2	Meaning
0	*	undefined	00	
1	V	name	00	variable, used in expression
			10	item-name
			20	class-name
			30	arithmetic-expression name
			31	Boolean-expression name
			40	qualifier name
2	C	descriptive-constant	00	
3	K	keyword) command)	00 to 50	user-command or keyword
			71	REQUEST
			72	EXECUTE
			73	OPTIONS
			8y	Dictionary building commands
			9y	Setup commands
4	D	descriptor	01	OF, WITH
			11	THIS
			12	NEXT, THIS + 1
			13	PREVIOUS, THIS - 1
			21	HENCE, THIS +< integer >
			22	AGO, THIS -< integer >
			31	FROM
			32	TO
			33	BY
			41	FIRST
			42	LAST
			43	ALL, from first to last
			50	FOR
			61	IF
			62	WHILE
			63	THEN
			64	DO
			81	LL, lower limit indicator
			82	UL, upper " "
			96	I, integer category indicator
			97	F, real category indicator
			98	A, string category indicator

TABLE 4-1 (continued) DESCRIPTION OF PARAMETER TYPES

Major category			Minor category	
TYP	ID	Meaning	TYP2	Meaning
5	0	operator	10	logical OR
			20	logical AND
			30	logical NOT
			41	less than
			42	less or equal
			43	equal
			44	greater or equal
			45	greater than
			46	not equal
			51	plus
			52	minus
			61	multiply
			62	divide
			63	unary minus
			64	separation operator
			70	exponentiation
			80	), right bracket
			90	(, left bracket
6	I	integer	00	
7	F	floating-point number	00	
8	A	string	xy	x increases by one for each successive word of the string; y is the number of characters in word x
9	L	label	00	

CHAPTER 5 - THE ULANG SYSTEM

5.1 Introduction

This chapter contains a description of the ULANG system in its current implementation, together with a discussion of the problems encountered and the techniques chosen, and of the performance attained.

In the remainder of this section, the general design goals are considered. The next section contains an overview of the components of the system. This is followed by a discussion of the implementation, and by more detailed descriptions of the preprocessor and of the run-time modules. Complete source listings of the system are given in Appendix B.

One of the prime goals of this thesis is to make the ULANG system attractive to the implementers of application systems by providing them with a powerful programming tool, which will make the implementation and the design of applications much faster and easier.

To achieve this, the following subordinate design goals were set:

- perspicuity, since no programmer will trust a tool which he cannot understand,
- portability, to make applications more machine independent,
- efficiency in processing time and storage space,
- extensibility, to enable the implementer to substitute and to add easily his own modules to the system,
- modularity, to clearly separate the different functions involved in user-request processing. This also helps perspicuity, extensibility, and storage efficiency.

5.2 Components of the ULANG system

The ULANG system is made up of three separate modules, only one of which is active at a given time. ULSETUP is used by the implementer to set up an application. The pre-processor ULPREP and the run-time module ULRUN are active in processing user-requests. A brief systems-summary follows.

5.2.1 The setup module ULSETUP

The flow of control and of data at setup-time is shown in Fig. 5-1. The task of this module is to define or to redefine parameter templates and to build up the user-dictionary. The usage of the setup commands and the organisation of the templates and of the user-dictionary have been described in sections 4.3 and 4.4. The complete source program listings are given in Appendix B.

The setup subsystem consists of several parts:

a) A control procedure ULSETUP.

b) The lexical analysis phase ULSCAN which reads the programmer's input string and assembles it into an atom string.

c) A setup-dictionary containing the terminology and the keywords used for building up the templates and the dictionaries.

d) A number of systems procedures, one for each setup command. Each procedure does a syntax analysis of the input and takes the associated semantic table building actions. The setup grammar consists of regular expressions, so that each analyser acts like a finite automaton, driven by the input-string tokens.

The principal setup procedures are the following:

i) DEFINE, STABOP, STABCLO initialise a template and dictionaries, and close them.

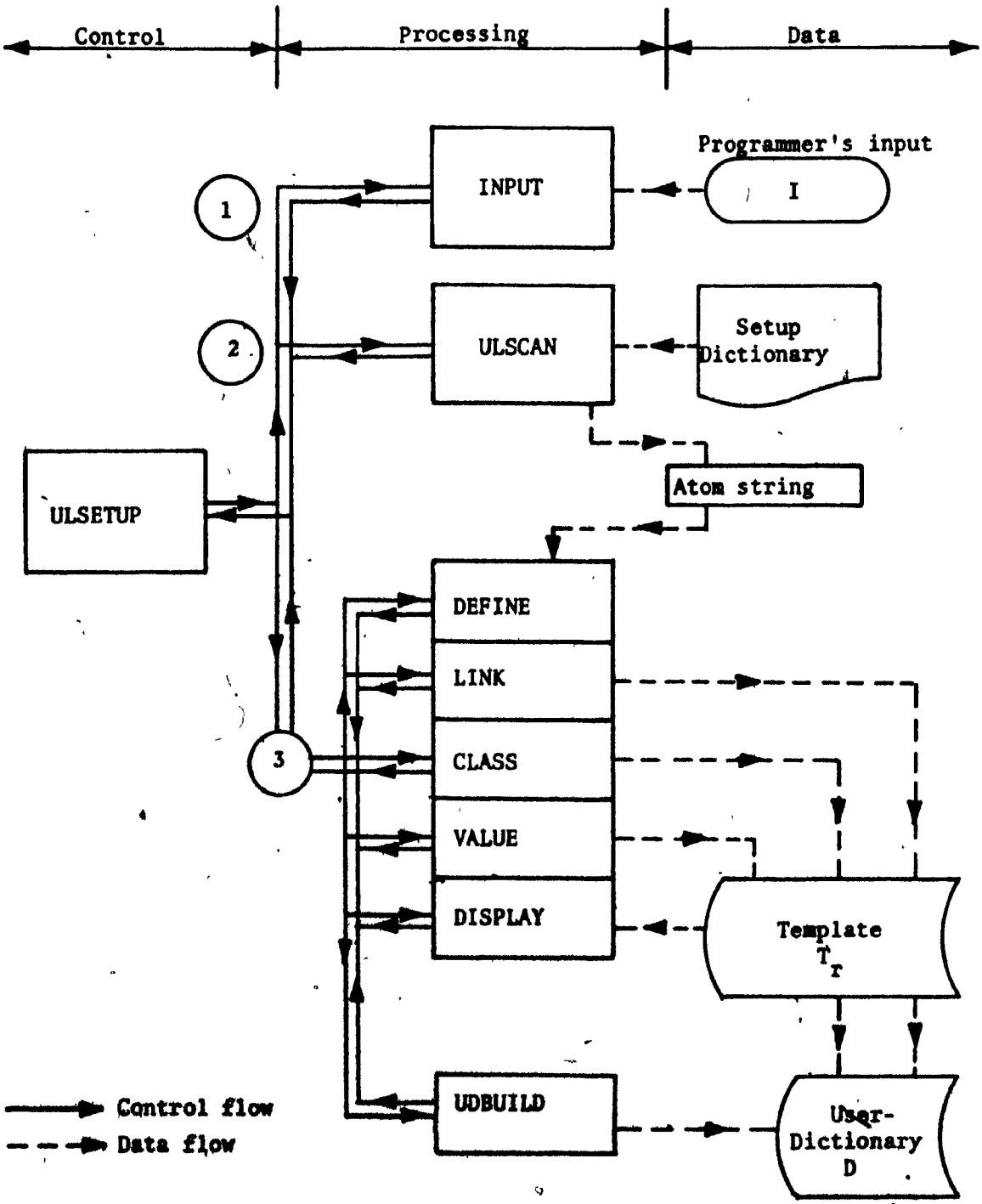


Figure 5-1. Set-up data and control flow.

(ii) CLINK and INSERF form the linkages between classes and build the F-cells of the template.

(iii) CLASS and QUALR define the members of a class with restrictions; they build the M-cells and R-cells.

(iv) DVALUE and QUALV build the V-cells.

(v) DISPLAY and DUMPTC are service procedures to display the contents of a template.

(vi) UDBUILD and INSERX build the user-dictionary.

(vii) INSERT builds TNAME.

5.2.2 Run-time operation

The flow of control and of data at run-time is shown in Fig. 5-2. Two separate phases are involved, the preprocessing phase and the execution phase.

The preprocessor ULPREP. This phase accepts and analyzes successive user-requests r, where r = 1,2, ..., N. It consists of:

- the control procedure ULPREP, which calls in turn,
- a procedure INPUT, to get the user's input string I in a buffer,
- the lexical analyser ULSCAN, which scans the string I, assembles symbols into tokens, outputs the tokens into an atom string, and looks up names in the Dictionary,
- the syntax analyser USERCOM, which parses the atom string, transforms expressions into postfix notation, and builds up the CT-step r,
- the semantic analyser USERSEM, for user-commands, which scans the template T_r, supplies missing parameters and checks the validity of user-supplied parameters.

The execution phase. This phase is active during the execution of the user-requests, which have been stored as steps in the Command Table

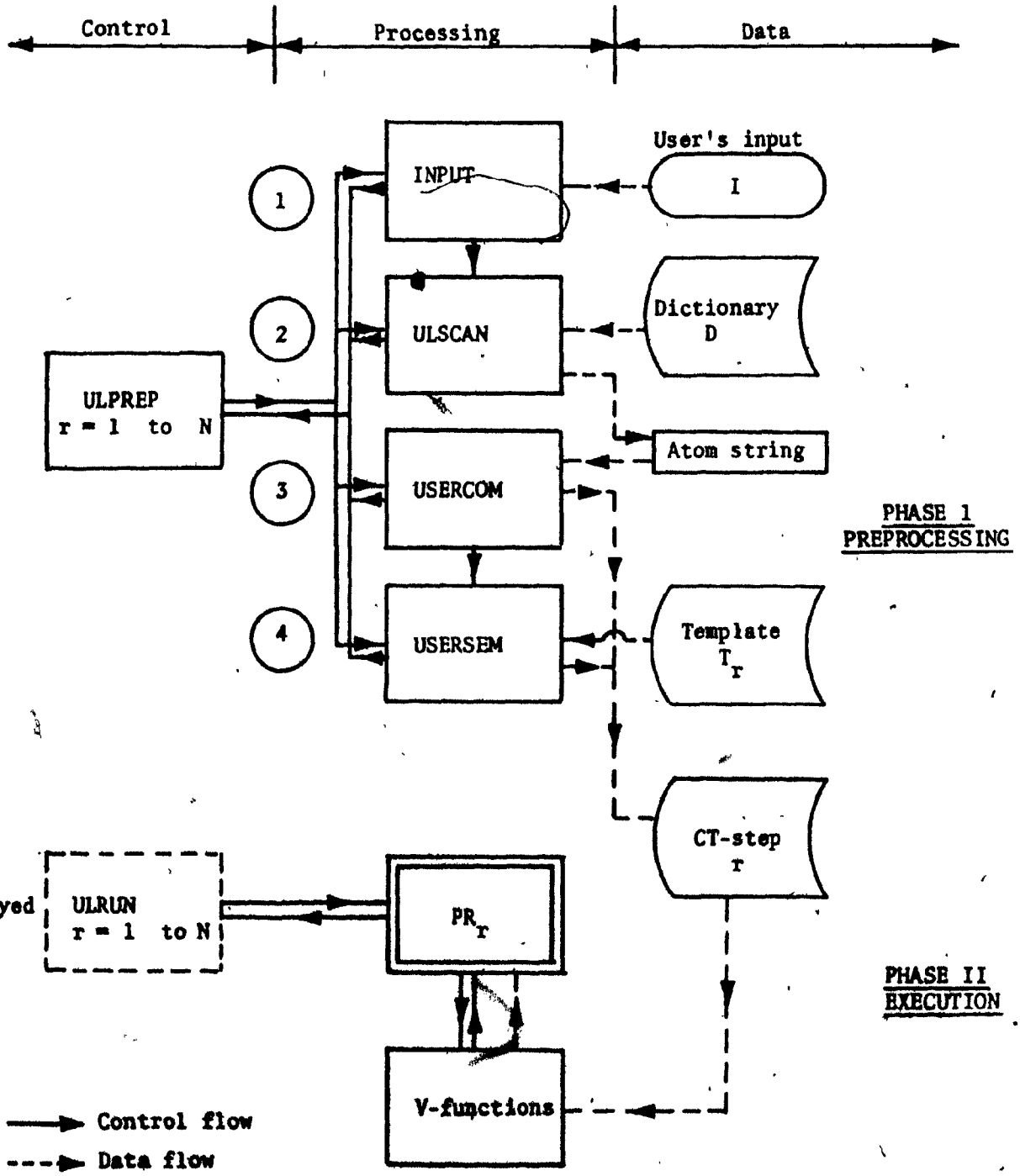


Figure 5-2. Run-time data and control flow.

by the preprocessor phase.

In the immediate mode, control is passed directly to the appropriate PR from ULPREP. In the request mode control passes to a control procedure ULRUN, which has to get the successive steps of CT and to call on the appropriate PRs to process them.

Included in this phase are value-functions, callable from PR to obtain or to set values of parameters in CT, and to evaluate arithmetic and Boolean expressions.

5.2.3 Information flow

The ULANG system is table-driven. A number of data-tables exist in the system. Their representation in storage has been discussed in section 4.4.

The interactions among the data tables are shown in Fig. 5-3. As the implementer is defining the parameters for the templates, the parameter-names used are reduced to a standard form by a procedure SPELL and then entered into the TNAME table and in the Dictionary. Values of the parameters are entered into the template.

When a user is making a request, the preprocessor reduces the names used to the standard form again by SPELL, and then looks them up in the Dictionary. The standard dictionary names are assembled into an intermediate string ATOM, which is matched and combined with the template information to make up a step of the Command Table CT. Values corresponding to the parameter-name are obtained from T or from the ATOM string and are also stored in the CT-step.

At run-time, the PR procedures may access the parameters in a CT-step by using the parameter-names in standard form as arguments of the value-functions, in the manner described in section 4.5.

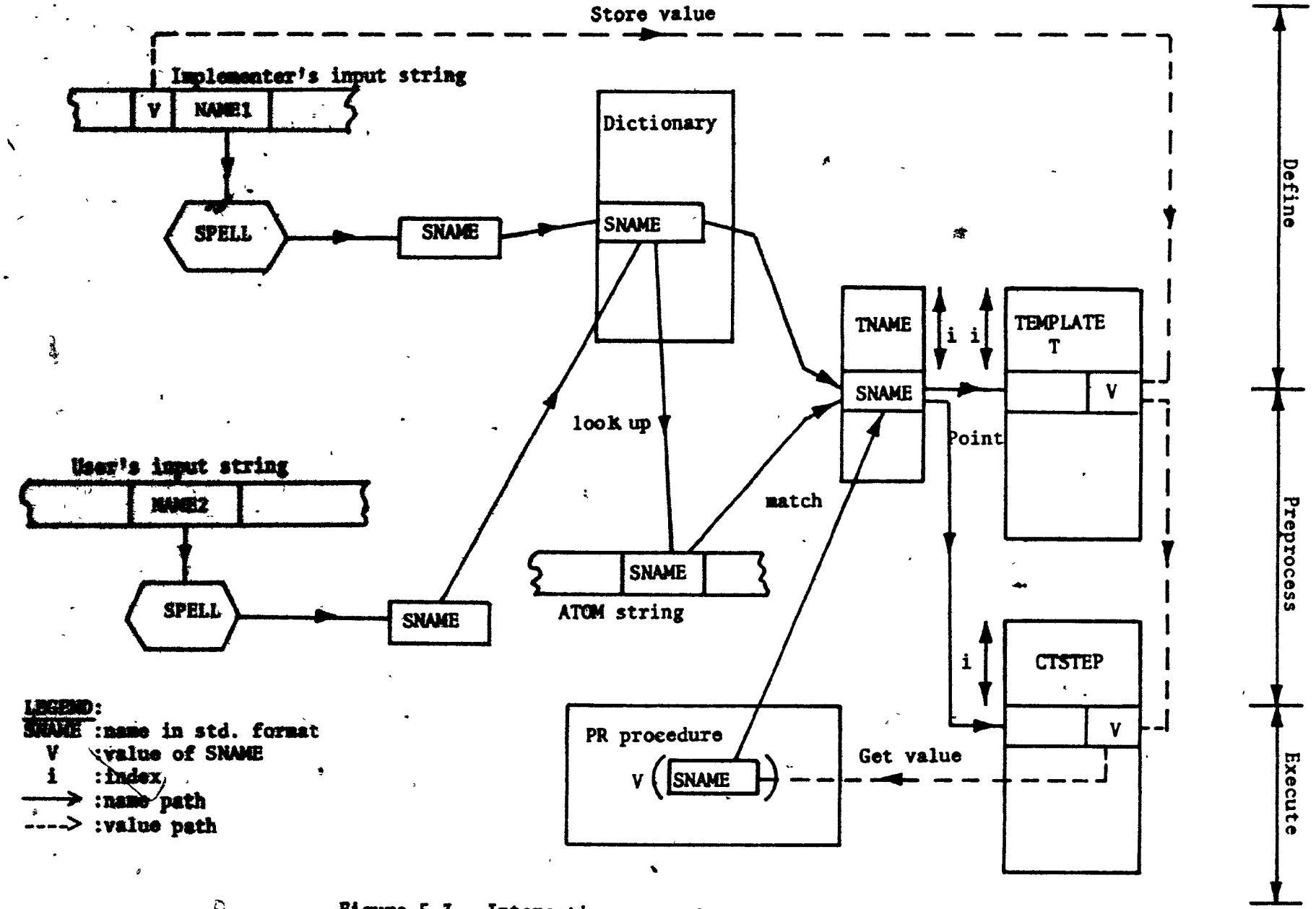


Figure 5-3. Interaction among data tables.

5.3 ULANG implementation

5.3.1 Choice of programming language

The implementation was dictated by the hardware and software available, as well as considerations of machine independence and completion within the shortest possible time. The system available was a CDC 6600 with a batch monitor, assembler, FORTRAN IV and ALGOL compilers. The ALGOL compiler being rather inefficient and unreliable, a choice had to be made between FORTRAN and assembly-language of CDC 6600. For reasons of program readability, portability, and speed of implementation, it was decided to use FORTRAN, which is efficient and widely used on the CDC 6600.

In June 1972 a PASCAL compiler also became available, but by then 50% of the implementation was already completed in FORTRAN. A subsequent version of ULANG could be implemented in PASCAL or in PL / I. Translation from a lower-level language such as FORTRAN, to one with more features such as PASCAL would be easier than translation in the other direction.

Another advantage of FORTRAN is that it still remains the most widely known language, in spite of its drawbacks. Many of the applications which could make use of ULANG are likely to be implemented in FORTRAN for some time to come (Thompson (72)), and programmers are more likely to use a programming tool which they can readily understand.

5.3.2 Program portability and readability considerations

The implementation was done by keeping constantly in mind machine differences and by attempting to localize them as much as possible.

The different modules are self-contained as much as possible. The design decisions and internal data structures of one module are hidden from the other modules. It has been suggested by Parnas (71), that in large systems, connections between modules should contain as little information as possible. In this way changes in one module have the least chance of affecting the others.

For instance, any template information is accessed in a uniform way as T(i,j), where i is the index of the cell and j is the index of the subfield within the call. In this implementation T is a function, but it could also be a two-dimensional array. The size and order of subfields within a cell is hidden and can be easily changed, same as the length of a cell in machine words. The index i is obtained as the position of the parameter name in TNAME by a function FIND. Here again the organisation of TNAME is hidden.

Global variables are used instead of constants as much as possible. All these variables are grouped into three labeled COMMONs and they are defined by BLOCKDATA routines. Blank COMMON has not been used, so as not to interfere with blank COMMON usage by the user programs. The value of a global variable can be easily redefined by simply changing its definition in the BLOCKDATA routine. For instance, the machine word length, LENALF, is set to 10 characters for the CDC 6600; it can be simply redefined as 4 for the IBM 360.

The labeled COMMONs are called ULDAT1, ULDAT2, and ULDAT3. They contain the following constants:

1) ULDAT1 has constants used by the scanner module only:
   - The character set of the machine in terms of lexical classes. An incoming character is used as an index

in this table of lexical classes, CHARCL.

- Word length, LENALF, token length, LENTOK, padding character, EOL.
- Identifiers for lexical classes, e.g. ALPHA is 1, DIGIT is 6.

2) ULDAT2 contains constants used by both the preprocessor and the setup modules:
   - Table dimensions that are used in the program for checking table capacities.
   - Token category names, e.g. CLNAME is 12.
   - Field and type identifiers for templates.
3) ULDAT3 defines constants used by the setup module only, in particular the initial setup dictionary and its categories.

Of course, local variables, used strictly within the context of a single subroutine, are defined locally, e.g. stack depth and precedence function values for the arithmetic-expression analyser AEX.

All inputs and outputs are isolated within specific routines designed for that purpose, for instance display and dump routines to echo print the user's input and computational parameters or to provide dumps of the ATOMs and templates. Errors are processed by specific error routines. At this stage a simple diagnostic only is printed and the processing continues, if possible. In an interactive version the error routine would have to be modified to allow for user interaction to correct the offending item.

Decisions in the program logic flow are taken on the basis of comparisons made between the categories of atoms (integers) rather than the atoms themselves (character strings).

Comparisons depending on the machine collating sequence of characters are localised within one subroutine FIND, which in this version looks up keyword names in the templates and in the Dictionary by a binary-search method. A simple arithmetic comparison works for the CDC 6600, but has to be replaced by a logical-compare routine for the IBM 360.

The programming has been done at a generally available FORTRAN IV level; care was taken not to use features unique to the CDC 6600. For packing and unpacking of words only the logical functions SHIFT, AND, OR are used. These are available on most machines, or they can be easily supplied in assembly language.

Direct access files are used for Dictionary and template storage. The direct access read and write statements of the CDC 6600 may have to be modified for other machines.

It has been pointed out by Dijkstra (68a) that statement numbers and GOTOs render a FORTRAN program difficult to understand, if indiscriminately used. In this work care has been taken to systematise the usage of statement numbers and to structure the programs in such a way as to minimize the use of GOTOs as much as possible.

Statement numbers have been assigned in a systematic way. They have been arranged in an ascending sequence. The high-order digit of a statement number indicates usually a broader program segment. In most cases the numbers also correspond to category definitions, e.g. 600 series statements deal with integers.

The programs are structured in such a way as to make the static structure of the program agree with the dynamic flow of control at run-time. This makes most of the transfers into forward references. Exceptions

to this rule are transfers back to the beginning of a loop, e.g.:

```
      NEXTV=< initial-value >
C          LOOP
20    NEWPTR = P(NEXTV,NXT)
      IF(NEWPTR.EQ.0) GOTO 50
      NEXTV = NEWPTR
      GOTO 20
C          PROGRAM SECTION USING NEXTV
50    ......
```

This loop follows a chain of pointers in NEXTV until it finds one equal to zero, indicating the last one in the chain. In a language having statements of the form

WHILE NEWPTR ≠ 0 DO ...

the statement numbers 20 and 50 of the above example could be eliminated.

An attempt was made to strike a reasonable balance regarding the usage of comment statements to improve program readability. A program becomes unreadable if it is overloaded with comments on every second line, in the sense that this detracts from following the logic of the program statements themselves. In this work, brief but meaningful one-line comments are inserted over small program sections of about five statements.

5.3.3 Program structuring

There is a hierarchical structure between the program modules in the sense illustrated by Dijkstra (68b); a relation of partial ordering exists between them. The levels of the hierarchy are shown in Fig. 5-4.

GETCHAR at level 1 obtains characters from the input string and

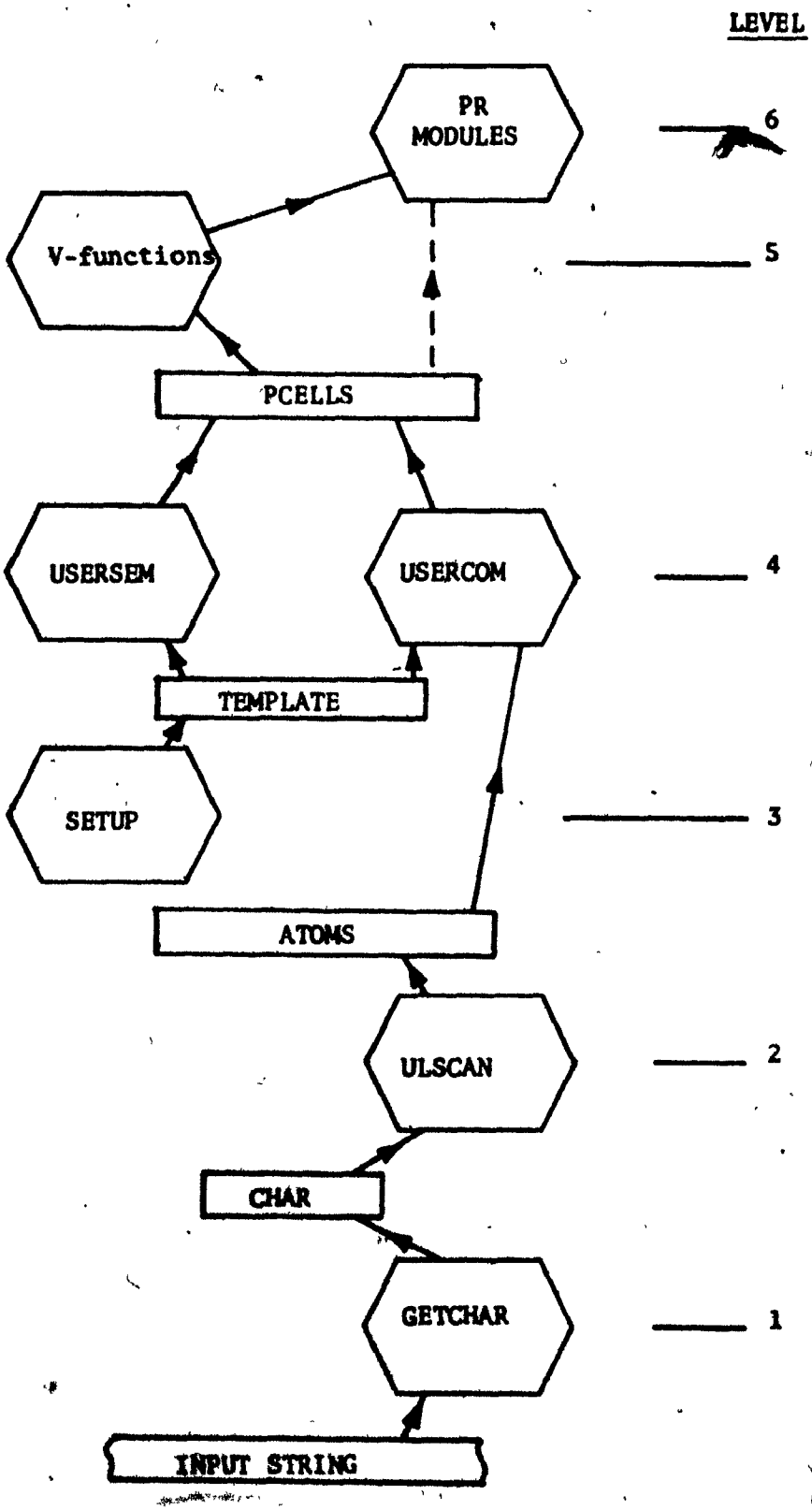


Figure 5-4. Hierarchical program structuring.

categorizes them. The scanner ULSCAN uses characters from level 1 and outputs atoms with their categories. ULSCAN functions on level 2. The user command analyser USERCOM is on level 4 if templates are used, on level 3 otherwise. It uses the categorized atoms from level 2 and the templates to put parameters in PCELLs. Similarly USERSEM is on level 4; it only uses template information. The problem logic modules PR are on level 6 if the V-functions are used, else on level 5.

The hierarchical tree can be cut off at any level and a new tree can be started on the old trunk. For instance, the scanner alone has been sucessfully used in several applications to assemble character strings into categorized atoms. At a higher level, the parameter information in PCELL can be used via the V-functions, or else it can be used directly by PR modules of different kinds.

5.3.4 Information structuring

A great deal of thought was given to the problems of information structuring and machine word length. The CDC 6600 has an unusually long word of 60 bits. Decisions had to be taken regarding the representation of the internal structures of the ATOMs and of the templates.

Atoms. Atoms can contain information of three types: identifiers, numbers, and character strings.

i) Identifiers can be item-names, class-names, descriptive constants, keywords, descriptors, or operators (classes 1 to 5 of the Dictionary). Each identifier occupies one machine word (maximum of LENALF characters). All identifiers are left justified within a word and the unused positions are padded with the EOL padding character, which is zero for the CDC 6600.

All identifiers are reduced to a standard form by a procedure SPELL, which can be easily adapted to different spelling algorithms of varying complexity according to the needs of a particular application. In this implementation, SPELL simply takes the first four characters of an input word unless it is a composite word, in which case the two first letters of the next two input words are also appended, e.g. DIVIDENDS becomes DIVI and DIVIDEND-RATE becomes DIVIRA.

SPELL can be modified without difficulty to build identifiers several machine words long. In that case the procedure NEXTA which obtains the next atom has to be modified accordingly.

ii) Numbers can be integer or real (classes 6 and 7). They are converted directly to their internal representation by the scanner and occupy one machine word each. No double-precision constants have been implemented in this version, but there is no difficulty in allowing a number to extend over several machine words.

iii) Strings (class 8) are stored into as many successive machine words as necessary.

Templates and Command Table steps. The templates and CT-steps consist of cells containing each between 2 and 8 subfields. Except for the subfields containing the actual values of parameters, the information content of the other subfields does not exceed 8 bits (one byte). Thus each cell can be contained within two machine words, for most machines (excepting mini-computers).

However in languages like FORTRAN, no provision is made for addressing subfields within a word, and allocating one word per subfield would be extremely wasteful on storage. In this implementation the subfields are packed within a cell of two words. They are accessed by

a function T(i,j), where i is the index of the cell, and j is the index of a subfield. To set the value of a subfield the routine TSET(i,j,value) is called. Similarly the functions P(i,j) and PSET(i,j,value) are used for accessing the cells of the Command-Table.

5.3.5. Core-storage requirements

The core-storage requirements have met the design goals. The total core storage required on the CDC 6600 by the setup and the preprocessor modules is only 10.5 K words each. This includes space for all tables, buffers, programs, and FORTRAN library routines.

This space requirement could be easily reduced further by program overlays. This can be done because of the hierarchical nature of the programs, as shown in Fig. 5-4. Only one of the program levels is required at a time, together with three data tables, through which the levels communicate. For instance, the syntax analyser phase USERCOM needs only the atom string /TOKEN/ and a template /TEMPL/ for data input and uses /CTSTEP/ for output. The complete data-table requirements for the various program phases are shown in Table 5-1.

This aspect makes the ULANG system particularly interesting to small systems having a limited amount of core storage. Moreover, the table dimensions can be adjusted, if needed,

5.4 The preprocessor module

In this section the components of the preprocessor phase are described in more detail. The general flow of control and of data for the preprocessor is shown in Fig. 5-2.

TABLE 5-1 MAXIMUM STORAGE NEEDS FOR DATA-TABLES

Program phase	Data-table name	Words	Total words
Lexical analyser ULSCAN	INBUF UDICT TOKEN	80 128 128	336
Syntax analyser USERCOM	TOKEN TEMPL CTSTEP	128 512 412	1052
Semantic analyser USERSEM	TEMPL CTSTEP	512 412	924
Run-time phase ULRUN			
Immediate mode	TNAME CTSTEP	111 412	523
Delayed mode	(2 CT-steps)		1026

Legend:

INBUF	input buffer
TOKEN	atom string
UDICT	user-dictionary
TEMPL	template (includes TNAME)
CTSTEP	Command Table step

5.4.1 Preprocessor control

ULPREP is the main-program of a ULANG application. It is a small control procedure, whose purpose is to monitor the flow of control between the various phases of the preprocessor for each user-request.

In the immediate mode, ULPREP also passes the control to the processing-logic, to execute the request in the CT-step. In the request mode, ULPREP simply stores the successive CT-steps on disk, until an EXECUTE command is sensed, at which point control passes to the run-time control routine ULRUN.

ULPREP first calls TABLOP to open the user-dictionary and at the end TABCLO is called to close it.

The remainder of the procedure consists of a loop over all user-requests. Inside of this loop, first INPUT and ULSCAN are called to do the lexical analysis of the input line. Then an auxiliary procedure ULKWD is called, which analyses the keyword, opens a template if necessary, and calls in turn on the appropriate keyword processing routine. For user-commands, USERCOM and USERSEM are called.

5.4.2 Lexical analysis

The lexical analysis phase is separate from the syntax analysis phase, which makes the system more modular. The advantages of separating the two have been pointed out by Johnson et al. (68), Gries (71), and others.

The lexical analysis phase is made up of two parts, INPUT which reads the input string into an input buffer, and ULSCAN which analyses it.

INPUT. This procedure obtains the user's input character string and its length in a buffer INBUF.

INPUT is device-dependent. Its logic is different depending whether the input cames from cards, tape, paper-tape, Teletype, IBM 2741, or other type of terminal. In this version, INPUT obains the input simply by a formated FORTRAN READ statement.

ULSCAN. The task of the scanner is to read successive characters from INBUF, to classify them into lexical categories, and to assemble them into tokens, or atoms.

The different internal representation of characters (BCD, EBCDIC,ASCII) is isolated within the table CHARCL of /ULDAT1/ . When ULSCAN needs the next input character, it calls on a procedure GETCHAR, which uses the hardware representation of the character as an index in CHARCL to obtain its lexical class. In this way it is easy to change the lexical classes of characters for different applications and for different machines, by simply changing the table CHARCL.

The scanner fuctions as a deterministic finite state automaton. Its new state is determined by its current state Q, and the incoming chracter T from the input string. It can be described by a quintuple (K, VT, M, START, Z), where K is the set of possible states of this automaton, VT is the input alphabet, START is the start state, Z is the set of terminal states, and M is a mapping of K x VT into K of the form M(Q, T) → R.

The input alphabet VT can be divided into a number of lexical classes, to be chosen by the implementer. At least one state of the automaton has to correspond to each lexical class. In practice, the number of lexical classes is limited. In this implementation the following lexical classes have been defined:

ALPHA, letters to be used for names

SEPAR, separators, for use with compound names or as special operators

DASH, sign of a negative number, or subtract operator

OPER, an operator

DIGIT, digits to form integer or real numbers

DECIM, decimal point, indicates a real number

QUOTE, delimiter for strings

COLON, delimiter for labels

TERMIN, end of sentence indicator

IGNORE, any other character, not defined elsewhere.

If for some application all of these categories are not needed, then the corresponding states can be removed from the scanner.

The state transition diagram of the scanner is shown in Fig. 5-5. The scanner remains in the START state while IGNORE characters are being encountered. Terminal states are shown by double lines. The number inside the double-lined circle has the same meaning as the TYP subfield for a parameter, shown in Table 4-1.

Semantic actions associated with a state are shown by rectangles with the name of the procedure called to perform the action. In the case of a name, the SPELL procedure is called to reduce the name to some standard form, and the AFTSCAN procedure is called to assign a type to the name.

For every terminal state (except the EOS state) a token is output to the ATOM string, with its type in CAT. The scanner then returns to the START state, ready to build the next atom. State EOS signals the end of the input string, which terminates the lexical analysis.

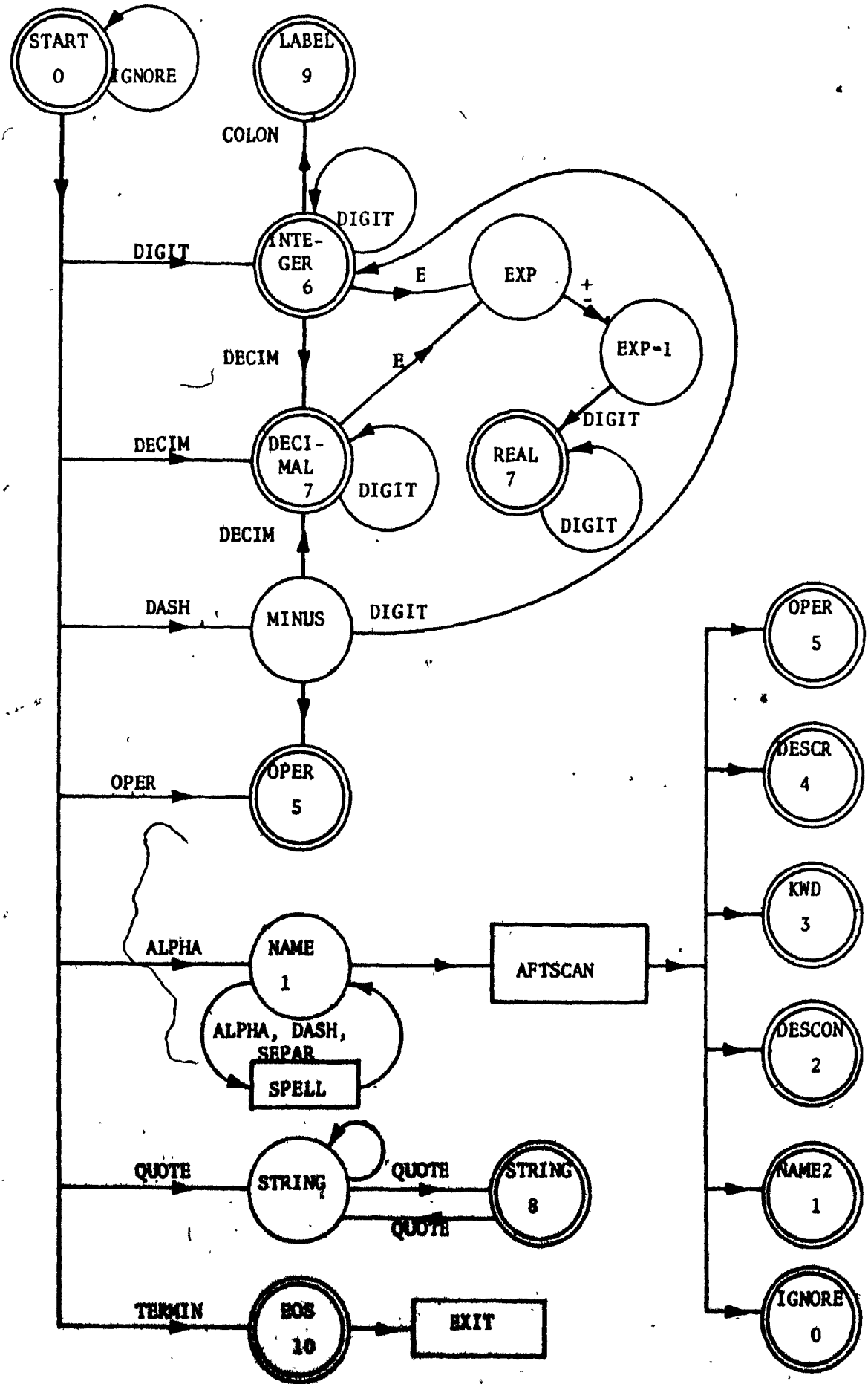


Figure 5-5. State transition diagram for lexical analysis.

The finite automata approach allows one to design the scanner as a set of modular units, where each unit is associated with a node. The logic of the scanner is simplified in this way, since at each node it is only necessary to test for the input characters which are legal at that point.

AFTSCAN. In order to arrive at a final state for the scanner state NAME, a semantic procedure AFTSCAN is called (see Fig. 5-5), which looks up the name in the user-dictionary, UDICT. The final state depends then on the dictionary category DCAT.

A NAME which is not found in the Dictionary is of type 0, and is considered to be a noise-word. A synonym is replaced by the main Dictionary entry. If NAME is a keyword, and no keyword has been encountered yet, then it is placed in position 2 of the ATOM string.

Presently the user-dictionary is organised alphabetically and it is accessed by the procedure FIND, which uses a binary-search method to locate a name.

5.4.3 Syntax analysis

The input to the syntax analysis phase consists of ATOM and CAT strings from the lexical analysis phase. These atoms constitute the terminal alphabet for syntax analysis.

The syntax analysis phase can be made up of a number of independent analyzers, which makes it easy to change them and to adapt them to different grammars. The task of each syntax analyser is to parse the atom string, according to some rules of grammar, and at appropriate points tie in with semantic routines to transfer the information being looked at from the ATOM string to the GT-step.

The auxiliary control-procedure ULKWD determines from the keyword type in CAT(2) which syntax analyser procedure is to be activated. Within the same application there may be different kinds of user-commands, each based on a different grammar, and each requiring a different analyser. Categories 300 to 350 have been reserved for various types of user-commands (see Table 4-1). Besides user-commands, there may also be ULANG system-commands, such as REQUEST, EXECUTE, etc.

At present, one type of user-syntax analyser has been implemented, called USERCOM, based on the rules of grammar described in chapter 3 and in Appendix A. USERCOM and its auxiliary routines are briefly described next. Complete source listings are presented in Appendix B.

USERCOM. The main objective of the parser is to parse the sentence in a single pass from left to right, without more than one symbol lookahead and without backing up. Other objectives are simplicity and perspicuity.

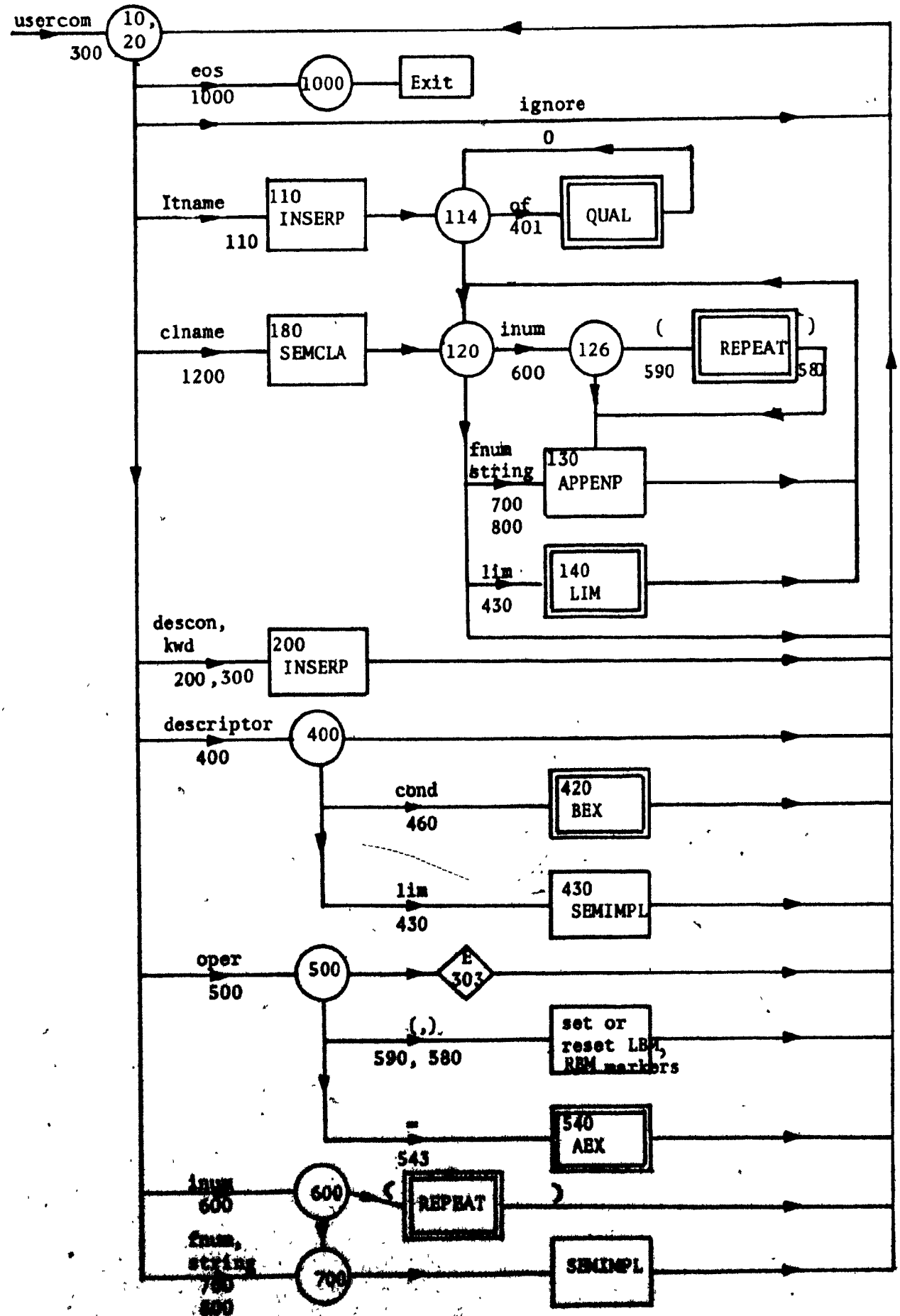
The user-language, as defined in Chapter 3 and in Appendix A, lends itself to a division into two parts, and each part can be analysed by a different method:

1) for arithmetic and Boolean expressions, operator precedence functions and a pushdown stack,

2) for the rest, simple finite-state recognizers, without recursion.

The recognizer is shown in Fig. 5-6, in the form of a transition-diagram, as defined by Conway (63). The only difference is that there is no recursion involved. The syntax of the language can be easily followed from the diagram. Each edge connecting two nodes labelled with a terminal

Figure 5-6. State transition diagram for USERCOM



symbol (ATOM) corresponds to a state-transition and acceptance of that symbol. The circles represent states in the program. The single-framed rectangles are calls to semantic actions. Double-framed rectangles invoke other finite-state recognizers, AEX, BEX, LIM, QUAL, or REPEAT. The numbers in the circles and rectangles correspond to statement numbers in the program.

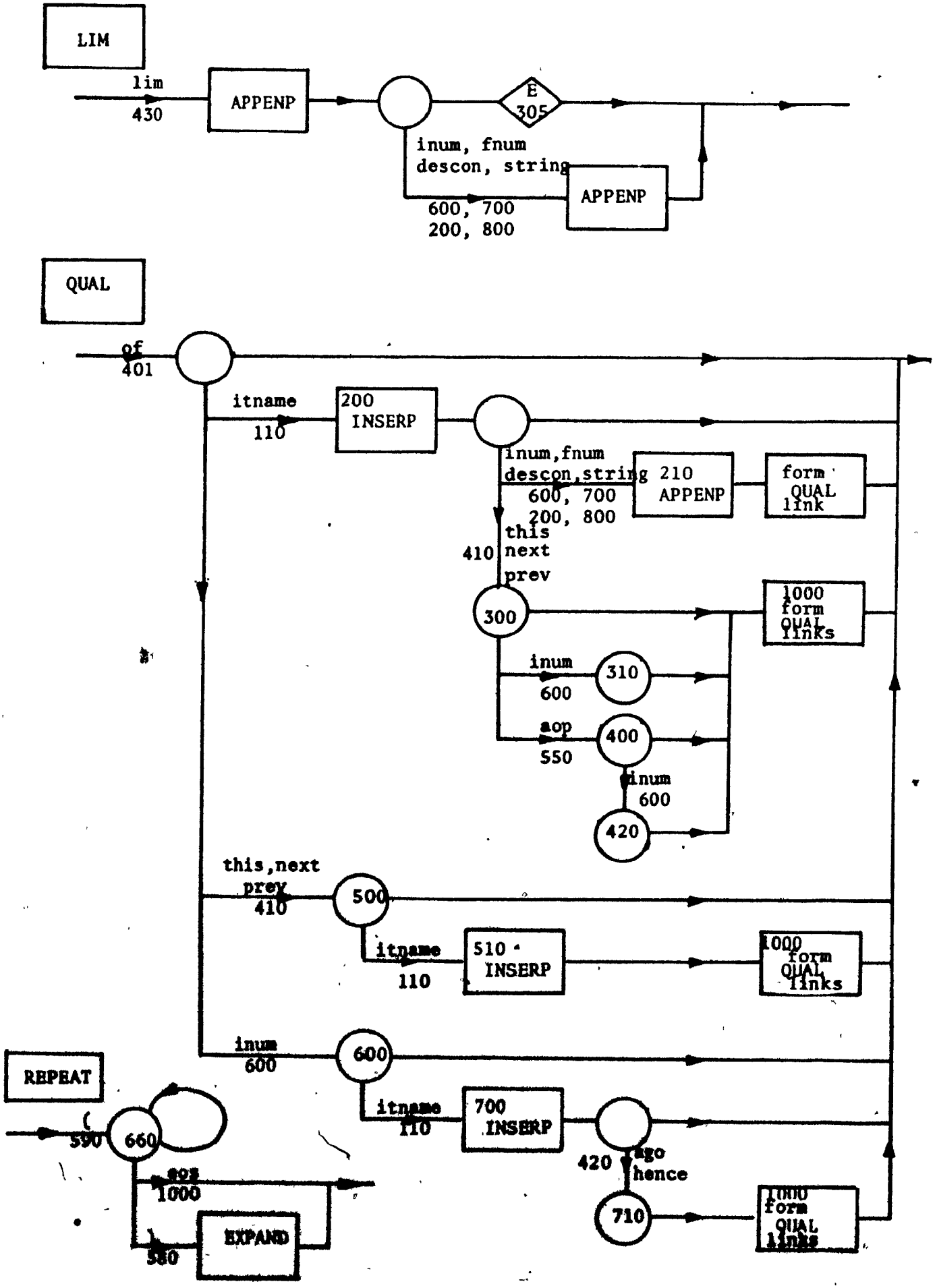
Tie-in with semantics. In order to extract all the information from the atom string in a single pass, tie-in with semantics takes place on each edge of the graph and involves transfer of information to the P-cells.

When it is known or suspected that a parameter P is encountered for the first time, INSERP is called. It activates the status of P in PCELL. To store a value of P, APPENP is invoked. SEMCLA expands a class-name into a string of item-names. The expansion is handled by EXPAND.

SEMIMPL attempts to connect a numeric or string value (types 6, 7, 8) with an item-name which is not present in the input string. This is done by looking at all classes in turn, in family order, starting from the currently active class. If the number of required parameters of a class F has not yet been reached, and the type of a member M of F agrees with the type of the input value, then M is taken as the parameter to which the value belongs.

LIM, QUAL, REPEAT. These are finite-state recognizers, shown by transition-diagrams in Fig. 5-7. Each recognizer analyses a portion of the input string. LIM parses limit ranges of the type FROM a TO b BY c. QUAL analyses name qualification, specified by OF. REPEAT expands a string of values, enclosed in parentheses and preceded by an integer repetition factor. EXPAND is called to do the expansion.

Figure 5-7. State transition diagrams for LIM, QUAL, REPEAT



AEX and BEX. These are entry points to the arithmetic and Boolean expression parser, which uses a different method of syntax analysis from the finite-state recognizers, used elsewhere.

AEX is called when a "noise-word" atom (TYP = 0) followed by an equal sign is encountered. The noise-word is taken as the name of a user-defined expression, and is entered into SPNAME. BEX is the entry point for user-defined conditions, starting with the keywords IF, WHILE.

For expressions, it is sufficient to represent the precedence relations between operators by precedence functions f(X) and g(X), given in Table 5-2. Floyd has shown that if precedence functions f(X) and g(Y) can be found for two operators X and Y, then if $f(X) \leq g(Y)$, this also implies that $X \leq Y$.

For an operator, the second digit of CAT is the index for the precedence function table, so that there is no lookup time involved. The precedence function table only requires 24 entries, instead of 144 which would be required for the full precedence matrix.

The precedence algorithm for parsing expressions has been described and flowcharted by Floyd (63) and by Gries (71, p.130).

An incoming symbol is assigned an index R for g(R). In the case of operators, R is the second digit of CAT. Names and values are terminal symbols VT, (R = 11). The end of the expression is signaled by a special EOS marker (R = 10). If $g(R) < f(S)$ for the top stack-symbol S, then the incoming symbol is stacked, otherwise the head of the prime-phrase is found in the stack, and the prime-phrase is reduced to a non-terminal symbol VN, (R = 12).

Certain semantic actions are performed at the time of this

TABLE 5-2 OPERATOR PRECEDENCE FUNCTIONS

CAT	Meaning	Index X	f(X)	g(X)
510	logical OR	1	5	4
520	logical AND	2	7	6
530	logical NOT	3	7	8
540	relational operator	4	9	8
550	add/subtract operator	5	11	10
560	multiply/divide operator	6	13	12
570	exponentiation	7	13	14
580	right bracket)	8	15	3
590	left bracket (	9	3	14
	EOS marker	10	2	0
110	item-name VT	11	16	16
130	arithmetic expr. name			
131	Boolean expr. name			
600	integer			
700	real			
800	string			
	non-terminal symbol VN	12	1	1

reduction, namely a Polish string is being built-up in P-stack by APPENP. For item-names, the index of the item in TNAME is stored in the value subfield of the P-stack cell. At run-time this index is used to obtain an actual value for the item before evaluating the expression.

5.4.4 Semantic analysis of user-commands

At the end of the syntax analysis phase, all the information from the user's input string has been transferred to the P-cells. The next step is to check if the information supplied is valid and moreover to provide default values from the template for those parameters for which the user has not done so.

This checking is done in a single pass through the template, by visiting all the classes in "family order". First, the values of the active members M of a class F are checked for validity and if more parameters are required, then the inactive members of F are consulted. This work is carried out by a procedure USERSEM, shown in Fig. 5-8.

Inside the loop over the active M-cells (labels 200 to 500), M is checked first to see if it ought to be active in the present context. If M's presence is conditional upon other members of higher classes, this is indicated by the R-cells. If one of the members pointed to by the R-cells is active, then M may be active as well. This is why the "family order" is important.

For items, values may exist in the V-cells. A value of M is acceptable if it has no restrictions, or if a restriction is active. If the user has not supplied an input value for M, then a default value is obtained and inserted into the P-cell by APPENP. If the user has supplied values, they are checked by PBOUNDS, flowcharted in Fig. 5-9.

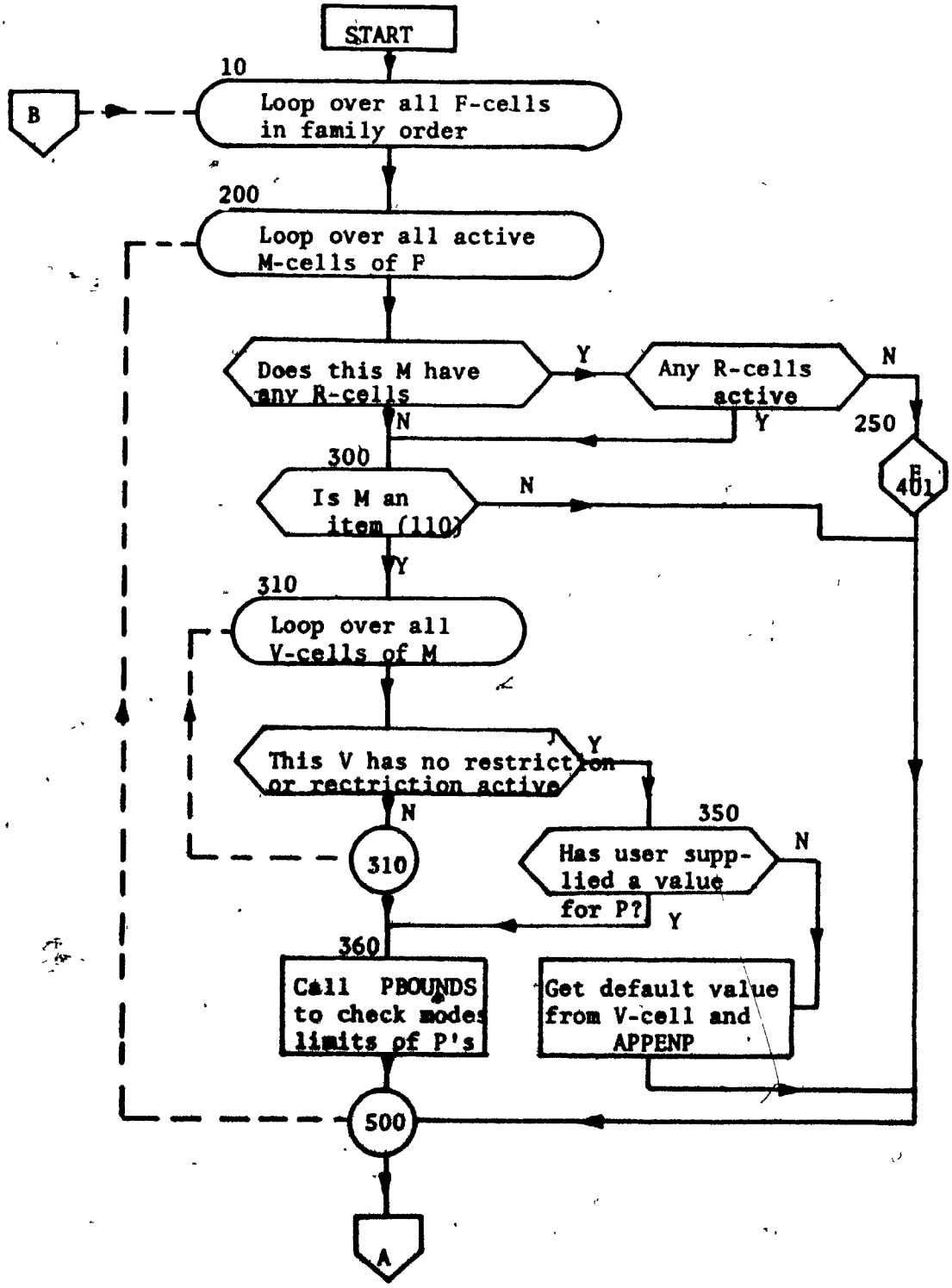


Figure 5-8. Flowchart of USERSEM.

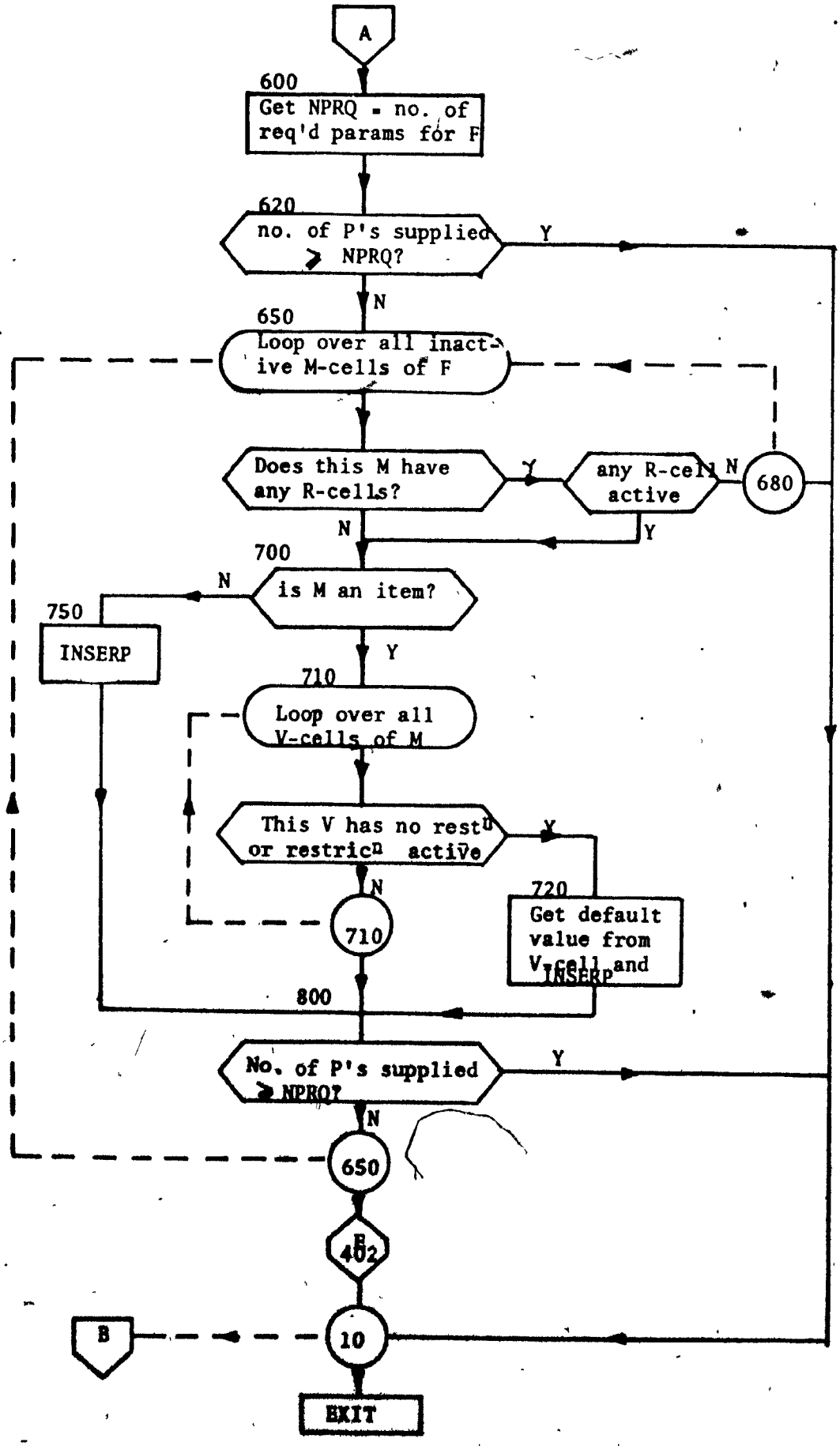


Figure 5-8 (continued). Flow chart of USERSEM

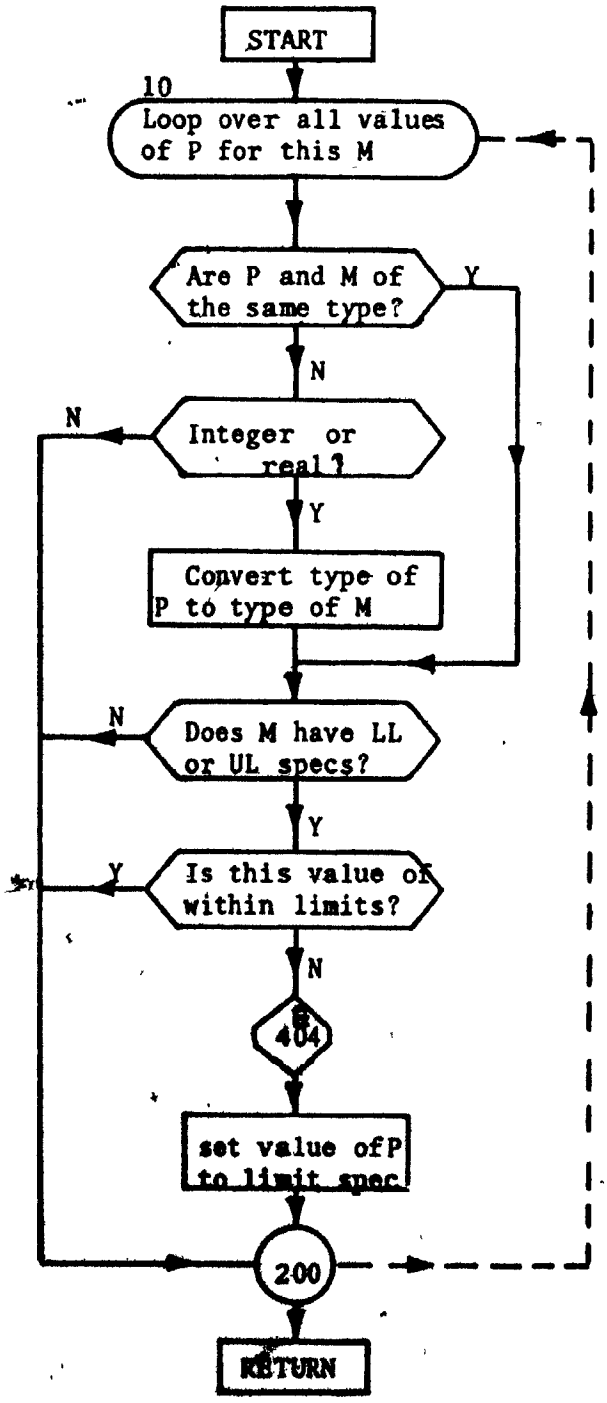


Figure 5-9. Flowchart of PBOUNDS

PBOUNDS checks and converts, if necessary, the modes of all input values to that of the M-cell in the template. The input values are also checked to see if they fall within the lower an upper limits, if specified.

At this point, further processing extensions may be incorporated. Specific procedures might be called to perform some more complex checking, or to obtain values form a data-bank, for instance.

At the end of the active M-cell loop, a check is made to see if the number of user-supplied parameters satisfies the parameter requirements for execution (label 620). If such is not the case, then a loop is entered over all the inactive M-cells of F (label 650). Again, dependencies on higher classes are checked before obtaining a default value for an M, which is inserted into the P-cells by INSERP.

At the end of USERSEM, all the processing parameters have been obtained and checked, and the current CT-step is ready for execution.

5.5 The run-time module

Run-time control. The flow of control and of data at execution time is shown in Fig. 5-1. In the immediate mode, each CT-step is immediately executed directly under the control of ULPREP. In the request mode, all CT-steps are stored on disk, until the EXECUTE command is sensed. Control then switches to a control procedure ULRUN, responsible for bringing in the successive CT-steps and passing the control on to the procedures specified by the KWDNAME of each step.

Value-functions. The purpose of the value-functions is to set or to return successive values of parameters to a calling procedure PR. Usage of the value-functions has been discussed in section 4.5.2 and is illustrated in Chapter 6.

The logic of the value-functions is straight forward. The function V calls on two auxiliary functions, on FINDP to find the index in TNAME of an argument ARG, and on VLIM to expand limit-loops of the type FROM a TO b BY c. VXA and VXB evaluate arithmetic and Boolean expressions.

During the implementation of value-functions three problems had to be resolved: conversion of values to appropriate modes, efficiency of value retrieval, and serial reusability.

Modes. In order to be able to return both a real and an integer value to the same calling program, two entry points V and IV have been provided. Strings are considered as integers. The V-function itself considers all values as integers, so that no mode conversions are necessary internally.

In VLIM, where values have to be incremented and tested, separate coding is necessary for integer and for real arithmetic. For strings and descriptive-constants no incrementing or testing is done, only the lower limit is returned, followed by the upper limit, followed by the EOV marker.

In VXA, separate coding would also be required for real and for integer expressions. For now, all integers are converted to real mode and floating-point arithmetic is used throughout.

Value retrieval efficiency. If the index I of an argument ARG is known, its value can be immediately obtained, otherwise FINDP has to find the index first. ARG is saved in a local variable ARGPR so that successive calls to the same ARG do not require this lookup more than once.

The first call to VLIM with an ARG requires the initialisation of the limits LLV and ULV, and of the increment INC. It has to be

checked that if ULV > LLV then INC is positive, and if ULV < LLV then INC is negative. This initialisation is bypassed for subsequent calls to VLIM with the same ARG. The current value of the limit-loop is kept in LLV. At each reference it is incremented, tested against ULV, and stored.

Serial reusability. The value-functions are serially reusable, since several arguments may be referenced in random order from different places in the calling procedures. All information pertinent to an argument has to be kept in P-cells. The current value subfield CV of PCELL(I) is used to point to the current value of ARG. After returning a value, the next value pointer is stored in CV.

To effect serial reusability, a switch INIYES is needed for each limit-loop. INIYES = 1 means that this loop has been accessed before, and the current value of LLV is reloaded, instead of the original lower limit as specified by the user. ULV and INC have to be reinitialised also.

Of course, the complexity of the run-time module is directly related to the user-language capabilities. For a simple language, allowing only a single value per parameter, no V-functions are needed, since /CISTEP/ can be directly included in the PR procedures. For a language where limit-loops are not allowed there would be no need for VLIM. Similarly if no user-expressions are to be handled, then VXA and VXB are not required. On the other hand, more complex languages require extensions to the value-functions developed here.

5.6 ULANG performance.

The performance of the ULANG system can be evaluated on the basis of several criteria, such as implementation effort for an application, efficiency in processing of user-requests, and portability of the system.

In order to have a basis of comparison, a subsystem of a large scale application was redone with ULANG. The application chosen was the AMECO (70) Structural Design System for concrete structures. The AMECO system contains over 20,000 FORTRAN statements -- it is comparable in size and in complexity to software systems for which implementation efforts have been evaluated by others. AMECO has a flexible command language, described by Palejs & Freibergs (73), and user-command analysis and interpretation is an important aspect of the system.

The application consists of several subsystems, of which the horizontal member analysis subsystem was selected, consisting of 3000 FORTRAN statements. The lexical analyzer and two command analysis procedures were redone with ULANG. The lexical analyzer was chosen, because it was not sufficiently flexible nor efficient, and it was difficult to transport it to another system. The two analysis procedures were chosen because they were becoming too large and unwieldy. They are subject to periodic changes and additions, and had proven to be difficult to implement and to maintain. Five programmers have contributed to the old version of the programs at various times, all with an educational level of B.Sc. or M.Sc., and having at least four years of system or scientific programming experience.

To describe the command syntax, the following metalinguistic notation is introduced:

- braces indicate a choice; only one item must be chosen among the alternatives presented,

- brackets also indicate a choice, but one or more of the alternatives may be chosen,

- items enclosed in parentheses are optional, that is one or none may be selected,

- a + sign after a closing bracket or parenthesis indicates that one or more repetitions of the enclosed items are allowed,

- upper case indicates actual item-names, whereas lower case indicates values of items.

One of the commands specifies design parameters. Its syntax is as follows:

USE [design parameters] ;

The different design parameters are presented in Fig. 5-10. The other command specifies horizontal member geometry and properties. It follows the syntax:

$$\left\{\begin{matrix}\text{BEAM}\\ \text{SLAB}\\ \text{WALL}\end{matrix}\right\} \left\{\begin{matrix}\text{mark}\\ \text{mark1 TO mark2}\end{matrix}\right\} \left(\left[\begin{matrix}\text{T}\\ \text{L}\\ \text{R}\\ \text{U}\end{matrix}\right]\right) \text{(span-ft) (span-in)}$$

(X (x-ft (x-in) mark-x)$^{+}$) (labeled data) ;

The labeled data are shown in Fig. 5-11.

Implementation effort. It is reasonable to take the number of source statements as a measure of the implementation effort. Studies done at Systems Development Corp. by Nanus & Farr (64) and by Weinwurm (65) show that for large software systems

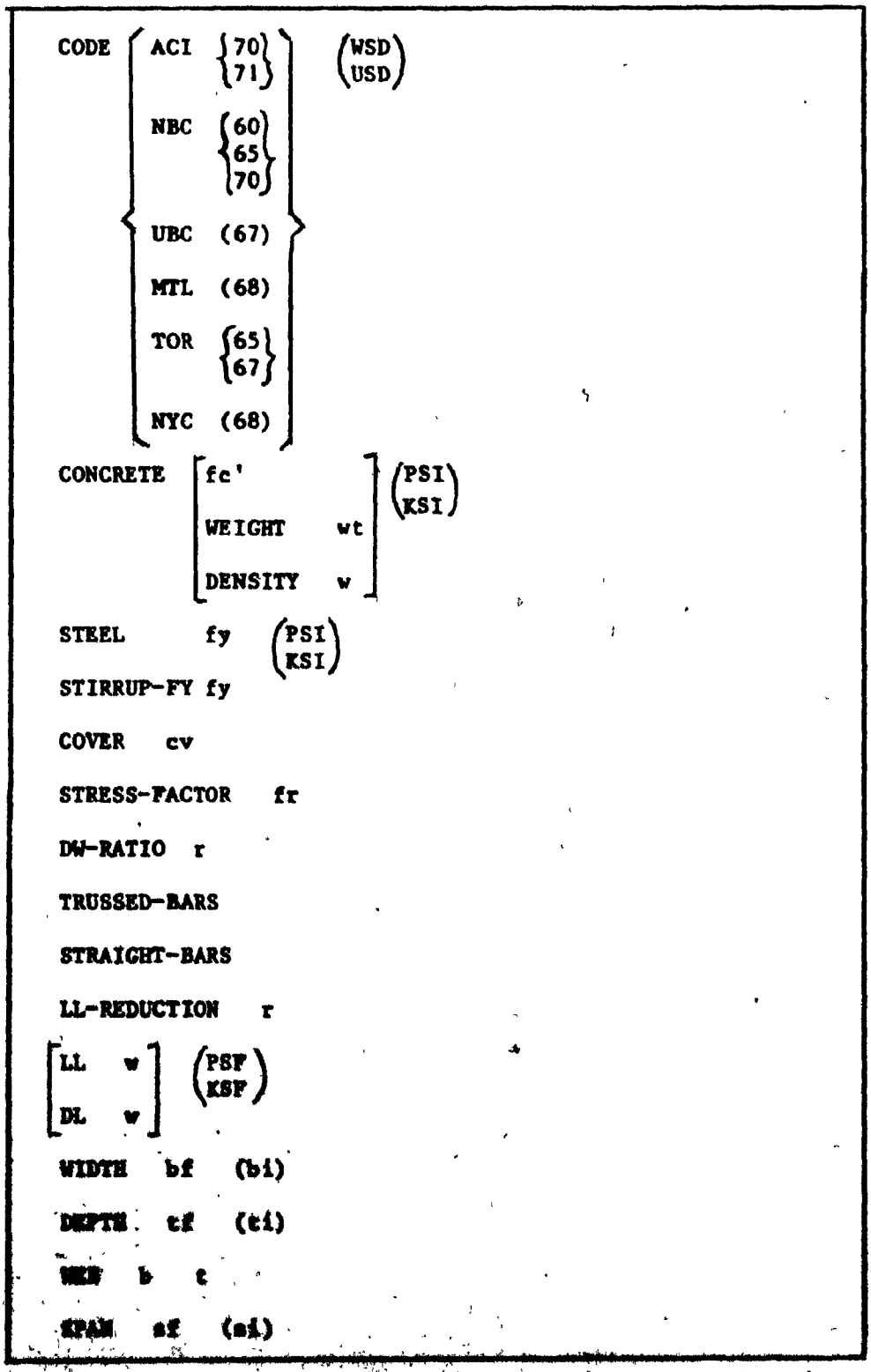


Figure 5-10. Design parameters for USE command.

```
FLOOR-WIDTH   bf   (bi)

SLAB   t

RIGIDITY  [ rg                    ]
          { [LEFT   rg1           ] }
          { [RIGHT  rg2           ] }

FRAME     [ACCURACY   a           ]
          [ITERATIONS i           ]

CONSTRUCTION   (SIMULATION)

{ SUBGRADE  [ k                   ]                ( COLS [ mk                          ]+ )
            [ VERTICAL   kv       ]                       [ FROM mka THROUGH mkb       ]
            [ ROTATIONAL kr       ]
  FOOTING   [ x                   ]
            { [XDIM   x           ] }
            { [YDIM   y           ] }
  [KX   k                         ]
  [KY   k                         ]
  [FORMULA  n                     ]
  [XY   r                         ] }
```

Figure 5-10 (continued). Design parameters for USE command.

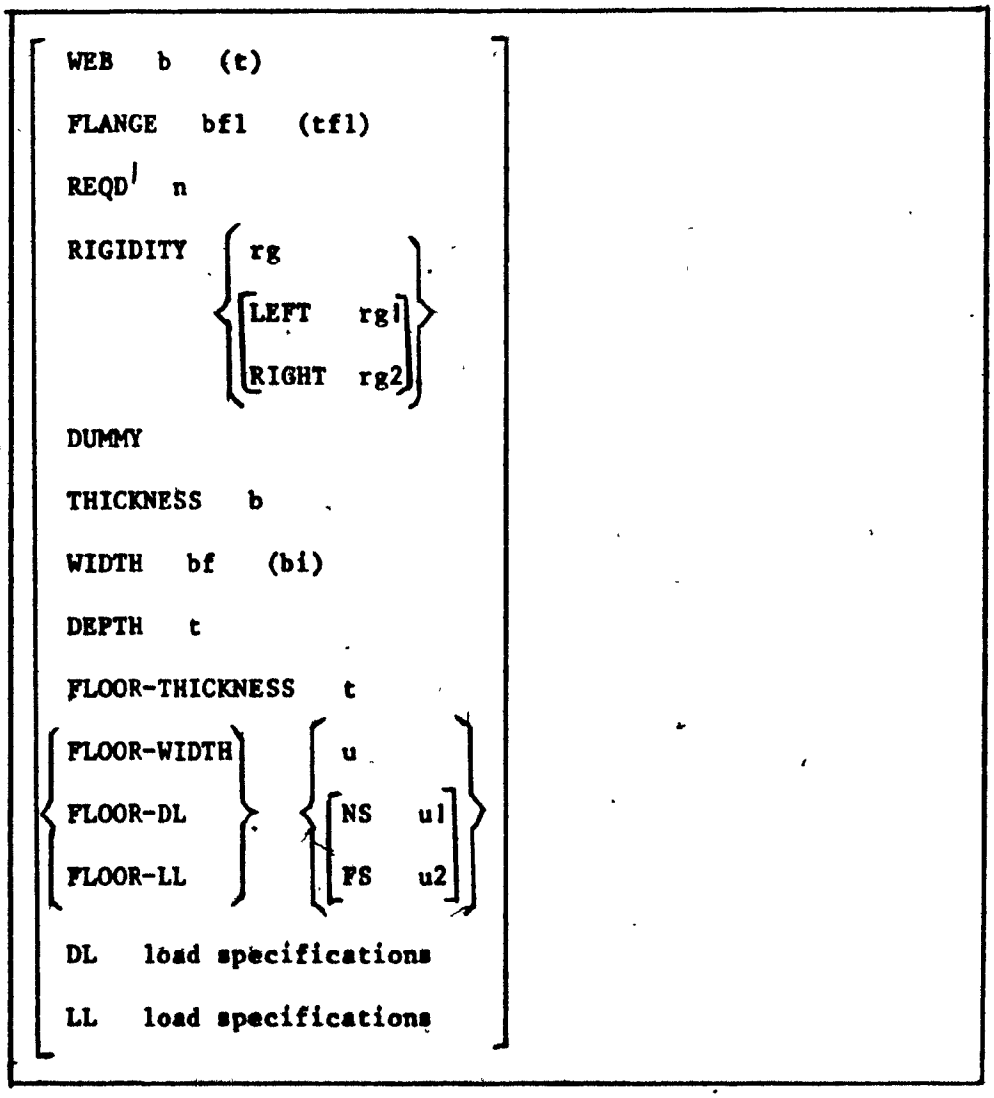


Figure 5-11. Labeled data for BEAM/SLAB/WALL command.

$$\text{Effort} = \text{constant} \times (\text{number of instructions})^{1.5}$$

The exponent 1.5 is introduced by increased communications problems between implementers for large systems. This data is confirmed by F. P. Brooks Jr. (74), ex-manager of IBM 360 software production. Brooks also summarizes data on programmer productivity from five sources, stated in "debugged" statements per man-year. An important result of these studies is that productivity stays constant in terms of source statements, whether Assembler, Fortran, PL/I, or Cobol, at about 2200 to 2400 statements per year.

Programmer productivity can be then increased by as much as five times when a high-level language is used, since each line expands into 3 to 5 words of assembly code. Cooke (74) further reports experimental evidence showing that another productivity factor of at least two can be obtained over Fortran and Algol by using a suitable problem-oriented language. An article by Boehm (73) mentions two more studies on software productivity, which come to similar conclusions. Halstead (73) also points out that economies may be realized by using higher-level languages (than the current languages of Fortran or Algol type), if the given class of applications represents a non-trivial programming work load.

In Table 5-3, the number of programming statements in the old version is compared with the corresponding ULANG version, for the modules investigated in this work. The saving in programming effort may be estimated as $(1165/470)^{1.5} = (2.5)^{1.5} = 4$, i.e. it takes four times less programming effort to produce the same software with ULANG. On the basis of programmer productivity data, one might say that the saving of 695 statements represents 3 1/2 man-months of effort.

TABLE 5-3 COMPARISON OF NUMBER OF PROGRAMMING STATEMENTS

Module name	Old version	ULANG version	Saving
Lexical analyzer			
Fortran sta.	230	50	280
Assembler sta.	100	-	
USE command			
Fortran sta.	550	224	300
ULANG setup sta.	-	26	
BEAM/SLAB/WALL command			
Fortran sta.	285	154	115
ULANG setup sta.	-	16	
TOTAL	1165	470	695

Another advantage of the reduced program length is increased perspicuity. In Fig. 5-13 two program sections performing the same task are exhibited, namely the setting of RIGIDITY LEFT and RIGIDITY RIGHT for the global variables CONSTS(1) and CONSTS(2), respectively. The parameter syntax is shown in Fig. 5-10. The value can be between 0 and 1. The complexity of the old program version (Fig. 5-13a) is largely due to checking order of words, cheking range of values, obtaining the next word of the input and determining its type and value. In the ULANG version (Fig. 5-13b) most of these tasks have

```
LINK SUBJECT TO CODE-NAME*0 TO CODE-YEAR*CODE-NAME
    SUBJECT TO UNITS*0, MODIFIER*0
CLASS KWD IS USE
CLASS SUBJECT=CODE,CONCRETE,DENSITY,WEIGHT,STEEL,COVER,STRESS-FACTOR
  SUBJECT=DW-RATIO,TRUSSED-BARS,STRAIGHT-BARS,LL-REDUCTION,LL, DL
  SUBJECT=WIDTH,DEPTH,WEB,SPAN,FLOOR-WIDTH,SLAB,RIGIDITY,LEFT,RIGHT
  SUBJECT=SUBGRADE,VERTICAL,ROTATION,FOOTING,XDIM,YDIM
  SUBJECT=FRAME, ACCURACY, ITERATIONS, CONSTRUCTION, STIRRUP-FY
  SUBJECT= KX, KY, FORMULA, XY
DCLASS CODE-NAME IS (ACI,NBC,MTL,NYC,TOR,UBC) FOR CODE
  CLASS CODE-YEAR IS YEAR*I FOR CODE
DCLASS UNITS= (WSD,USD) FOR CODE, (PSF,KSF) FOR (DL,LL)
    UNITS= (PSI,KSI) FOR (STIRRUP-FY,CONCRETE,STEEL)
CLASS MODIFIER= COLS*I FOR (SUBGRADE,FOOTING,KX,KY,FORMULA,XY)
VALUE YEAR 71 LLIM=60 ULIM=71
VALUE DENSITY 145 LLIM=50 ULIM=300
VALUE WEIGHT=150 LLIM=50 ULIM=350
VALUE COVER=1.5 LLIM=0.5 ULIM=3
VALUE STRESS-FACTOR LLIM=0 ULIM=1.
VALUE LL-REDUCTION=0.85 LLIM=0.3 ULIM=1.
VALUE RIGIDITY ULIM=1.
  LEFT=1. LLIM=0 ULIM=1.
  RIGHT=1. LLIM=0 ULIM=1.
VALUE ACCURACY LLIM=0.001 ULIM=10.
  ITERATION LLIM=10. ULIM=50000.
UDICT OPTIONS 373 FROM 431 TO 432
```

Figure 5-12. a) Setup statements for USE command.

```
TEMPLATE    BM    IS OPEN
 LINK KWD TO MKA TO MKB*O TO SHAPE TO BSPAN* TO TRANS*O TO LABEL*O TO
                                                          MODIF*O
 CLASS KWD IS BM,SLAB, WALL
  MKA IS MARK*I
  MKB IS THRU*I
 DCLASS SHAPE IS T, L, R, U
 CLASS BSPAN IS SPAN
  TRANS IS X
 CLASS LABEL= WEB, FLANGE, REQD, RIGI, LEFT, RIGHT, DUMMY, THICKNESS
  LABEL*WIDTH, DEPTH, FLOOR-THICK, FLOOR-WIDTH, FLOOR-DL, FLOOR-LL
  LABEL=DL, LL
 CLASS MODIF=(NS,FS) FOR (FLOOR-DL, FLOOR-LL, FLOOR-WIDTH)
 VALUE MARK LLIM 101 ULIM 3015
  THRU LLIM 101 ULIM 3015
 VALUE RIGIDITY ULIM=1.
  LEFT=1. LLIM=0 ULIM=1.
  RIGHT=1. LLIM=0 ULIM=1.
```

Figure 5-12 (continued). b) Setup statements for BEAM/SLAB/WALL command.

USER DICTIONARY

1	ACCU	110	44	NS	110
2	ACI	200	45	NYC	200
3	BM	300	46	OPTI	373
4	BSPA	120	47	PSF	200
5	CODE	110	48	PSI	200
6	CODN	120	49	R	200
7	CODY	120	50	REQD	110
8	COLS	110	51	RIGH	110
9	CONC	110	52	RIGI	110
10	CONS	110	53	ROTA	110
11	COVE	110	54	SHAP	120
12	CENS	110	55	SLAB	300
13	DEPT	110	56	SPAN	110
14	DL	110	57	STEE	110
15	DUMM	110	58	STIF	110
16	DWRA	110	59	STRB	110
17	FLAN	110	60	STRF	110
18	FLOO	110	61	SUBG	110
19	FLOL	110	62	SUBJ	120
20	FLOT	110	63	T	200
21	FLOW	110	64	THIC	110
22	FOOT	110	65	THRU	110
23	FORM	110	66	TO	432
24	FRAM	110	67	TOR	200
25	FROM	431	68	TRAN	120
26	FS	110	69	TRUB	110
27	ITER	110	70	U	200
28	KSF	200	71	UBC	200
29	KSI	200	72	UNIT	120
30	KWD	100	73	USD	200
31	KX	110	74	USE	300
32	KY	110	75	VERT	110
33	L	200	76	WALL	300
34	LABE	120	77	WEB	110
35	LEFT	110	78	WEIG	110
36	LL	110	79	WIDT	110
37	LLRE	110	80	WSD	200
38	MARK	110	81	X	110
39	MKA	120	82	XDIM	110
40	MKB	120	83	XY	110
41	MKDI	120	84	YDIM	110
42	MTL	200	85	YEAR	110
43	NBC	200			

Figure 5-12 (continued). c) User-dictionary for AMECO commands.

```
C     BEAM RIGIDITY
400   CONTINUE
      IF (I.EQ.NO) GO TO 302
      I=I+1
      IF (KEY(I).EQ.1) GO TO 405
      IF (WORD(I).EQ.USE(45)) GO TO 401
      IF (WORD(I).EQ.USE(46)) GO TO 403
      CALL SYNTAX(4,I,WORD(I),1H )
      GO TO 201
C     LEFT
401   IF (I.EQ.NO) GO TO 302
      I=I+1
      IF (KEY(I).EQ.1) GO TO 402
404   CALL SYNTAX(2,I,WORD(I),1H )
      GO TO 201
402   IF (WORD(I).GT.1..OR.WORD(I).LT.0.) WORD(I)=1.
      CONSTS(1)=WORD(I)
      IF (I.EQ.NO) RETURN
      I=I+1
      IF (WORD(I).NE.USE(46)) GO TO 201
C     RIGHT
403   IF (I.EQ.NO) GO TO 302
      I=I+1
      IF (KEY(I).NE.1) GO TO 404
      IF (WORD(I).GT.1..OR.WORD(I).LT.0.) WORD(I)=1.
      CONSTS(2)=WORD(I)
405   IF (WORD(I+1).EQ.USE(45).OR.WORD(I+1).EQ.USE(46)) GO TO 406
C     BOTH ENDS
      IF (WORD(I).GT.1..OR.WORD(I).LT.0.) WORD(I)=1.
      CONSTS(1)=WORD(I)
      CONSTS(2)=WORD(I)
      GO TO 208
406   IF (WORD(I+1).EQ.USE(46)) GO TO 408
      IF (WORD(I).GT.1..OR.WORD(I).LT.0.) WORD(I)=1.
      CONSTS(1)=WORD(I)
      I=I+2
      IF (I.GT.NO) RETURN
      IF (KEY(I).NE.1) GO TO 201
      IF (WORD(I+1).EQ.USE(46)) GO TO 407
409   CALL SYNTAX(5,I,WORD(I),1H )
      GO TO 208
407   IF (WORD(I).GT.1..OR.WORD(I).LT.0.) WORD(I)=1.
      CONSTS(2)=WORD(I)
      I=I+2
      IF (I.GT.NO) RETURN
      GO TO 201
408   IF (WORD(I).GT.1..OR.WORD(I).LT.0.) WORD(I)=1.
      CONSTS(2)=WORD(I)
      I=I+2
      IF (I.GT.NO) RETURN
      IF (KEY(I).NE.1) GO TO 201
      IF (WORD(I+1).NE.USE(45)) GO TO 409
      IF (WORD(I).GT.1..OR.WORD(I).LT.0.) WORD(I)=1.
      CONSTS(1)=WORD(I)
      I=I+2
      IF (I.GT.NO) RETURN
      GO TO 201
```

Figure 5-13. a) Old version for setting RIGIDITY.

```
                C RIGIDITY
0166            1300  RIG=V(M)
0167                  IF(RIG.EQ.EOV) GOTO 1320
                C BOTH ENDS
0168                  CONSTS(1)=RIG
0169                  CONSTS(2)=RIG
0170                  GOTO 2000
                C LEFT OR RIGHT END ONLY
0171            1320  CONSTS(1)=V('LEFT')
0172                  CONSTS(2)=V('RIGH')
0173                  GOTO 2000
```

Figure 5-13 (continued). b) ULANG version for setting RIGIDITY.

already been done by the preprocessor ULPREP, according to the definitions from the template USE (Fig. 5-12a).

The increased gain in program simplicity helps to avoid programming errors and to make additions and modifications simpler. For instance, in analysing the old version of the program for steel strength specifications, an error in the existing program logic was discovered, which had crept in because of the complexity of the logic.

Run-time efficiency. The old lexical analysis module, although using IBM Assembly language subroutines to improve register usage, was judged to be inefficient, hard to comprehend and to change, and difficult to transfer to another system. The ULANG scanner module was substituted in its place, with a 50 statement interface to transfer ULANG tokens to AMECO tokens. The performance of the two scanners was compared on the IBM 360 for 206 typical AMECO commands, taken from the AMECO User's Manual (70). The following results were obtained:

- old version	8.34 sec.
- ULANG version	2.96 sec.
- base time for reading and printing of cards, without scanning	2.82 sec.

The lexical analysis times are 5.52 sec. vs. 0.14 sec., giving an improvement factor of 5.52/0.14 = 39.4.

To check the command analysis timing, the lexical analysis effect was first eliminated by using the ULANG scanner for both, the old, and the ULANG versions. Since the USE and BEAM/SLAB/WALL commands are embedded among other commands, their effect is difficult to isolate. The total running times obtained for a sample, consisting of

10 examples from the AMECO User's Manual, were comparable (20 sec. vs. 15 sec.), as expected .

The execution times were also measured for the ULANG examples of Chapter 6. On the CDC 6600 it takes between 25 and 45 msec of CPU time to validate and to analyse an input command, to supply missing parameters from the template, to build up a CT-step and to display or to execute it. The corresponding cost is between 0.5 and 0.75 cents per command.

This low cost confirms the validity of the principle used in the design of ULANG: never to look up the same information twice. This principle has been adhered to at all levels.

The hardware representation of input characters is used directly as an index in the CHARCL table to assign lexical categories to characters. The input character string itself is scanned only once by the scanner ULSCAN, which functions as a deterministic finite-state automaton. The syntax analyser USERCOM in turn makes only a single pass over the categorized atom string, output by the scanner. Tie-in with semantics occurs at appropriate points by calls to the semantic routines INSERP and APPENP, which build up the CT-step. At the end of the scan over the atom string, all the user supplied information has been extracted and transfered to the CT-step.

The semantic analyser USERSEM makes one pass over the classes of the template. If the minimum number of parameters of a class has been supplied by the user, then these parameters are checked for consistency, validity, and converted the appropriate mode if necessary. If the user has not supplied all the required parameters, then default

values are taken from the template and inserted into the CT-step.

To summarize, a user's input line is processed in one pass over the character string, one pass over the atom string, and one pass through the template.

Portability. As mentined above, the old lexical analyser of AMECO was hardly transferable to the CDC 6600, because it was relying on the IBM 360 character sequence (special characters before letters), whereas it is the reverse for CDC 6600. Also the special IBM 360 Assembler routines had to be simulated on the CDC 6600. As a result, the scanning time was increased by a factor of two over the IBM version.

On the other hand, the ULANG version of the scanner was easily transfered from CDC 6600 to IBM 360. Only one table, CHARCL, defining the lexical classes of characters had to be redefined, together with a few constants from /ULDAT1/, such as word length and padding character. Among the procedures, only basic character handling routines, such as logical OR, AND, shifting, and word packing routines GET and PUT, had to be replaced to account for the differences between 6-bit and 8-bit code.

The whole ULANG system, developed on the CDC 6600, was transfered to the IBM 360 by doing the lexical changes just indicated, plus changes in the direct-access statements, which are localised in table opening and closing routines, and in template opening and closing routines.

5.7 Treatment of user-errors

Several options are open for treating errors in user-requests. One choice has been made in the present version of ULANG, but modifications to suit each application would be normally required. Errors can be detected and corrected at different levels. The various error considerations are discussed in this section.

A basic premise to keep in mind, is that the context and the area of discourse of a given application is limited, as discussed in section 1.4.1. The user works with a predefined vocabulary, which is entered into the user-dictionary at setup time (see for instance Fig. 5-12c for the vocabulary of the two AMECO commands). Any ambiguities and conflicts of usage are detected and eliminated at setup time. If a user subsequently tries to add synonyms, conflicting with prior definitions, this is also detected.

5.7.1 Standardization of names

In order to make matching possible with dictionary entries and with problem-solving procedure variables, all names have to be standardized according to some rule (see Fig. 5-3). In the present version all names are truncated to a length of 4, except for compound names, mainly because this has been defined as an acceptable practice for the AMECO language (Palejs & Freiberga (73)). In the eight years of AMECO usage, no complaints have been voiced by users about this feature. Brevity is a desirable feature for experienced users (Joyce (72)), and it is commonly used in command languages for on-line text editing.

If at the setup time of an application it turns out that unavoidable ambiguity may arise, e.g. for DIVI_DE and DIVI_DEND, then the standard length can be increased to 6 or to 8, as necessary to eliminate the ambiguity. This is done by redefining the value of the parameter LENTOK in /ULDAT1/.

Alternatively, some other method of standardization might be used. A plausible scheme, used in some applications (BCS (70)), is to eliminate vowels, except at the beginning of a word, i.e. one would obtain DVD and DVDND for the words above. It suffices to change the subroutine SPELL of the scanner to compare the current character against a list of letters to be eliminated.

5.7.2 Correction of spelling errors

In the present work no spelling error correction is attempted, this being a somewhat controversial topic. It may be envisaged to do probabilistic matching of names with user-dictionary entries, so that if no exact match has been obtained, then the closest matching neighbouring entry is taken, provided that the match reaches some predefined level, say 80%. To do such matching, a scoring function has to be added to the AFTSCAN routine of the scanner.

Based on PL/C experience at Cornell University, Morgan (70a) reports that over 80% of all spelling errors fall into one of four classes of single error:

- one letter wrong,
- one letter missing,
- an extra letter inserted,

- two adjacent letters transposed.

He also provides a detailed algorithm and an IBM 360 Assembler procedure for correcting these errors. As an alternate approach to probabilistic matching, Morgan's spelling algorithm could be called in AFTSCAN.

5.7.3 Discarding of superfluous words

Different attitudes may be adopted regarding superfluous words from the user's input sentence. Such words may be kept or discarded at the lexical, syntactical, or semantical level.

Lexical analysis level. Three possibilities are open:

a) Any unrecognized word is rejected and a user-error is indicated. This usually terminates the processing of the request.

b) Certain words may be specifically designated to be ignored. In ULANG this is done by giving such words the category zero in the dictionary.

c) If a word is not marked to be specifically discarded, and if after all standardization and/or spelling correction treatments it still remains unrecognizable, then a choice has to be made, whether to

i) keep it for further processing,

ii) discard it, and warn the user about it,

iii) discard it, and not mention it to the user, unless he has specifically requested otherwise.

The scanner routine AFTSCAN is easily adjustable to suit either choice. At present, option i) is chosen, because not all user-commands are treated by ULANG.

The ULANG system option ECHO-IN permits the user to obtain feedback of his input, i.e. the words in standard form, with categories attached. As an alternative, the original, non-standardized word could be displayed.

Syntax analysis level. Unrecognized words at the lexical level (category zero), may be useful in certain contexts at the syntax analysis level. In the present version of ULANG, the user-command syntax analyser USERCOM discards such words, except in the case of arithmetic expressions, when the name preceding the = sign is taken as the name of the expression. As an alternative, a message could be printed at this point.

For some commands, such words may have a local meaning (Notley (71)). Calls on special procedures can then be inserted.

On the other hand, simply because a word is recognized at the lexical level, say as being an integer, does not guarantee that it is used correctly, from the point of view of syntax and of semantics. For instance, in ULANG item-names may be implied under certain conditions, as discussed in section 3.4.3. If the implied item-name routine SEMIMPL cannot associate a name with a value, then an error message is printed.

5.7.4 Rejection of user-supplied values

Another reason for a user-supplied value to get rejected, is that it may be outside of the allowable range, as defined at setup time. For instance, in the AMECO USE command, a RIGIDITY specification has to be within the range 0 to 1.

The range limitation may be a result of the nature of the application or of the computing equipment used. All calculating machines have a limited word length, so that numbers can be represented up to a certain magnitude and precision. For a pocket calculator this limit may be 8 digits, or 10^8, whereas for the IBM 360 this limit is 10^{75}. Thus, in example 6.1, factorials may have to be limited to 11! for a pocket calculator and to about 70! for the IBM 360. This is illustrated in Fig. 6-2, where in the setup statements for the CALCULATE command, input values for factorials are limited between 1 and 10 -- of course, the upper limit UL could have been specified as 70 for the IBM 360.

Yet another reason for rejecting a user's input value is that it is not used in the appropriate context. Within a given application, the usage of certain parameters may be conditional upon the presence of others. In the example of chapter 4, it was specified that the method of exponential smoothing be used for projecting dividends only (Fig. 4-2). If the user forgets this, he gets warned about it. The processing may be terminated at this point, or it may be carried out regardless, at the implementer's choice.

The value ranges and processing contexts of parameters are defined at setup time, and they have to be made known to the users, or better yet, defined in conjunction with the users. The important point is that error checks of this type, if not done by the ULANG interface, have to be incorporated into the logic of the application, resulting in rather intricate logic, as illustrated in Fig. 5-12a. The advantage of ULANG is to separate out such verification functions in the interface, and to provide them automatically.

In a truly interactive version of ULANG, which, due to the equipment available, could not be implemented as yet, the user may be permitted to interrogate the ranges of validity and the processing contexts, and possibly override them.

Boehm (73) stresses the importance of validation-oriented languages, which require the programmer "to specify such items as allowable limits on variables, inadmissible states and relations between variables", for increased software productivity.

5.7.5 Default values

ULANG will supply default values for mandatory processing parameters, if asked to do so at setup time. If no default values have been set-up, and the user has not provided all the necessary values either, then an error indication is given.

The system option ECHO-OUT permits the user to display and to check all the active parameters and their values for his request before execution. From this he can clearly tell how his request has been interpreted, what, if anything, has been deleted, and what has been added by default.

CHAPTER 6 - SPECIFIC APPLICATION EXAMPLES

In the previous chapters the various aspects of ULANG have been described from the user's as well as from the programmer's points of view without the restriction to any specific application. The range of applications for which ULANG would be useful was outlined in section 1.4.3. In general, it benefits applications requiring a flexible user-interface, where activities have to be performed in a certain sequence and where each user-request can be reduced to the canonical form < command >, < $parameter_1$ >, ...,< $parameter_n$ >.

In this chapter the use of ULANG is illustrated on three specific applications, in order to provide a more integrated view of the concept. The examples chosen are somewhat simplistic for illustrative purposes, and as such, they do not represent an exhaustive view of all the capabilities or advantages of ULANG. Only one type of user-request is shown in each example in order to avoid repetition and to keep the illustrations simple.

First, the scope of each example is described. This is followed by some typical user-requests. Then the parameter structure is discussed. The corresponding ULANG setup statements and the resulting template contents are shown. Finally, the active parameters in CTSTEP are examined for a few requests and, in the case of two examples, the FORTRAN source statements of the problem-solving procedure are given, illustrating the usage of ULANG run-time facilities.

The format of displays for templates needs more explanation. It consists of one line for each class, member, and value. All classes are listed in family-order. Below each class are listed the members of that

class. Below each member its values may be listed, in turn.

Format of a class-name line:

C = < index > < class-name1 > NPR = < integer > S = < class-name2 > B = < class-name3 >

where

< index > = index of < class-name1 > in TNAME

< class-name1 > = name of this class

< class-name2 > = name of son's family

< class-name3 > = name of brother's family

NPR = < integer > indicates the number of parameters required for this class.

Format of a member-name line:

M = < index > < type > < member-name1 > IV = < implicit value > {FOR < member-name2 >}

where

< index > = index of < member-name1 > in TNAME

< type > has the following meanings D = descriptive constant,

K = keyword,

I = integer,

F = floating-point,

A = alphabetic string

< member-name1 > name of this member

< implicit-value > an integer, meaningful only for descriptive constants and keywords

FOR < member-name2 > restriction specifications (optional, as indicated by { }).

Format of a value line, for item-names:

V = < value > {FOR < member-name >} LL = < value > UL = < value >

where

< value > = an iteger, a floating-point number, or a string

LL, UL are lower and upper limit values

{FOR < member-name >} indicates an optional restriction.

6.1 Simple mathematical calculations application

This application consists of a single command to compute and display the values of some mathematical functions, such as sine, cosine, square root, factorial, integral, for an argument X, or for a range of arguments. In this case the entire application consists of a single procedure PR, which calls on subroutines to evaluate the specific functions.

User-requests. Typical user-requests are shown (underlined) in Fig. 6-1, together with the calculated results. The keyword CALCULATE needs to be entered on the first line only, since the previous keyword is implied for subsequent lines.

Parameter-structure. The parameter structure for this command is summarized in Table 6-1. The corresponding setup statements, to be entered by the implementer, as well as template contents and the user-dictionary, printed out by the system, are given in Fig. 6-2.

TABLE 6-1 PARAMETER STRUCTURE FOR CALCULATE REQUEST

Command	Function	Arguments	Units
Calculate	1. Sine	X	Degrees, radians
	2. Cosine	X	Degrees, radians
	3. Sqrt	X	-
	4. Factorial	X	-
	5. Integral	X,Y	-

```
CALCULATE SINE FOR X=50;
                .872            .766
SQUARE-ROOT OF .5 1 2 4.75;
                .500            .707
               1.000           1.000
               2.000           1.414
               4.750           2.179
FACTORIAL FOR X FROM 1 TO 20 BY 2;
     X       INPUT VALUES OUT OF RANGE
               1.000           1.000
               3.000           6.000
               5.000         120.000
               7.000        5040.000
               9.000      362880.000
INTEGRAL FOR X FROM 1 TO 5 AND Y=-7.3, 13.3,25.9 45.2 60;
               1.000           0.000
               2.000           3.000
               3.000          22.600
               4.000          58.150
               5.000         110.750
CALC SINE FOR X=12.2 RADIANS,COSINE X=15
              12.200           -.358
              15.000           -.760
FACTORIAL X=5, 7, SQUARE-ROOT X=3.456;
               5.000         120.000
               7.000        5040.000
               3.456           1.859
                                .156 SEC.
```

Figure 6-1. Typical user-requests for CALCULATE command.

A. SETUP STATEMENTS

```
LINK KWD TO FUNCTION TO ARG TO UNITS;

CLASS KWD IS CALCULATE

DCLASS FUNCTION SINE,COSINE,SQUARE-ROOT,FACTORIAL,INTEGRAL;

CLASS ARG= X, Y FOR INTEGRAL;

DCLASS UNITS IS (DEGREES,RADIANS) FOR (SINE,COSINE);

VALUE X FOR SQUARE-ROOT LL=0

VALUE X FOR FACTORIAL LL=1 UL=10

UDICT FROM 431 TO 432 BY 433 IF 461 THEN 463 OPTION 373;
```

B. TEMPLATE CALC

```
C =    8 KWD          NPR =    1 S = FUNC        B =
    M =    2 K CALC             IV =   1

C =    6 FUNC         NPR =    1 S = ARG         B =
    M =   10 D SINE             IV =   1
    M =    3 D COSI             IV =   2
    M =   11 D SQUARO           IV =   3
    M =    5 D FACT             IV =   4
    M =    7 D INTE             IV =   5

C =    1 ARG          NPR =    1 S = UNIT        B =
    M =   13 F X                IV =   0
        V =        0.000  FOR SQUARO    LL =       0.000
        V =        0.000  FOR FACT      LL =       1.000 UL =      10.000

    M =   14 F Y                IV =   0 FOR INTE

C =   12 UNIT         NPR =    1 S =             B =
    M =    4 D DEGR             IV =   1 FOR SINE      COSI
    M =    9 D RADI             IV =   2 FOR SINE      COSI
                                                .198 SEC.
```

C. USER DICTIONARY

1	ARG	120	10	INTE	200
2	BY	433	11	KWD	100
3	CALC	300	12	OPTI	373
4	COSI	200	13	RADI	200
5	DEGR	200	14	SINE	200
6	FACT	200	15	SQUARO	200
7	FROM	431	16	THEN	463
8	FUNC	120	17	TO	432
9	IF	461	18	UNIT	120
			19	X	110
			20	Y	110

Figure 6-2. Parameter structure for CALCULATE command.

```
OPTIONS ECHO-OUT

CALC INTEGRAL FOR X FROM .5 TO 7.6 BY .75, Y=1.2 3.65 7.89 9.24
                                                        24 52 45.6 45

        ACTIVE PARAMETERS
NAME     TYPE   VALUE     MEANING

KWD        K          1 CALC
FUNC       C          5 INTE
X          D FROM
           F    .500
           D TO
           F    7.60
           D BY
           F    .750
Y          F    1.20
           F    3.65
           F    7.89
           F    9.24
           F    24.0
           F    52.0
           F    45.6
           F    45.0

                .500          0.000

               1.250          1.819

               2.000          6.146

               2.750         12.570

               3.500         25.035

               4.250         53.635

               5.000         90.135

               5.750        124.110

                                          .077 SEC.
```

Figure 6-3. Active parameters for a CALCULATE request.

```
      SUBROUTINE CALC
*  TO BE WRITTEN BY THE IMPLEMENTER
*  USES FORTRAN LIBRARY FUNCTIONS SIN COS SQRT
*       A USER-WRITTEN FUNCTION FACT
*       AND AN SSP LIBRARY SUBROUTINE QTFG
      DIMENSION A1(10),A2(10),AZ(10)
      COMMON /RLNDAT/EOV
      EQUIVALENCE (EOV,IEOV)
*  SELECT A FUNCTION
    1 I=0
      L=IV(4LFUNC)
      IF(L.EC.IEOV) RETURN
*  LOOP OVER ALL VALUES
   10 ARG=V(1LX)
      IF(ARG.EC.EOV) GOTO 1
      GOTC (100,100,300,400,500),L
*  SINE COSINE
  100 IUNIT=IV(4LUNIT)
      IF(IUNIT.EQ.1) ARG=ARG*3.14/180.
      IF(L.EQ.1) Z=SIN(ARG)
      IF(L.EC.2) Z=COS(ARG)
      GOTO 600
*  SQUARE RCCT
  300 Z=SQRT(ARG)
      GOTC 600
*  FACTCRIAL
  400 Z=FACT(ARG)
      GOTO 600
*  INCREMENTAL INTEGRAL
  500 ARG2=V(1LY)
      IF(ARG2.EC.EOV) GOTO 1
      I=I+1
      IF(I.GT.10) GOTO 1
      A1(I)=ARG
      A2(I)=ARG2
      CALL QTFG(A1,A2,AZ,I)
      Z=AZ(I)
*  CUTPUT
  600 PRINT 611,ARG,Z
  611 FORMAT(1H0,5X,2F15.3)
      GOTC 10
      END
```

Figure 6-4. Problem Logic for CALCULATE command

Parameter usage. The active parameters for a typical request are shown in Fig. 6-3, together with the results. The user's input is shown underlined. The values of X are stored in the form of a limit-loop and the values of Y are simply stacked as encountered in the input. The calculation of the integral terminates when all the Y-values have been exhausted.

The corresponding problem solving procedure CALC is shown in Fig. 6-4. This procedure illustrates simple use of the V-functions by the programmer, as described in section 4.5.2. The value of FUNC is used as a switch to select the appropriate function. In this case the function evaluation is to be done by the built-in functions SIN, COS, and SQRT, by a programmer-written function FACT, and by a standard SSP library subroutine QTFG for the integration. The arguments to QTFG have been set up in the standard way required by this library routine. The EOV marker indicates when all the values of a parameter have become exhausted.

6.2 A design application

In a certain structural engineering application, an engineer may request the design or the analysis of the structural members of a building.

Typical user-requests may be stated as follows:

```
DESIGN BEAMS;
DESIGN CONCRETE COLUMNS SQUARE, LENGTH = 12
A SLAB OF THICKNESS 12 INCHES IS TO BE ANALYSED;
I WANT TO DESIGN A BEAM, L SHAPED, USING CONCRETE FC = 3500;
```

A number of design parameters are required in order to process such requests. The design parameters can be grouped into classes, as shown in Fig. 6-5, and one or more parameters are required of each class; if not

supplied by the user's input, then default values must be given instead.

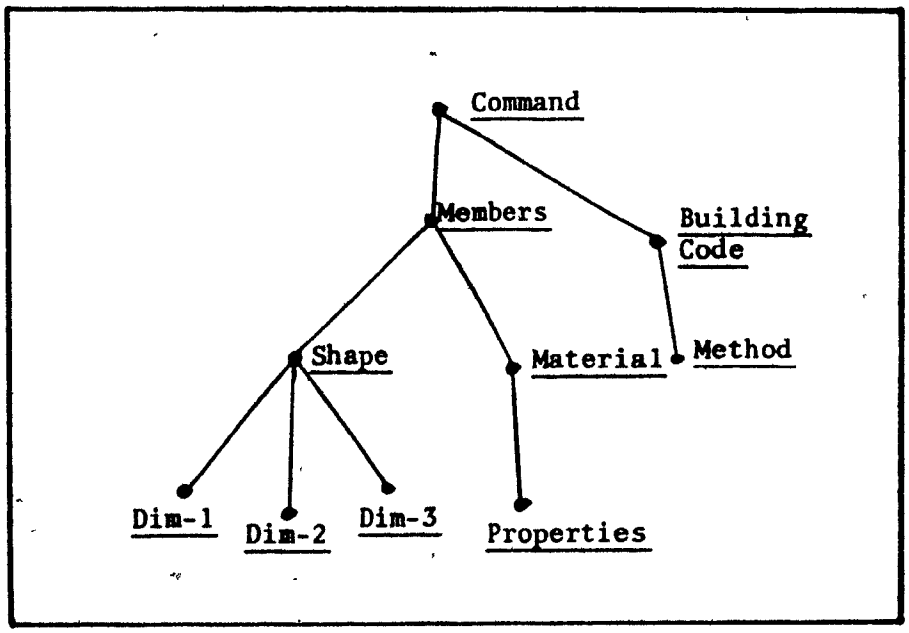


Figure 6-5. Parameter structure for DESIGN command.

This application illustrates a more complex parameter structure than in the previous example. This is typical of design applications with numerous restrictions and default values.

The setup statements to be entered by the implementer are given in Fig. 6-6. The corresponding DESIGN template generated by the system is displayed in Fig. 6-7. Since no categories have been specified, then all numeric values are floating-point by default.

Parameter usage. Although the DESIGN template is reasonably complex, the active parameter structure is simple, as shown in Fig. 6-8 for some typical design requests. The user's input is underlined. If the input values are out of range, then appropriate lower or upper limits are substituted and an informative message is printed.

```
LINK MEMBERS TO SHAPE, MATERIAL;
 SHAPE TO DIM-ONE, DIM-TWO, DIM-THREE;
 MATERIAL TO PROPERTIES=3;
 CODE TO METHOD;
DCLASS KWD= DESIGN, ANALYSE;
DCLASS MEMBERS= BEAMS,COLUMNS (WALLS,SLABS) FOR ANALYSE;
DCLASS MATERIAL= CONCRETE, STEEL FOR(ANALYSE,BEAMS,COLUMNS);
CLASS PROPERTIES= FS,(FC,DENSITY) FOR CONCRETE;
DCLASS SHAPE=(FLAT,ONE-WAY,TWO-WAY) FOR SLABS
 SHAPE=(T,U,PRISMATIC,L) FOR BEAMS
 SHAPE=(SQUARE,RECT,ROUND) FOR COLUMNS, SHEAR FOR WALLS
CLASS DIM-ONE=THICKNESS FOR (SLABS,WALLS), DIAMETER FOR ROUND
 DIM-ONE= X-DIM FOR (SQUARE,RECTANGULAR),DEPTH FOR BEAMS;
 DIM-TWO= Y-DIM FOR RECTANGULAR WIDTH FOR BEAM;
 DIM-THREE =LENGTH FOR COLUMNS,SPAN FOR  (BEAMS,SLABS,WALLS);
DCLASS CODE IS ACI, NBC, NYC;
DCLASS METHOD = WSD, USD;
VALUE FC FOR MEMBERS=3000,3000,2500,3500 LL 4(1000) UL 4(12000)
 FS FOR (BEAMS,COLUMNS)=60000,50000 LL=2(20000) UL 2(90000)
 DENSITY= 145 LL 100 UL 200
 THICKNESS FOR (ONE-WAY,FLAT)=8,8 LL=2(4) UL=2(20)
 THICK FOR (TWO-WAY,WALL)= 2(10), LL=2(6) UL 2(20)
 DIAMETER 12 LL 8, UL 100
 LENGTH= 9 LL 2 UL 20
 SPAN =20 LL=1 UL=30
 X-DIM= 12 LL 8 UL 120
 Y-DIM= 12 LL 8 UL 150
 DEPTH= 12, LL 6 UL 180;
 WIDTH 18.,LL=10.5, UL= 240;
```

Figure 6-6. Setup statements for DESIGN command.

```
C =  17 KWD          NPR =    1 S = MEMB        B =
     M =   9 K DESI           IV =  1
     M =   2 K ANAL           IV =  2

C =  21 MEMB         NPR =    1 S = SHAP        B = CODE
     M =   3 D BEAM           IV =  1
     M =   5 D COLU           IV =  2
     M =  41 D WALL           IV =  3 FOR ANAL
     M =  32 D SLAB           IV =  4 FOR ANAL

C =  30 SHAP         NPR =    1 S = DIMONE      B = MATE
     M =  15 D FLAT           IV =  1 FOR SLAB
     M =  25 D ONEWAY         IV =  2 FOR SLAB
     M =  38 D TWOWAY         IV =  3 FOR SLAB
     M =  36 D T              IV =  4 FOR BEAM
     M =  39 D U              IV =  5 FOR BEAM
     M =  26 D PRIS           IV =  6 FOR BEAM
     M =  18 D L              IV =  7 FOR BEAM
     M =  34 D SQUA           IV =  8 FOR COLU
     M =  28 D RECT           IV =  9 FOR COLU
     M =  29 D ROUN           IV = 10 FOR COLU
     M =  31 D SHEA           IV = 11 FOR WALL

C =  11 DIMONE       NPR =    1 S =             B = DIMTWO
     M =  37 F THIC           IV =  0 FOR SLAB      WALL
          V =      8.000  FOR ONEWAY    LL =   4.000 UL =    20.000
          V =      8.000  FOR FLAT      LL =   4.000 UL =    20.000
          V =     10.000  FOR TWOWAY    LL =   6.000 UL =    20.000
          V =     10.000  FOR WALL      LL =   6.000 UL =    20.000

     M =  10 F DIAM           IV =  0 FOR ROUN
          V =     12.000                LL =   8.000 UL =   100.000

     M =  44 F XDIM           IV =  0 FOR SQUA      RECT
          V =     12.000                LL =   8.000 UL =   120.000

     M =   8 F DEPT           IV =  0 FOR BEAM
          V =     12.000                LL =   6.000 UL =   180.000

C =  13 DIMTWO       NPR =    1 S =             B = DIMTHR
     M =  45 F YDIM           IV =  0 FOR RECT
          V =     12.000                LL =   8.000 UL =   150.000

     M =  42 F WIDT           IV =  0 FOR BEAM
          V =     18.000                LL =  10.500 UL =   240.000

C =  12 DIMTHR       NPR =    1 S =             B =
     M =  19 F LENG           IV =  0 FOR COLU
          V =      9.000                LL =   2.000 UL =    20.000

     M =  33 F SPAN           IV =  0 FOR BEAM      SLAB      WALL
          V =     20.000                LL =   1.000 UL =    30.000
```

Figure 6-7. Template for DESIGN command.

```
TEMPLATE DESI

C =  20 MATE        NPR =    1 S = PROP        B =
   M =   6 D CONC            IV =  1
   M =  35 D STEE            IV =  2 FOR ANAL       BEAM       COLU

C =  27 PROP        NPR =    3 S =             B =
   M =  16 F FS              IV =  0
      V =  60000.000  FOR BEAM       LL =  20000.000 UL =  90000.000
      V =  50000.000  FOR COLU       LL =  20000.000 UL =  90000.000

   M =  14 F FC              IV =  0 FOR CONC
      V =   3000.000  FOR BEAM       LL =   1000.000 UL =  12000.000
      V =   3000.000  FOR COLU       LL =   1000.000 UL =  12000.000
      V =   2500.000  FOR WALL       LL =   1000.000 UL =  12000.000
      V =   3500.000  FOR SLAB       LL =   1000.000 UL =  12000.000

   M =   7 F DENS            IV =  0 FOR CONC
      V =    145.000                   LL =    100.000 UL =    200.000


C =   4 CODE        NPR =    1 S = METH        B =
   M =   1 D ACI             IV =  1
   M =  23 D NBC             IV =  2
   M =  24 D NYC             IV =  3

C =  22 METH        NPR =    1 S =             B =
   M =  43 D WSD             IV =  1
   M =  40 D USD             IV =  2
                                                .991 SEC.
```

Figure 6-7 (continued). Template for DESIGN command.

```
OPTIONS ECHO-OUT;

DESIGN BEAMS;

        ACTIVE PARAMETERS
NAME     TYPE   VALUE      MEANING

KWD        K              1 DESI
MEMB       C              1 BEAM
SHAP       C              4 T
DEPT       F     12.0
WIDT       F     18.0
SPAN       F     20.0
MATE       C              1 CONC
FS         F   6.000E+04
FC         F   3.000E+03
DENS       F      145
CODE       C              1 ACI
METH       C              1 WSD

DESIGN COLUMNS, LENGTH=15 USD;

        ACTIVE PARAMETERS
NAME     TYPE   VALUE      MEANING

KWD        K              1 DESI
MEMB       C              2 COLU
SHAP       C              8 SQUA
XDIM       F     12.0
LENG       F     15.0
MATE       C              1 CONC
FS         F   5.000E+04
FC         F   3.000E+03
DENS       F      145
CODE       C              1 ACI
METH       C              2 USD

                                    .107 SEC.
```

Figure 6-8. Typical DESIGN requests.

```
OPTIONS ECHO-OUT;

ANALYSE L BEAMS, SPAN 34.5;
    SPAN      INPUT VALUES OUT OF RANGE

      ACTIVE PARAMETERS
NAME   TYPE  VALUE     MEANING

KWD      K            2 ANAL
MEMB     C            1 BEAM
SHAP     C            7 L
DEPT     F    12.0
WIDT     F    18.0
SPAN     F    30.0
MATE     C            1 CONC
FS       F  6.000E+04
FC       F  3.000E+03
DENS     F     145
CODE     C            1 ACI
METH     C            1 WSD

I WANT A FLAT SLAB, THICK 30, CONCRETE FC=3500 FS=5.5E4
    THIC      INPUT VALUES OUT OF RANGE

      ACTIVE PARAMETERS
NAME   TYPE  VALUE     MEANING

KWD      K            2 ANAL
MEMB     C            4 SLAB
SHAP     C            1 FLAT
THIC     F    20.0
SPAN     F    20.0
MATE     C            1 CONC
FS       F  5.500E+04
FC       F  3.500E+03
DENS     F     145
CODE     C            1 ACI
METH     C            1 WSD

                                          .113 SEC.
```

Figure 6-8 (continued). Typical DESIGN requests.

The processing logic is too lengthy and too complex to be shown here. All the required parameter values can be obtained very simply by the V-functions. Since all the parameters are single valued, an alternative would be to include the /GTSTEP/ directly in the procedure, as described in section 4.5.1.

6.3 A data-bank application

A common business application is to query a data-base. In this example there is a financial data-bank containing information about securities. One of the user-activities for this application may be to display or to change some of the data-bank information. Since the parameter structure is similar for both of these activities, the same setup statements may be used initially, but two templates are generated, DISPLAY and CHANGE.

The key to accessing any stock in the data-bank is its stock ticker-symbol. Each stock has several items of information in the data-bank, such as:

a) A classification index
   - by industry
   - by growth
   - by volatility.

b) Basic information about the security
   - name of stock
   - purchase cost.

c) Actual dividends distributed, for the last 20 quarters
   - dividend amount
   - dividend record date,

d) Periodic information, for the 15 most recent weeks

- market price
- dividend rate
- earnings.

For the purpose of this example, financial data for five stocks has been entered, with periodic information for five weeks.

User-requests. Typical user-requests are shown in Fig. 6-9, where the user's input is underlined. Simple requests involve only parameters defined at setup time, as in the case of the first two requests. The user may also make more complex requests, involving arithmetic expressions and conditions, which would screen the securities to be displayed, as in the last request where only three of the stocks mentioned meet the price to earnings ratio threshold specified.

The fifth request illustrates the useage of a class-name to shorten the input: PERIODIC-INFO stands for PRICE, DIVIDENDS and EARNINGS.

Parameter structure. The parameter structure closely parallels the structure of the data-bank. The setup statements, entered by the implementer, are shown in Fig. 6-10. The template contents are displayed by the system in Fig. 6-11. The only mandatory parameters for a request are the keyword and a stock symbol. The PERIOD parameter is activated only if any of the periodic information parameters are active.

Parameter usage. The active parameters for two requests are shown in Fig. 6-12. For the first request the option ECHO-IN prints the atom and category strings at the end of the lexical analysis phase. Before execution of a request begins, the values and types of data bank items, such as PRIC, EARN, are undefined. The storage in Polish string form of user-defined conditions is illustrated in the second request.

```
DISPLAY ≠AB≠ NAME, COST, PRICE;

STOCK   NAME                COST    PRIC
AB      ASBESTOS CORP       24.62   27.50

≠AB≠ EARNINGS, PRICE WEEK 3;

STOCK   NAME                PRIC    EARN
AB      ASBESTOS CORP       22.50    2.88

≠AB≠ EARNINGS, PRICE, EP-RATIO=PRICE/EARN, WEEK 4;

STOCK   NAME                PRIC    EARN    EPRATI
AB      ASBESTOS CORP       23.00    2.88    7.99

≠AB≠ DIVICENC-AMT QUARTER=7;

STOCK   NAME                DIVIAM
AB      ASBESTOS CORP          .25

≠CP≠ NAME, PERIODIC-INFO;

STOCK   NAME                PRIC    DIVI    EARN
CP      CANADIAN PACIFIC    79.50    3.00    5.26

DISPLAY ≠BNS≠ DIVI-AMT,PRICE,COST, XX=(PRICE+DIVI)/COST*100

STOCK   NAME                DIVIAM  COST    PRIC    XX
BNS                            .70   20.00   25.00  138.50

NAME PRICE EARN FOR STOCKS ≠AC≠ ≠BNS≠ ≠CAE≠ ≠CF≠ IF PRICE/EARNINGS>20

STOCK   NAME                PRIC    EARN
AC      ATLANTIC SUGAR       6.50     .03

STOCK   NAME                PRIC    EARN
BNS     SCOTIABANK          25.00    1.10

STOCK   NAME                PRIC    EARN
CAE     CAE INDUSTRIES      10.62     .10

                                        .255 SEC.
```

Figure 6-9. Typical user-requests for DISPLAY command.

```
LINK KWD TO SECURITY TO CLASSIFICATION*;
 SECURITY TO BASIC-INFO*
 SECURITY TO PERIODIC-INFO* TO PERIOD*PERIODIC-INFO;
 SECURITY TO ACTUAL-DIVIDENDS* TO DIVI-PERIOD*ACTUAL-DIVI;
CLASS KWD= DISPLAY, CHANGE
CLASS CLASSIFICATION=INDUSTRY, GROWTH,VOLATILITY
 SECURITY IS STOCK*A;
 BASIC-INFORMATION IS NAME*A, COST;
 PERIODIC-INFO PRICE, DIVIDENDS,EARNINGS;
 ACTUAL-DIVIDENDS= DIVIDEND-AMOUNT, DIVIDEND-DATE*I;
 PERIOD IS WEEK;
 DIVIDEND-PERIOD IS QUARTER
CATEGORY I= WEEK,QUARTER,INDUSTRY,GROWTH,VOLATILITY;
VALUE WEEK= 1 LL=1, UL15
 QUARTER =1 LL=1 UL 20
UDICT BY 433 EXEC 320 FROM 431 OF 401 REQU 310 TO 432;
 IF 461 WHILE 462 THEN 463 DO 464 OPTIONS 373;
DISPLAY
```

Figure 6-10. Parameter structure for DISPLAY command.

B. TEMPLATE DISP

```
C =  14 KWD          NPR =    1 S = SECU        B =
   M =   6 K DISP             IV =  1
   M =   3 K CHAN             IV =  2

C =  20 SECU         NPR =    1 S = CLAS        B =
   M =  21 A STOC             IV =  0

C =   4 CLAS         NPR =    0 S =             B = BASIIN
   M =  13 I INDU             IV =  0
   M =  12 I GROW             IV =  0
   M =  22 I VOLA             IV =  0

C =   2 BASIIN       NPR =    0 S =             B = PERIIN
   M =  15 A NAME             IV =  0
   M =   5 F COST             IV =  0

C =  17 PERIIN       NPR =    0 S = PERI        B = ACTUDI
   M =  18 F PRIC             IV =  0
   M =   7 F DIVI             IV =  0
   M =  11 F EARN             IV =  0

C =  16 PERI         NPR =  -17 S =             B =
   M =  23 I WEEK             IV =  0
     V =          1                    LL =          1 UL =          15

C =   1 ACTUDI       NPR =    0 S = DIVIPE      B =
   M =   8 F DIVIAM           IV =  0
   M =   9 I DIVIDA           IV =  0

C =  10 DIVIPE       NPR =   -1 S =             B =
   M =  19 I QUAR             IV =  0
     V =          1                    LL =          1 UL =          20

                                                .384 SEC.
```

Figure 6-11. Template for DISPLAY command.

```
OPTIONS ECHO-IN ECHO-OUT;

DISPLAY STOCKS ≠BNS≠ ≠CAE≠ NAME, DIVI-AMT, PRICE WEEK 10;

     DISP STOC BNS CAE NAMF DIVIAM PRIC WEEK 10
      300 110 803 803 110 110 110 110 600

      ACTIVE PARAMETERS
NAME   TYPE  VALUE     MEANING

KWD       K            1 DISP
STOC      A BNS
          A CAE
NAME                   0
PRIC                   0
WEEK      I           10
DIVIAM                 0
QUAR      I            1

STOCK   NAME                DIVIAM  PRIC
BNS     SCOTIABANK             .70  25.00

STOCK   NAME                DIVIAM  PRIC
CAE     CAE INDUSTRIES         .15  10.62

OPTIONS NOECHO-IN;

IF PRICE/EARN>10^PRICE/COST>3 THEN FOR ≠AB≠ ≠AC≠ PRICE,EARNING,
                                                        COST
      ACTIVE PARAMETERS
NAME   TYPE  VALUE     MEANING

KWD       K            1 DISP
STOC      A AB
          A AC
COST                   0
PRIC                   0
EARN                   0
WEEK      I            1
ULCOND    V            0
          V           18 PRIC
          V           11 EARN
          O /
          I           10
          O >
          V           18 PRIC
          V            5 COST
          O /
          I            3
          O >
          O ^

STOCK   NAME                COST    PRIC    EARN
AC                          1.50    6.50     .03
```

Figure 6-12. Active parameters for DISPLAY requests.

The problem solving procedure DISPLAY is shown in Fig. 6-13. It consists of 38 executable FORTRAN statements. This procedure illustrates the use of some additional run-time facilities, beyond the simple V-functions used in example 6.1. Numbers in parentheses refer to statement numbers in the source listing, in the description that follows.

The major loop (10) runs over all the stock symbols to be displayed. For each symbol mentioned in the input, another loop (200) examines each of the 10 items in the data bank.

First, the index M of each item is found in TNAME by FINDP. The first argument of FINDP is the standard-name of the item. The status of item M in this request is given by P(M,STAT). If zero, then the user has not mentioned this item, and it is inactive. If the status is one, then the item is to be displayed.

For active items, values have to be obtained from the financial data-bank (100). For certain items, the time period has also to be known (50,80). V(VPREV) is called if an EOV condition is detected, to return the preceding value of WEEK or QUARTER. Procedure GETFD is the interface with a data-base management system. In this example, only sample data for a few stocks has been entered, which is returned in VAL.

Some items are allowed to be operands of arithmetic expressions and of conditions. This is indicated by a status of 2 and of 4. For these items the value obtained from the data-bank is stored in the P-cell of M by the function VS.

Finally, item M is entered into the output buffer by procedure PUTFD, which could be some standard report generator. Here, PUTFD makes use of global data areas /TEMPL/ and /CTSTEP/ to obtain the standard-names of items and of expressions for the headings of items.

```
      SUBROUTINE DISPLAY                                                IC
C        PROBLEM SOLVING PROCEDURE FOR EXAMPLE 5.2                      IC OK DISPLAY THE LINE
      COMMON /RUNDAT/EOV,TYP,NXT,CV,STAT,TYP2,VALUE,VPREV               I       IF(FOL(1).NE.SYMBOL) GOTO 10
      EQUIVALENCE (EOV,IEOV)                                            I       PRINT 311,(FOH(J),J=1,NIT)
      COMMON /FCLINF/NIT,FOH(13),FOL(13)                                I  311 FORMAT(1H0,3A8,10(2X,A6))
      INTEGER P,FINDP,STATUS,PERIOD,VS,SYMBOL,FCL,VPREV                 I       PRINT 313,(FOL(J),J=1,NIT)
      DIMENSION FONAM(18),VAL(2)                                        I  313 FORMAT(1H ,3A8,10F8.2?)
      DATA FONAM/4LINCU,4LGROW,4LVOLA,4LNAME,6LCIVIAN,6LDIVIDA          I       NIT=3
     1,4LCCST,4LPRIC,4LDIVI,4LEARN/                                     I       GOTO 10
C                                                                       IC ERRORS
      IF(IV(3LKWD).NE.1) RETURN                                         I  910 PRINT 911,SYMBOL
C KWD IS #DISPLAY#                                                      I  911 FORMAT(1H0,A6,* HAS NO INFORMATION IN DATA-BANK*)
C LOOP OVER SYMBOLS TO BE DISPLAYED                                     I       GOTO 10
   10 SYMBCL=IV(4LSTOC)                                                 I       END
      IF(SYMBCL.EQ.IEOV) RETURN                                         I*************************************************************
C LOOP OVER ALL FINANCIAL DATA ITEMS OF SYMBOL                          I
      DO 200 ITEM=1,18                                                  I
C FIND INDEX IN FNAME                                                   I
      N=FINDP(FONAM(ITEM),STATUS)                                       I       SUBROUTINE GETFD(SYMBOL,ITEM,PERIOD,VAL,STATUS)
      IF(STATUS.EQ.8) GOTO 200                                          IC        CALLED BY DISPLAY
C IS ITEM ACTIVE IN THIS REQUEST                                        IC INTERFACE TO FILE MANAGEMENT SYSTEM
      NSTAT=P(N,STAT)                                                   IC OBTAINS VALUES OF ITEMS FROM FINANCIAL DATA-BANK
      IF(NSTAT.EQ.0) GOTO 200                                           I       INTEGER SYMBOL,PERIOD,STATUS,STOCK(5)
      GOTO(180,100,100,150,50,50,100,60,60,60),ITEM                     I       DIMENSION FD(21,5),VAL(1)
C ACTUAL DIVIDENDS NEED QUARTER                                         IC SAMPLE FINANCIAL DATA
   50 PERIOD=IV(4LQUAR)                                                 IC ITEM  WORD  DESCRIPTION
      IF(PERIOD.EQ.IEOV) PERIOD=IV(VPREV)                               IC   1     1   INDUSTRY CLASS
      GOTO 100                                                          IC   2     -   GROWTH CLASS
C PERIODIC INFORMATION, NEEDS WEEK                                      IC   3     -   VOLATILITY CLASS
   60 PERIOD=IV(4LWEEK)                                                 IC   4    2,3  NAME
      IF(PERIOD.EQ.IEOV) PERIOD=IV(VPREV)                               IC   5     4   DIVI-AMT (QUARTER 1)
C GET VALUE FROM DATA BANK                                              IC   6     5   DIVI-DATE (QUARTER 1)
  100 CALL GETFD(SYMBOL,ITEM,PERIOD,VAL,STATUS)                        IC   7     6   COST
      IF(STATUS.EQ.8) GOTO 910                                          IC   8    7-11 PRICE (WEEKS 1-5)
      IF(ITEM.LE.6) GOTO 150                                            IC   9   12-16 DIVIDEND-RATE (WEEKS 1-5)
      IF(NSTAT.EQ.1) GOTO 150                                           IC  10   17-21 EARNINGS (WEEKS 1-5)
C SOME ITEMS COULD BE IN EXPRESSIONS                                    I       DATA STOCK/2HAU,2HAC,3HBNS,3HCAE,2HCP/
      NXV=VS(P,VAL,8,8)                                                 I       DATA (FD  (J),J=1,21) /2143.,8HAMERICAN,8H CORP   ,.25,710331
      IF(PCO(NSTAT,21,NE.1) GOTO 200                                    I      124.62,27.50,23.62,22.50,23.,.25,.90,.90,.90,1.,1.,5*2.88/
C ITEM IS TO BE DISPLAYED                                               I       DATA (FD(J),J=22,42)  /1372.,8HATLANTIC,8H SUGAR  ,.25,710703
  150 CALL PUTFD(SYMBOL,ITEM,VAL,N)                                     I      21.50,6.5,6.25,6.87,6.0,6.12,3*.25,.5,.5,3*.03,.35,.35/
  200 CONTINUE                                                          I       DATA (FD(J),J=43,63)  /2320.,8HSCOTIA3A,8HNK      ,.70,710501
C SEE IF ANY USER-EXPRESSIONS, MAX 5                                    I      320.,25.,25.25,24.12,20.5,23.,3*2.70,2.9,2.9,1.1,1.1,3*1.14/
      NEXPR=0                                                           I       DATA (FD(J),J=64,84)  /2218.,8HCAE INDU,8HSTRIES  ,.15,710731
      DO 250 ITEM=11,15                                                 I      49.25,10.62,8.75,8.87,7.50,7.25,.35,.35,3*.45,.1,4*.8/
      NEXPR=NEXPR+1                                                     I       DATA (FD(J),J=85,105) /2330.,8HCANADIAN,8H PACIFIC,2*0,
      INDEX=-NEXPR                                                      I      560.37,79.5,84.,88.12,75.12,75.,3.,2.9,3*2.85,5.26,2*5.18,2 5.
      VAL(1)=VXA(INDEX)                                                 IC
      IF(INDEX.EQ.0.OR.VAL(1).EQ.EOV) GOTO 250                          IC FIND STOCK SYMBOL
      CALL PUTFD(SYMBOL,ITEM,VAL,INDEX)                                 I       DO 10 K=1,5
  250 CONTINUE                                                          I       IF(SYMBOL.EQ.STOCK(K)) GOTO 20
C CHECK IF DISPLAYING SUBJECT TO USER-CONDITIONS                        I    10 CONTINUE
      INDEX=1                                                           I       STATUS=0
      IF(VXB(INDEX).EQ.0) GOTO 10                                       I       RETURN
```

Figure 6-13. Problem-solving procedure DISPLAY.

PROBLEM LOGIC FOR EXAMPLE 6.3

```
C GET THE DATA ITEM SPECIFIED                                    IC NUMERIC ITEMS
   20 STATUS=1                                                   I   60 NIT=NIT+1
      IF(PERIOD.LT.1.OR.PERIOD.GT.5) PERIOD=1                    I      IF(NIT.GT.13) RETURN
      GOTO(110,120,130,140,150,160,170,180,190,200),ITEM         I      FOH(NIT)=TNAME(INDEX)
  110 VAL(1)=FO(1,K)                                             I      IF(ITEM.GT.10) INDEX=INDEX-MAXM
      RETURN                                                     I      IF(ITEM.GT.10) FOH(NIT)=SPNAME(INDEX)
  120 VAL(1)=0                                                   I      FOL(NIT)=VAL(1)
      RETURN                                                     I      RETURN
  130 VAL(1)=0                                                   I      END
      RETURN                                                     I
  140 VAL(1)=FO(2,K)                                             I
      VAL(2)=FO(3,K)                                             I
      RETURN                                                     I
  150 VAL(1)=FO(4,K)                                             I
      RETURN                                                     I
  160 VAL(1)=FO(5,K)                                             I
      RETURN                                                     I
  170 VAL(1)=FO(6,K)                                             I
      RETURN                                                     I
  180 VAL(1)=FO(PERIOD+6,K)                                      I
      RETURN                                                     I
  190 VAL(1)=FO(PERIOD+11,K)                                     I
      RETURN                                                     I
  200 VAL(1)=FO(PERIOD+16,K)                                     I
      RETURN                                                     I
      END                                                        I
C*************************************************************I
                                                                 I
                                                                 I
      SUBROUTINE PUTFO(SYMBOL,ITEM,VAL,INDEX)                    I
C        USED BY DISPLAY IN EXAMPLE 5.2                          I
C TO TO PUT A FINANCIAL DATA-ITEM IN OUTPUT LINE                 I
      DIMENSION VAL(1)                                           I
      INTEGER SYMBOL,FOL1                                        I
      EQUIVALENCE (FOL(1),FOL1)                                  I
      COMMON /TEMPL/MAXM,MAXV,TNAME(109)                         I
      COMMON /CTSTEP/MAXP,SPNAME(10)                             I
      COMMON /FOLINE/NIT,FOH(13),FOL(13)                         I
C                                                                I
      IF(SYMBOL.EQ.FOL1) GOTO 50                                 I
C INITIALISE LINE FOR NEW STOCK                                  I
      FOL1=SYMBOL                                                I
      FOL(2)=FOL(3)=1H                                           I
      FOH(1)=5HSTOCK                                             I
      FOH(2)=4HNAME                                              I
      FOH(3)=1H                                                  I
      NIT=3                                                      I
C ADD THIS ITEM TO LINE                                          I
   50 IF(ITEM.NE.4) GOTO 60                                      I
C NAME                                                           I
      FOL(2)=VAL(1)                                              I
      FOL(3)=VAL(2)                                              I
      RETURN                                                     I
```

Figure 6-13 (continued). Problem-solving procedure DISPLAY.

Once all the standard ten items have been examined, user-defined expressions are evaluated (250) by the VXA function. INDEX is returned as zero if no expression is active. Finally VSB is called to check if the displaying is subject to a user-defined condition e.g. IF PRICE/EARNINGS > 5. VXB of zero means that the condition is false and the displaying is bypassed, else the line assembled by PUTFD is printed out.

6.4 Summary

The above examples of applications show that ULANG is capable of handling simple and complex data structures, which can be easily specified by the implementer with a small number of setup statements. The coding of the problem solving procedures is reduced to a minimum. The execution times and costs are small.

The setup and execution statistics for the examples of this chapter are summarized in Table 6-2.

The examples have been presented here to illustrate the different aspects and ways of using ULANG, rather to demonstrate its full capabilities. It has been noted by Ross (67) and by Halstead (73) that data based on comparisons between small programs tends to underestimate the advantage of higher-level languages for large programs and that it is difficult to compare conventional systems with the new systems, because the old systems have less power and less comparable features. Nonetheless a comparison of a conventional system with ULANG is presented in section 5.6. When he is not using ULANG, the implementer has to provide himself a programming effort equivalent to produce the interface logic, represented by 1425 Fortran statements.

TABLE 6-2 SETUP AND EXECUTION STATISTICS

EXAMPLE	6.1	6.2	6.3
1. Names of templates generated	CALCULATE	DESIGN ANALYZE	DISPLAY CHANGE
2. Number of setup statements entered by implementer	8	29	17
3. Template size (number of cells used)	22	123	31
4. CPU time (msec) to generate and to display template and user-dictionary	200	1000	400
5. Total cost of generation and display (cents)	5	19	15
6. Number of executable FORTRAN statements in problem solving procedure	25	not available	38
7. Average CPU time (msec) to process one user-request (*preprocessing only)	26	30*	37
8. Total cost of processing one request (cents)	0.9	1.2*	0.9

CHAPTER 7 - CONCLUSIONS

7.1 Summary of the work

In this thesis a new system for implementing computerized application systems has been introduced. At this time, the range of applications to be computerized appears to be too wide in order to be attacked by general problem-solving systems. The need exists for many different systems, each using its own terminology of trade and easily usable by people having little programming knowledge.

ULANG provides a tool that is flexible enough to allow for the differences between various applications, yet at the same time remains practical enough to be of assistance to application-programmers. This approach allows to implement packages at a considerable savings in time and manpower. At the same time it eases the dynamic future growth and the maintenance of systems.

ULANG is an assertion in the belief "of power of tools rather than in large number of people in programming", voiced by Cheatham & Standish (70), Merrett (71), and others. It is this same belief which has prompted the introduction of other programming tools, such as input-output control systems, file management systems, and translator writing systems.

ULANG has a dual orientation: toward the users of applications, and toward the implementers. It helps them to do their jobs more efficiently and allows them to concentrate on the solution of their problems, instead of having to worry about side-issues.

Advantages to users. The user is assured of good man-machine communication. The language for making requests uses a subset of

terms from the user's profession, with emphasis on reducing programming detail. The application is easy to use at all levels of personnel. The vocabulary is the user's own, and it can be modified at any time. Synonyms and misspelled words may be allowed, extraneous words can be ignored.

Some of the input information for processing a request may be implicit, reducing the amount of input to be entered. All input is systematically checked for validity and context before being processed. This will cause fewer erroneous runs and save on the costs of processing. In other words, the user can concentrate on his request, instead of how to state it, and of where to place the commas.

Advantages to the implementers. First of all, the interface problem with the users is taken care of. The writing of the processing-logic routines is simpler and shorter, because the values of all processing parameters are readily available. There is no need to check for missing parameters, or for checking of parameter validity.

ULANG provides an organizational framework for coordinating the efforts of several programmers, working on the same application. This makes the system modular since each programmer has to concentrate only on his logic, and there is a uniform way of communication between them. Automatic documentation is provided about the structures and values of parameters. Maintenance, changes, and improvements are easy to make, permitting growth of the system.

ULANG enables application systems to be generated with a considerable savings over the customary cost in time and resources.

A tool is hereby provided, which permits the programmer to tailor an application package to suit the exact needs of his users, a wish expressed by many of them (Etude (74)), who find standard packages sold too general for their purpose. By defining the templates first, together with the users, the implementer is also forced to a "top-down" approach, proven valuable in software production (Boehm (73) , Baker (72)).

A certain amount of reluctance in using application writing systems such as this one can be expected initially from those programmers who in general have the tendency to personally "reinvent the wheel" for each new application. This has proven to be the case with programming tools introduced previously, which nonetheless have become widely accepted. Similarly, the savings in implementation efforts and the ease of use are expected to win acceptance of this system. Other key factors in gaining programmer acceptance are simplicity, perspicuity, and extendibility, which are also among the design goals of ULANG.

The usual argument of operating efficiency of custom-made applications versus those based on a general system will probably be made. It may be argued that the generality of systems such as ULANG must be paid for by increased costs in computer time and in memory. All general programming aids, beginning with compilers and with operating systems, have been subject to this criticism, yet very few users program in machine language and run stand-alone programs today. The continuously reducing unit-costs and increasing capabilities of the available technology are working against the argument for the need of maximum hardware efficiency. On the other hand, the cost of programming services is continuously rising at a rapid rate, so that the

balance is definitely in favor of tolerating some additional operating costs due to the use of general tools, but which do reduce the need of human effort.

In the light of the implementation experience with ULANG, it is even doubtful that a system is necessarily more inefficient by being more general. Such systems can be more carefully designed and incorporate efficient techniques based on recent theoretical advances, beyond the immediate reach of an ordinary programmer. This is the case here. The cost of generating a template is only 15 cents, and the cost of processing a user-request is of the order of one cent.

7.2 Further research

There are two general directions open for further research, reflecting the two problems addressed: communication and implementation.

On the communication side, the addition of metalinguistic facilities, as described by Lecarme (72), for the definition and automatic construction of syntax recognizers would allow to experiment with and to implement different user-languages. As an application of this, a second-generation setup-language could be produced for the implementers.

Addition of morphological analysis of word stems, as described by Winograd (71, p. 172), would allow for a greater flexibility in word endings, and would reduce the number of synonyms in the Dictionary.

The addition of more formal macro-processing facilities would add transformational grammar, text expansion, and text replacement capabilities to the system. This would allow for even more unrestricted

and concise user-languages than in the present version.

Incorporating and extending the template setup facilities to run-time would allow the users to make declarative statements as well as requests. This implies facilities for creating templates at run-time. The feasibility of extracting and storing information from declarative statements for the purposes of subsequent retrieval and display, has been shown by Kellog (68) in a data-base management systems, and by Thompson (69) in the REL system.

Generally, the transferring of implementer's facilities over to the users at run-time would make the system more extensible, in the sense defined by Wegbreit (71) for the ECL system. This necessitates additional ULANG system commands.

Interfacing ULANG with some existing Data-Base Management system (ASAP, Mark IV) would extend the capabilities of both systems. Another possibility would be to develop an interface to a data-base management system based on the recommendations of the CODASYL systems committee (71).

A number of features, not original yet useful, could be built into the ULANG system. One of these is the security and password aspect. An application system has to be accessed at different levels by different classes of users. This requires different layers of capabilities for accessing and changing the contents of the system. Another useful feature would be the addition of report definition and generation capabilities by the user. Of related interest are capabilities for browsing, or inspecting output produced by the processing-

logic. The user should be able to inspect parts of the total output generated, and to select portions of it to be printed in hard-copy form. This would save on unnecessary printing costs.

These proposed run-time features can be presently provided by the programmers for each application. However making them general and placing them at the users' disposal would remove this burden from the programmer, in line with the extendibility objectives of the system.

Examination of the characteristics of a generalized application system generator, such as ULANG, points to certain hardware features which may be useful, and about which further investigations are warranted. It may be advantageous to use associative memory for tables such as the Dictionary, the parameter templates and the Command Table, where the entries are located by content, i.e. by name. A small block of associative memory could be loaded with the appropriate portion of a table, which would then be searched by content.

For activities where the execution is dependent on certain conditions, specified by an IF or WHILE clause, hardware monitoring of the status of variables would be useful. This has been investigated by Morgan (70), and he has proposed a solution called "event sequenced programming".

Finally, the lexical, syntactic and semantic functions of the user-interface could be handled on a local basis, reducing the connect-time with the central computer, in a time-sharing environment. The interface functions could be carried out by a mini-computer based local terminal. The interface functions can be microprogrammed

into such a terminal. The table information could be stored on casettes, attached to the terminals. In order to use an application, the user would simply insert the appropriate casette, and then begin to compose his requests, which would be analysed locally by the ULANG interface, and the CT-steps would be again stored on casette. At that stage the central computer would be called to handle the processing-logic and to access data-banks, if necessary.

The increasing capabilities of mini-computers also make possible the handling of complete applications on the mini-computer alone. Here again, a system such as ULANG would be essential for assisting in the implementation of mini-computer based applications.

BIBLIOGRAPHY

Aho, A.V., and Ullman, J.D. The theory of parsing, translation, and compiling. Vol.1 : Parsing, 1972. Vol. 2 : Compiling, 1973. Prentice-Hall, Englewood Cliffs, N.J.

AMECO Structural Design System, User's Manual. Babylon, N.Y., 1970.

Baker, F.T. Chief programmer team. IBM Sys. J. 2, 1 (1972), 56-73.

Balzer, R.M., and Farber, D.J. APAREL - a parse-request language. Comm. ACM 12, 11 (Nov. 1969), 624-631.

BCS Portfolio Evaluation System, User's Manual, Montreal, Que., 1970.

Berkeley, E.C. Computers and Automation 19, 6B (Aug. 30, 1972), p.3.

Boehm, B.W. Software and its impact : a quantitative assessment. Datamation 19, 5 (May 1973), 48-59.

Booz, Allen, and Hamilton Inc. Data communications applications for the 1970s. Comm. ACM 14, 2 (Feb. 1971), 134-137.

Brooker, R.A. et al. The compiler-compiler. Ann. Rev. in Aut. Prog., Vol. 3, 1963, p. 229.

Brooks, F.P. Jr. The mythical man-month. Datamation 20, 12 (Dec. 1974), 45-52.

Brown, P.J. A survey of macro processors. Ann. Review in Aut. Prog. 6, Part 2, 1969, pp. 37-88.

Canning, R. G. (Ed.) The programmer-operations interface. EDP Analyzer 9, 4 (April 1971a) p. 7.

Canning, R.G. (Ed.) Application packages revisited. EDP Analyzer 9, 7 (July 1971b), p.7.

Chaucé, J. Etudes et réalisations de systèmes appliqués aux grammaires transformationnelles. Thèse de doctorat, Univ. de Grenoble, France, 1974.

Cheatham, T.E. Jr., and Standish, T.A. Optimization aspects of compiler-compilers. ACM SIGPLAN Notices 5, 10 (Oct. 1970), 10-17.

CODASYL Systems Committee. Feature Analysis of Generalized Data Base Management Systems, Technical Report, ACM, New York, May 1971.

Conway, M. Design of a separable transition-diagram compiler. Comm. ACM 6 7 (July 1963), 296-298.

Conway, R.W., Maxwell, W.L., and Morgan, H.L. On the implementation of security measures in information systems. Comm. ACM 15, 4 (Apr. 1972), 211-220.

Cooke, L.H. Jr. Programming time vs. running time. Datamation 20, 12 (Dec. 1974), 56-58.

DATA-TEXT System Manual. Dept. of Social Relations, Harvard., Cambridge, Mass., 1967.

DeRemer, F.L. Simple LR (k) grammars. Comm. ACM 14, 7 (July 1971), 453-460.

Dijkstra, E.W. Goto statement considered harmful. Letter to the Editor. Comm. ACM 11, 3 (March 1968a), 147-148.

Dijkstra , E.W. The structure of THE - multiprogramming system. Comm. ACM 11, 5(May 1968b) 341-346.

Dmytryshak, C.A. The universal consulting language alias- the investment analysis language. Proc. AFIPS 1972 Fall Joint Comput. Conf., AFIPS Press, Montvale, N.J., Vol. 41, Part 1, pp. 525-535.

Dostert, B.H. REL - an information system for a dynamic environment. REL report No.3, Caltech, Pasadena, Calif., Dec. 1971.

Etude sur les conditions d'utilisation des produits-programmes de gestion et d'administration. Rapport présenté au Ministère de l'Industrie et de la Recherche., I.U.T., Univ. de Montpellier II, France, Oct. 1974.

Feldman, J. A formal semantics for computer languages and its application in a compiler-compiler. Comm. ACM 9, 1(Jan. 1966), 3-9.

Feldman, J., and Gries, D. Translator writing systems. Comm. ACM. 11, 2-(Feb. 1969), 77-113.

Fenves, S.J. et al. STRESS User's Manual. MIT Press, Cambridge, Mass., 1964.

Floyd, R.W. Syntactic analysis and operator precedence, J. ACM 10, (July 1963), 316-333.

Frank, W.L. Software for terminal-oriented systems. Datamation 14, 6 (June 1968), 30-34.

Gawlik, J.H. MIRFAC : a compiler based on standard mathematical notation and plain English. Comm. ACM 6, 9(Sept. 1963), 545-548.

GIS Application description manual H20-0574. IBM systems reference library, White Plains, N.Y., 1968.

Gray, J.C. Compound data structures for computer-aided design; a survey. Proc ACM 22nd Nat. Conf., 1967, pp. 355-365.

Green, B.J. Jr., Wolf, A.K., Chomsky, C., and Laugherty, K. BASEBALL : an automatic question answerer. In Feigenbaum, E.A., and Feldman, J. (Eds.), Computers and Thought, McGraw-Hill, New York, 1963, pp.207-216.

Gries, D. Compiler Construction for Digital Computers, Wiley, New York, 1971.

Guilliano, V.E. Comments. Comm. ACM 8, 1 (Jan. 1965), 69-70.

Halpern, M.. Foundations of the case for natural-language programming. IEEE Spectrum 4, 3 (March 1967), 140-149.

Halpern, M.I. Toward a general processor for programming languages. Comm. ACM 11, 1(Jan 1968), 15-25.

Halstead, M.H. Language selection for applications. Proc. AFIPS 1973 Nat. Comput. Conf., Vol. 42, pp. 211-214.

Harrison, M.C. BALM - An extendable list processing language. Proc. AFIPS.1970 Spring Joint Comput. Conf., Vol. 36, pp.507-511.

Hershey, E.A. et al. PSL/II Language Specifications, Version 1.0. ISDOS working paper No. 68, Univ. of Michigan, Dept. of Industrial and Operations Engineering, Ann Arbor, Mich., Feb. 1973.

Holland, S.A. The remote inquiry of data bases. Datamation 16, 15 (Nov. 15, 1970), 54-59, 62.

Homa, D. Private communication. Xoma Systems Ltd., Montreal, Que., 1970.

Hurwitz, A., and Citron, J.P. GRAF : Graphic additions to FORTRAN. Proc. AFIPS 1967 Spring Joint Comput. Conf., Vol. 30.

IBM Cambridge Scientific Center report 320-2032, An introduction to CP-67/CMS, Cambridge, Mass., 1969.

ICP Software Directory. Vol. I, ICP Inc., Indianapolis, Ind; 1974.

Johnson, W.L., Porter J.H., Ackley, S.I., and Ross, D.T. Automatic generation of efficient lexical processors using finite state techniques. Comm. ACM 11, 12 (Dec. 1968), 805-813.

Joyce, S.M. The development of an interactive statistical language. ONLINE 72 Conf. Proc., Brunel Univ., Uxbridge, England, Sept. 1972, Vol.1, pp. 477-496.

Kellogg, C.H. A natural language compiler for on-line date management. Proc. AFIPS 1968 Fall Joint Comput. Conf., Vol. 33, Part 1, pp.473-492.

Knowlton, K.C. A computer technique for processing animated movies. Proc. AFIPS 1964 Spring Joint Comput. Conf., Vol. 25, pp. 67-87.

Lalonde, W.R., Lee, E.S., and Horning, J.J. An LALR(k) parser generator. Proc. IFIP Congress 1971. North Holland Pub. Co., Amsterdam, Vol. TA-3. pp. 153-157.

Leavenworth, B.M., and Sammet, Jean E. An overview of nonprocedural languages. Proc. ACM SIGPLAN Symposium on Very High Level Languages, Santa Monica, Calif., March 1974, pp. 1-12.

Lecarme, O. Un générateur d'analyseurs syntaxiques. Document de travail No. 27, Dépt. d'informatique, Univ. de Montréal, Montréal, P.Q., Aug. 1972.

Lecarme, O. Un générateur d'analyseurs lexicaux. Document de travail No. 40. Dépt. d'informatique, Univ. de Montréal, Montréal, Qué., June 1973.

LISP 1.5 Programmer's Manual. MIT Computation Center, Cambridge, Mass., 1962.

Lombardi, L.A., and Raphael, B. LISP as the language for an incremental computer. In Berkeley, E. (Ed.), The Programming Language LISP : its operation and applications, Information International Inc., Cambridge, Mass., 1964, pp.204-212.

Martin, T.H., and Guertin, R.L. Language decisions made while designing an interactive information retrieval system. Proc. SIGPLAN - SIGIR Interface Meeting, Gaithersburg, Md., Nov. 1973, pp. 86-90.

McKeeman, W.R. An approach to computer language design. Technical report CS-48, Comp. Science Dept., Stanford Univ., Calif., 1966.

McKeeman, W.R., Horning, J.J. and Wortman, D.B. A Compiler Generator, Prentice-Hall, Englewood Cliffs, N.J., 1970.

McLure, R.M. An appraisal of compiler technology. Proc. AFIPS 1972 Spring Joint Comput. Conf., AFIPS Press, Montvale, N. J., Vol. 40, pp. 1-9.

Merrett, T.H. General programs for management systems. Information Processing Letters 1, (1971), 17-20.

Morgan, H.L. Spelling correction in systems programs. Comm. ACM 13, 2 (Feb. 1970a), 90-94.

Morgan, H.L. An interrupt based organization for management information systems. Comm. ACM 13, 12 (Dec. 1970b), 734-739.

Nanus, B., and Farr, L. Some cost contributors to large-scale programs. Proc. AFIPS 1964 Spring Joint Comput. Conf., Vol. 25, pp.239-248.

Nelson, W., Phillips, Mary, and Thumhart, L. More effective computer packages for applications. Proc. AFIPS 1973 Nat. Comput. Conf., Vol. 42, pp. 607-614.

Notley, MG. A model of extensible language systems. ACM SIGPLAN Notices 6, 12 (Dec. 1971), 29-38.

Paleja , A.A., and Freibergs, I.F. Computerized automatic design of concrete buildings. Comput. and Structures 3 (July 1973), 937-953.

Parnas, D.L. Information distribution aspects of design methodology. Proc. IFIP Congress 1971, North Holland Pub. Co., Amsterdam, Vol. TA-3, pp.26-30.

Raphael, B. SIR : a computer program for semantic information retrieval. In Minsky, M. (Ed.), Semantic Information Processing, The MIT Press, Cambridge, Mass., 1968, pp.33-145.

Rice, J.R., and Rosen, S. NAPSS - a numerical analysis problem solving system. Proc. ACM 21st Nat. Conf., 1966, pp.51-56.

Ross, D.T. The AED approach to generalized computer-aided design. Proc. ACM 22nd Nat. Conf., 1967, pp. 367-385.

Sammet, Jean E. Programming Languages : History and Fundamentals, Prentice-Hall, Englewood Cliffs, N.J., 1969, 727-731.

Sammet, Jean E. An overview of programming languages for specialized application areas. Proc. AFIPS 1972 Spring Joint Comput. Conf., AFIPS Press, Montvale, N.J., Vol. 40, pp.299-311.

Samuel, I.N. Synthesis and Analysis : a flexible technique for processing command language. IEE Transact. on Computers C-18, 11 (Nov.1969), 1053-1061.

Schlesinger, S., and Sashkin, L. POSE : a language for posing problems to a computer. Comm. ACM 10, 5 (May 1967), 279-285.

Simmons, R.F. Answering English questions by computer : a survey . Comm. ACM 8, 1 (Jan. 1965), 53-70.

Simmons, R.F. Natural language question-answering systems : 1969. Comm. ACM 13, 1 (Jan. 1970), 15-30.

Thompson, F.B., and Dostert, B.H. The future of specialized languages, Proce. AFIPS 1972 Spring Joint Comput. Conf., AFIPS Press, Montvale, N.J., Vol. 40, pp.313-319.

Thompson, F.B., Lockemann, PC., Dostert, B.H., and Deverill, R.S. REL : a rapidity extensible language system. Proc. ACM 24th Nat. Conf., 1969, pp. 399-417.

Wegbreit, B. The ECL programming system. Proc. AFIPS 1971 Fall Joint Comput. Conf., AFIPS Press, Montvale, N.J., Vol. 39, pp. 253-262.

Weinwurm, G.F. Research in the management of computer programming. Report SP-2059, System Development Corp., Santa Monica, Calif., 1965.

Weiss, E. Interactive Information Retrieval Systems. Ph.D. Thesis, Cornell Univ., Ithaca, N.Y., 1970.

Weizenbaum, J. ELIZA - a computer program for the study of natural language communication between man and machine. Comm. ACM 9, 1 (Jan. 1966), 36-45.

Winograd, T. Procedures as a representation for data in a computer program for understanding natural language. MAC TR-84, Project MAC, MIT, Cambridge, Mass., Feb. 1971.

Wirth, N., and Weber, H. EULER - A generalization of ALGOL and its definition. Comm. ACM. 9, 1 (Jan. 1966), 13-25, and 2(Feb. 1966), 89-99.

Woods, W.A. Transition network grammars for natural language analysis. Comm. ACM 13, 10(Oct. 1970), 591-606.

APPENDIX A - FORMAL DESCRIPTION OF SYNTAX

The syntax is formally described, in a metalinguistic notation, similar to BNF. In the notation used, < x > stands for certain strings of symbols, collectively referred to as "x". Anything enclosed in the angular braces < > is called a non-terminal symbol, meaning that it can be composed of other symbols, either non-terminal or terminal. Any symbol to the right of a := sign, which is not enclosed in < > is a terminal symbol at that level of syntax. The meta-symbol := means "may be composed of", and | indicates a choice.

To avoid recursive definitions for an arbitrary number of elements in a definition, and to make the definition more readable, braces { } and square brackets [ ] are introduced, with the meaning that anything enclosed in braces { } means zero or more occurrences of the contents of the braces, and anything enclosed in square brackets [ ] is optional, that is, it may occur once, or it may be omitted. &denotes the null string. This notation is used by Gries (71), and others. For instance, given the definition

<x> := <y>|<z>

<y> := {A|B}

<z> := C|D

then valid input strings of x would be :

&, A, B, C, D, AA, BA, BBB, ABBBAAB, or any combination of A and B.

First, the syntax for the lexical analyzer is defined in section A.1. Capital letters and special characters denote actual symbols of the input string. The meaning of the symbols suggested here may of course be changed to suit a particular application by simply redefining the lexical

classes in the CHARCL table of /ULDAT1/. The output from the lexical phase is a categorized atom string. These atoms become terminal symbols at the next syntax level ; there they are denoted in lower-case without angular braces, e.g. item-name stands for any item-name.

The syntax of the user-language implemented in this work is defined in section A.2. The syntax of arithmetic and Boolean expressions is shown separately in section A.3. The syntax of the setup language is given in section A.4.

A.1 Lexical syntax

<letter> := A|B|C .....|Z

<separator> := -|_

<name> := <letter>{<separator>|<letter>}

<digit> := 1|2|3|4|5|6|7|8|9|0

<integer> := <digit>{<digit>}

<fraction> := {<digit>}·{<digit>}

<exponent> := E[<aop>]<integer>

<real> := <fraction>[<exponent>]|<integer><exponent>

<numeric-cons> := [-]<integer>|[-]<real>

<string> := '{any input symbol}'

<position-marker> := THIS|NEXT|PREVIOUS

<direction-marker> := AGO|HENCE

<limit-descriptor> := FROM|TO|BY

<cond-descriptor> := IF|WHILE

<cond-trailer> := THEN|DO|ε

<equals> := =|IS

<or> := ∨

<and> := ∧

<not> := ¬

<relop> := <equals>|<|>

<aop> := +|-

<mop> := *|/

<exop> := ↑

A.2 User-language syntax

<request> := <immediate-request>|<delayed-request>

<immediate-request> := <sentence>

<delayed-request> := REQUEST ; {<sentence>}EXECUTE ;

<sentence> := {<data} [keyword] {<data>} ;

<data> := <data-name>|<value>|<expression>|<integer ({item-name|<value>})
|item-name <limit>|<condition>

<data-name> := item-name [OF<qualifier >]|expr-name|class-name

<value> := descriptive-cons|string|numeric-cons

<limit> := {limit-descriptor <value>}

<condition> := cond-descriptor <bool-expression> cond-trailer

<expression> := name equals <arith-expression>

<qualifier> := <qualifier-constant>|<qualifier-variable>

<qualifier-constant> := item-name <value>

<qualifier-variable> := position-marker item-name
|item-name position-marker [aop integer]
|integer item-name direction-marker

A.3 Arithmetic and Boolean expression syntax

<bool-expression> := <bool-term> {or <bool-term>}

<bool-term> := <bool-factor> {and<bool-factor>}

<bool-factor> := <predicate>|(<bool-expression>)|not <bool-factor>

<predicate> := <arith-expression> relop <arith-expression>

<arith-expression> := <term>{aop <term>}

<term> := <p-factor>{ mop <p-factor>}

<p-factor> := <factor> {exop <factor>}

<factor> := (<arith-expression>)|item-name[OF<qualifier >]|expr-name
|numeric-cons

A.4 Setup-language syntax

```
DEFINE keyword ;

LINK <class-spec> TO {<class-spec>[<npr>]}[TO{<class-spec>[<npr>]}] ;

<npr> := # integer | #<class-spec>

<class-spec> := class-name|KWD

CLASS   class-name [equals]{<item-spec>[FOR<restrict-spec>]} ;

<item-spec> := item-name [#<cat-spec>]|({item-name[#<cat-spec>]})

<cat-spec> := A | F | I

<restrict-spec> := <class-spec>|<member-spec>|({<member-spec>|<class-spec>})

<member-spec> := item-name|descriptive-cons|keyword

DCLASS <class-spec>[equals]{<dck-spec>[FOR<restrict-spec>]} ;

<dck-spec> := descriptive-cons | keyword

VALUE item-name [FOR<restrict-spec>] equals{<value-spec>}[LL{<value-spec>}]
      [UL{<value-spec>}] ;

<value-spec> := numeric-cons|string|integer ({numeric-cons|string})

CATEGORY {<cat-spec>[equals]{item-name}} ;

DISPLAY ;

DUMP-T ;

OPTIONS [ECHO-IN][ECHO-OUT][DUMP-P][NOECHO-IN][NOECHO-OUT][NODUMP-P] ;

SYNONYM   dictionary-name [equals]{synonym} ;

UDICT{dictionary-name  dictionary-type} ;
```

APPENDIX B - LISTING OF SOURCE-PROGRAM

In this appendix the FORTRAN IV program listings of the ULANG system are presented. The programs are grouped into five modules, summarized in Table B-1. Within each module the principal program is listed first, followed by other subroutines in alphabetical order. Within each module the pages are numbered consecutively. The number of cards given does not include comments cards.

TABLE B-1 SOURCE PROGRAM STATISTICS

	Cards	Entry points	Pages
1. General purpose routines, used by the other modules	309	9	4
2. Lexical analyser module	256	6	4
3. Preprocessor or interface module	860	20	12
4. Value-functions for run-time	311	9	4
5. Setup module	908	19	12
Totals	2644	63	36

```
[illegible] COMUSF
C [illegible] CONSTANTS FOR ULANG SCANNER AND USER-INTERFACE
C [illegible] SETLP
C            /ULDAT1/ HAS SCANNER CONSTANTS ONLY
      COMMON/ULDAT1/LENALF,LENTOK,CHARC0,CHARCL(63),KLB,KLE,ECL
[illegible]
     [illegible],SEPAR,DASH,DESCR,DIGIT,DECIM,QUOTE,COLON,TERMIN
     [illegible],LABELS,STRINGS,FNUMS,INUMS,SEPCP,PLUSS,MINUSS

      INTEGER    LENALF,LENTOK,CHARC0,CHARCL,KLB,KLE,EOL,MXTCK
     [illegible],SEPAR,DASH,DESCR,DIGIT,DECIM,QUOTE,COLON,TERMIN
     [illegible],LABELS,STRINGS,FNUMS,INUMS,SEPCP,PLUSS,MINUSS

C     /ULDAT2/ HAS CONSTANTS USED BY BOTH ULPREP AND SETLP
      COMMON/ULDAT2/TYP,NXT,BRO,SON,NPR,FAM,RES,MVAL,CV,LIM,VALUE
     [illegible],TYP2, KNAME, MXDICT,MXSPNAM,MXTNAM,MXTCEL,MXPCEL
     [illegible],FNUM,INUM,OPER,KWD,DESCON, KWDCL,ITNAME,CLNAME
     [illegible],KEYWL,REQU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TO,OF
     [illegible],ULTYP,LLTYP, ECV,ECVCAT,LIMIT,KCND
[illegible]
     [illegible],FNUM,INUM,OPER,KWD,DESCON, KWDCL,ITNAME,CLNAME
     [illegible],KEYWL,REQU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TO,OF
     [illegible],ULTYP,LLTYP, ECV,ECVCAT

C     ------------------------------------------
C             DEFINITIONS FOR ULDAT1
      DATA LENALF,LENTOK,KLB,KLE,ECL,MXTOK/10,4,600,105,8,63/
C     [illegible] CHARACTER SET IN TERMS OF CHAR CLASSES AND OPERATORS
      DATA [illegible],CHARCL
     [illegible],21*100,800,601,602,603,604,605,606
     [illegible],551,300,561,562,590,580,0,543,0,0,700,1000
     [illegible],200,0,510,520,570,1000,541,545,0,0,800,1000/
[illegible]
      DATA [illegible],SEPAR,DASH,DESCR,DIGIT,DECIM,QUOTE,COLON,TERMIN
     [illegible],6,7,8,9,10/
[illegible]
      DATA [illegible],LABELS,STRINGS,FNUMS,INUMS,SEPCP,PLUSS,MINUSS
     [illegible],400,700,600,501,551,552/
C     ------------------------------------------
C             DEFINITIONS FOR ULDAT2
C     [illegible] FOR TCELLS AND PCELLS
      DATA [illegible],NXT,BRO,SON,NPR,FAM,RES,MVAL,CV,LIM,VALUE,STAT,TYP2
     [illegible] 3,3,20, 4,5/  ,KNAME/3LKWD/
C     [illegible] (FOR MXTCEL,MXPCEL GIVE ONLY 1/2 DIMENSION)
      DATA MXDICT,MXSPNAM,MXTNAM,MXTCEL,MXPCEL/63,10,109,200,200/
[illegible]
      DATA [illegible],STRING,FNUM,INUM,OPER,KWD,DESCON,KWDCL,ITNAME,CLNAME
     [illegible],KEYWL,REQU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TO,OF
     [illegible],9,2,2,10,11,12,13,14,37,371,550,520,552,55,543,432
     4,400/
      DATA LIMIT,KCND/43,46/
C TCELL TYPE IDS  AND EOV MARKER
      DATA RESTYP,ULTYP,LLTYP /255,254,253/,ECV,ECVCAT/999999,900/
      END
```

```
I*****************************************************************
I
I
I
I       SUBROUTINE ERROR(I,N)
IC IC PRINT ERROR MESSAGES
IC FOR ERROR NUMBER N
IC DETERMINE ERROR TYPE
I        NN=N/100
I        NNN=N-NN
I        IF(NNN.LE.0) GOTO 9000
I        IF(NN.EQ.3) GOTO 300
I        IF(NN.EQ.4) GOTO 400
I        IF(NN.EQ.5) GOTO 500
I        IF(NN.EQ.10) GOTO 1000
I        GO TO 9000
IC
IC USER SYNTAX ERRORS
I   300 II=I-1
I        PRINT 341,II
I   341 FORMAT(* ---- WORD *,I3,)
I        GOTO(351,352,353,354,355,356,357,9000,9000,360),NNN
IC          CALLED FROM USERCOM
I   351 PRINT 301
I   301 FORMAT(1H+,T15,*UNRECOGNIZABLE NAME*)
I        RETURN
IC          CALLED FROM USERCOM
I   352 PRINT 302
I   302 FORMAT(1H+,T15,*ILLEGAL LEFT BRACKET*)
I        RETURN
IC          CALLED FROM USERCOM
I   353 PRINT 303
I   303 FORMAT(1H+,T15,*ILLEGAL OPERATOR*)
I        RETURN
IC          CALLED FROM USERCOM
I   354 PRINT 304
I   304 FORMAT(1H+,T15,*UNBALANCED BRACKETS*)
I        RETURN
IC          CALLED FROM LIM
I   355 PRINT 305
I   305 FORMAT(1H+,T15,*ILLEGAL LIMIT RANGE SYNTAX*)
I        RETURN
IC          CALLED FROM SEMCLA
I   356 PRINT 306
I   306 FORMAT(1H+,T15,*CLASS-NAME DOES NOT EXIST*)
I        RETURN
IC          CALLED FROM SEMIMPL
I   357 PRINT 307
I   307 FORMAT(1H+,T15,*ITEM IGNORED*)
I        RETURN
IC          CALLED FROM ULKWD
I   360 PRINT 310
I   310 FORMAT(1H+,T15,*ILLEGAL KEYWORD, LINE IGNORED*)
I        RETURN
```

```
[illegible]),NNN
[illegible] APPEND
[illegible],I
[illegible] ---- NO NAME FOR THIS VALUE *,I10)
[illegible] NEXT
[illegible]
[illegible] ---- CLASS-NAME NOT FOUND IN TEMPLATE *,I10)
[illegible]
[illegible]
[illegible],560,561,562,563,564)
[illegible]
[illegible] EXPECTED*)
[illegible] NAMES*)
[illegible] ERROR*)
[illegible] MEMBER NAME*)
[illegible] NOT FOUND IN TEMPLATE*)
[illegible] CLASS NAME*)
[illegible]*TOO MANY RESTRICTIONS*)
[illegible] VALUE SPEC*)
[illegible]*ILLEGAL DEFINE SPEC*)
[illegible] DOES NOT EXIST*)
[illegible]
[illegible] PRINT [illegible]
[illegible] FORMAT([illegible],T15,*ILLEGAL TEMPLATE NAME*)
      RETURN
[illegible] PRINT [illegible]
[illegible] FORMAT([illegible],T15,*ILLEGAL CATEGORY SPEC*)
```

```
I       RETURN
I  564 PRINT 514
I  514 FORMAT(1X,T15,*ILLEGAL CELL SUBFIELD SPEC*)
I       RETURN
IC
IC CAPACITY EXCEEDED, SHOULD INCREASE DIMENSIONS
I 1000 PRINT 1050,I
I 1050 FORMAT(* ---- *,I5)
I       GOTO(1051,1052,1053,9000,1055,1056,9000,1058,9000,9000,1061,1
I      .1063),NNN
I 1051 PRINT 1001
I 1001 FORMAT(1X,T15,*TOO MANY N-CELLS*)
I       RETURN
I 1052 PRINT 1002
I 1002 FORMAT(1X,T15,*TOO MANY V-CELLS*)
I       RETURN
I 1053 PRINT 1003
I 1003 FORMAT(1X,T15,*TOO MANY DICTIONARY ENTRIES*)
I       RETURN
I 1055 PRINT 1005
I 1005 FORMAT(1X,T15,*TOO MANY PARAMETER-CLASSES*)
I       RETURN
I 1056 PRINT 1006
I 1006 FORMAT(1X,T15,*TOO MANY KEYWORDS FOR THIS APPLICATION*)
I       RETURN
IC          CALLED FROM EXPAND, ULSCAN
I 1058 PRINT 1008
I 1008 FORMAT(1X,T15,*TOO MANY WORDS IN THIS LINE*)
I       RETURN
I 1061 PRINT 1011
I 1011 FORMAT(1X,T15,*TOO MANY SPECIAL NAMES*)
I       RETURN
I 1062 PRINT 1012
I 1012 FORMAT(1X,T15,*TOO MANY VALUES FOR THIS STEP*)
I       RETURN
IC          CALLED BY AEX, BEX
I 1063 PRINT 1013
I 1013 FORMAT(1X,T15,*EXPRESSION TOO LONG*)
I       RETURN
IC          UNDEFINED ERROR
I 9000 PRINT 11,N,I
I   11 FORMAT(5X,*UNRECOGNIZABLE ERROR *,2I5)
I       RETURN
I       END
I***************************************************************
I
I
I
I       SUBROUTINE EXPAND(NSHELL,KOND)
IC TO PUSH ATOMS RIGHT BY NSHELL PLACES
I       COMMON /TOKEN/ATOM(63),CAT(63),A,MAXA,K,KK
IC
I       COMMON /ULDAT1/ DUMMY(69),MXTOK
I       INTEGER ATOM,CAT,A,MAXA,K,KK
```

```
                      I      IF(ECF,5) 20,10
                      I   11 FORMAT(80R1)
                      I   10 MAXI=80
                      I      IPTR=0
                      IC
                      I      BRIAI 13,INBUF
                      I   13 FORMAT(1H0,80R1)
                      I      RETURN
                      IC END OF FILE
                      I   20 IEOF=1
                      I      RETURN
                      I      END
                      I*****************************************************************
                      I
                      I
                      I
                      I      SUBROUTINE NEXTA
                      IC TO OBTAIN NEXT ATOM AND CAT
                      IC CAT IS DECOMPOSED INTO COMPONENTS K AND KK
                      I      COMMON /ULDAT2/ DUMMY(19),EOS
                      I      INTEGER EOS
                      I      COMMON /TOKEN/ATOM(63),CAT(63),A,MAXA,K,KK
                      I      INTEGER ATOM,CAT,A,K,KK
                      IC
                      I      K=EOS
                      I      A=A+1
                      I      IF(A.GT.MAXA) RETURN
                      IC
                      I      KK=CAT(A)/10
                      I      K=CAT(A)/100
                      I      IF(K.LT.0.OR.K.GT.EOS) GOTO 90
                      I      RETURN
                      IC ERROR IN CATEGORY
                      I   90 CAT(A)=KK=K=0
                      I      RETURN
                      I      END
                      I*****************************************************************
                      I
                      I
                      I
                      I      FUNCTION NEXTF(F)
                      IC TO OBTAIN THE NEXT CLASS IN FAMILY ORDER
                      I      COMMON/ULDAT2/TYP,NXT,BRO,SON,NPR,FAM,RES,MVAL,PNO,LIM,VALUE
                      I     2,STAT,TYP2,KNAME
                      I      COMMON /TEMPL/MAXM,MAXV,TNAME(100),TCELL(400)
                      I      INTEGER T,FIND,STATUS,P,S,SONNY,BROTH,STK(20)
                      IC
                      I      IF(F.GT.0) GOTO 100
                      IC FIRST VISIT
                      I      S=0
                      I      P=FIND(TNAME,MAXM,KNAME,STATUS)
                      I      IF(STATUS.EQ.1) GOTO 500
                      I      CALL ERROR(0,402)
                      I      RETURN
```

```
C
C [illegible]
  [illegible](P,SON)
      [illegible](P,BRO)
      [illegible].EQ.0) GOTO 200
      [illegible].EQ.0) GOTO 120
C [illegible]
      [illegible]
      [illegible].LE.20) GOTO 110
      CALL [illegible](P,1000)
      [illegible]
      [illegible]
C
  [illegible]
  [illegible]
      GOTO 100
C [illegible]
  [illegible].EQ.0) GOTO 220
C [illegible]
      [illegible]
      [illegible]
C [illegible]
  [illegible]) GOTO 230
[illegible]
C [illegible]
[illegible]
***************************************************************

      [illegible] REPEAT
C TO [illegible] OF ATOMS BETWEEN PARENTHESES
C
      COMMON /[illegible]2/ DUMMY(19)
     1[illegible],OPER,KNC,DESCON, NECL,ITNAME,CLNAME
     4,[illegible],EXEC,REQU,LPAREN,RPAREN
      INTEGER EOS,PPAREN
C
      COMMON /ICNEX/ATOM(63),CAT(63),A,MAXA,K,KK
      INTEGER ATOM,CAT,A,MAXA,K,KK
      INTEGER NR
C ----------------------------------------------
C
      LP=A+1
      NREP=ATOM(A-1)-1
      CAT(A-1)=0
      CAT(A)=0
```

```
I660   CALL NEXTA
I      IF(CAT(A).EQ.RPAREN) GOTO 680
I      IF(K.EQ.EOS) RETURN
I      GOTO 660
IC RIGHT EXPAND BRACKET
I680   RP=A-1
I      NSHELL=(RP-LP+1)*NREP-1
I      CALL EXPAND(NSHELL,KOND)
I      IF(KOND.EQ.0) RETURN
IC EXPAND NREP TIMES
I      A=RP
I      DO 690 NR=1,NREP
I      DO 690 L=LP,RP
I      IF(CAT(L).EQ.0) GOTO 690
I      A=A+1
I      ATOM(A)=ATOM(L)
I      CAT(A)=CAT(L)
I690   CONTINUE
I      A=LP
I      K=CAT(A)/100
I      KK=CAT(A)/10
I      RETURN
I      END
I***************************************************************
I
I
I
I      INTEGER FUNCTION T(MK,IPOS)
IC TO EXTRACT BYTE I OF CELL K OF TCELL
IC EACH CELL IS 2 WORDS LONG
I      COMMON /TEMPL/MAXN,MAXV,TNAME(109),TCELL(400)
I      INTEGER TCELL
IC
I      K=2*MK
I      I=IPOS
I      IF(I.LE.4) GOTO 10
IC 2ND WORD
I      K=K-1
I      IF(I.EQ.20) GOTO 20
I      I=I-4
IC RIGHT SHIFT N BYTES
I10    NBYTES= -8*(I-1)
I      T = SHIFT(TCELL(K),NBYTES).AND.255
I      RETURN
IC GET THE WHOLE 2ND WORD
I20    T=TCELL(K)
I      RETURN
I      END
I
I
I
I
I
I
```

```
      SUBROUTINE NXSCAN                                                 I  300 ATOM(A)=INCAR
C ULANG LEXICAL ANALYSER                                                I      CALL GETCHAR
C                                                                       I      IF(INCLAS.EQ.DIGIT.OR.INCLAS.EQ.DECIM) GOTO 310
      COMMON/LEXATT/LENALF,LENTOK,CHARC0,CHARCL(63),KL0,KLE,EOL         IC SUBTRACT CP
     1,QUICK                                                            I      CAT(A)=MINUSS
     2,ALPHA,SEPAR,DASH,DEPCP,DIGIT,DECIM,QUOTE,CCLON,TERMIN            I      LEN=1
     3,EQUI,LABELS,STRINGS,FNUMS,IACPS,SEPCP,PLUSS,MINUSS               I      CALL FADD
     INTEGER      LENALF,LENTOK,CHARC0,CHARCL,KL0,KLE,EOL,PXTCK         I      GO TO 20
     1,ALPHA,SEPAR,DASH,DEPCP,DIGIT,DECIM,QUOTE,CCLON,TERMIN            IC NEG NUMBER
     2,EQUI,LABELS,STRINGS,FNUMS,INUMS,SEPCP,PLUSS,MINUSS               I  310 ATOM(A)=0
C                                                                       I      SIGN=1
      COMMON/SCHEM/ATCH(63),CAT(63),A,MAXA,LEN,LIXTCK,INCAR,            I      GOTO 30
     1 INCLAS,DICLAS                                                    IC
      INTEGER ATOM,CAT,A,DICLAS,SCALE,SIGN,EXP                          IC  UNDEFINED
      DIMENSION ATOM(63)                                                I  400 INCLAS=0
      EQUIVALENCE (ATOM,FATOM)                                          I      A=A-1
C ------------------------------------------------------------          I      GO TO 10
C ENTRANCE = LABEL, ATOM(A) = KWD                                       IC
      DO 2 A=1,PXTCK                                                    IC  OPERATOR
    2 ATOM(A)=0                                                         I  500 ATOM(A)=INCAR
      CAT(1)=LABELS                                                     I      LEN=1
      ATOM(1)=1                                                         I      CALL FADD
      A=2                                                               IC     CALL CPOP
      LENGTH                                                            I      GOTO 10
      LIXTCK=LENTOK                                                     IC
C                                                                       IC  INTEGER
C START STATE                                                           I  600 ATOM(A)=10*ATOM(A) + (DICLAS-KL0)
   10 CALL SPECHAR                                                      I      CALL GETCHAR
      IF(INCLAS.EQ.DIGIT) GO TO 10                                      I      IF(INCLAS.EQ.DIGIT) GO TO 600
      CAT(A)=INUMS                                                      I      CAT(A)=INUMS
      IF(A.GT.PXTCK) GOTO 1010                                          I      SCALE=0
      SIGN=0                                                            I      IF(INCLAS.EQ.DECIM.OR.INCLAS.EQ.COLON) GO TO 30
   30 CAT(A)=INCLAS                                                     I      IF(SIGN.EQ.1) ATOM(A) = -ATOM(A)
C FREE ENTRIES OF CP                                                    I      IF(DICLAS.NE.KLE) GO TO 20
      GOTO(100,200,300,400,500,600,700,800,900,1000), INCLAS            IC EXPONENT E
C                                                                       I      FATOM(A)=ATOM(A)
C NAME                                                                  I      CAT(A)=FNUMS
  100 CALL SPELL                                                        I      GO TO 724
      CALL GETCHAR                                                      IC
      IF(INCLAS.NE.ALPHA.OR.INCLAS.EQ.SEPAR) GOTO 100                   IC  REAL NUMBER
      IF(INCLAS.EQ.QUOTE) GO TO 110                                     I  700 SCALE=0
      CALL ERRO                                                         I      FATOM(A)=ATOM(A)
C LOOKUP DICTIONARY                                                     I  710 CALL GETCHAR
      CALL AFTSCAN                                                      I      IF(INCLAS.NE.DIGIT) GO TO 720
      GOTO 20                                                           I      FATOM(A)=10.0*FATOM(A) + (DICLAS-KL0)
C                                                                       I      SCALE=SCALE-1
C  SEPARATOR                                                            I      GO TO 710
  800 ATOM(A)=INCAR                                                     I  720 IF(SIGN.EQ.1) FATOM(A)= -FATOM(A)
      CAT(A)=SEPOP                                                      I      IF(DICLAS.NE.KLE) GO TO 750
      LEN=1                                                             IC SCALE FACTOR  E
      CALL PACK                                                         I  724 SIGN=0
      GO TO 10                                                          I      CALL GETCHAR
C                                                                       I      IF(INCLAS.EQ.DASH) SIGN=1
C ERROR                                                                 I      IF(INCLAS.EQ.DASH.OR.DICLAS.EQ.PLUSS) CALL GETCHAR
```

```
      [illegible]
C EXPONENT PART
  [illegible] IF(INCLAS.NE.DIGIT) GO TO 740
      [illegible] + (INCLAS-KL0)
      CALL GETCHAR
      [illegible]
  740 IF(SIGN.EQ.1) EXP= -EXP
      SCALE=SCALE + EXP
C ADJUST EXPONENT
  750 IF(SCALE.EQ.0) GO TO 20
      [illegible]
      [illegible]
      IF(SCALE.LT.0) FAC= 0.1
      IF(SCALE.LT.0) SCALE= -SCALE
  760 IF(MOD(SCALE,2).EQ.1) R= R*FAC
      FAC = FAC*FAC
      SCALE= SCALE/2
      IF(SCALE.GT.0) GO TO 760
      [illegible] = [illegible]*R
      GO TO 20
C STRING
  [illegible] CALL GETCHAR
      [illegible]
      LEN=0
  810 IF(INCLAS.EQ.QUOTE) GO TO 830
      IF(INCLAS.EQ.EOSS) GOTO 840
      IF(LEN.LT.LENHALF) GO TO 820
C ATOM FULL
      [illegible] CAT(A)=CAT(A)+10
      LEN=0
      A=A+1
      IF(A.GT.MXTOK) GOTO 1010
      [illegible] + 10
      CAT(A) = STRINGS + NAT
C [illegible]
  820 LEN=LEN+1
      ATOM(A)=SHIFT(ATOM(A),6).OR.INCAR
      CAT(A)=CAT(A)+1
      CALL GETCHAR
      GO TO 810
C [illegible]
  830 CALL GETCHAR
      IF(INCLAS.NE.QUOTE) GO TO 840
C [illegible] QUOTE
      INCLAS=8
      GO TC 810
C END OF STRING
  840 IF(LEN.EQ.10) CAT(A)=CAT(A)-10
      CALL PACC
      GOTC 20
C
C A LABEL
  900 INCLAS=0

I       IF(ATOM(A).EQ.0) A=A-1
I       ATOM(1)=ATOM(A)
I       ATOM(A)=0
I       A=A-1
I       GO TO 10
IC
IC END OF SENTENCE
I 1000 A=A-1
I       MAXA=A
I       RETURN
IC ATOM STRING FULL
I 1010 CALL ERROR(A,1008)
I       GOTO 1000
I       END
I*****************************************************************
I
I
I
I       SUBROUTINE AFTSCAN
IC POSTSCAN PHASE, LOOKS UP DICTIONARY
I       COMMON /ULDAT2/ DUMMY(19)
I      3,EOS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWDCL,ITNAME,CLNAME
I       INTEGER EOS,STRING,FNUM,INUM,OPER,KWD,DESCON
IC
I       COMMON/TOKEN/ ATOM(63),CAT(63),A,MAXA
I       INTEGER ATOM,CAT,A
I       COMMON /DICT/ DNAME(63),DCAT(63),MAXD,D,DUMMYD(24)
I       INTEGER DNAME,DCAT,MAXD,D,STATUS,FIND,DICTCL
IC -------------------------------------------
IC
IC LOOKUP DICT
I       CAT(A)=0
I       D= FIND(DNAME,MAXD,ATOM(A),STATUS)
I       IF(STATUS.EQ.0) RETURN
IC FOUND, FOR SYNONYMS DCAT IS -VE
I       IF(DCAT(D).LT.0) D=IABS(DCAT(D))
I       DICTCL= DCAT(D)/100
I       IF(DICTCL.EQ.0) GO TO 80
I       IF(DICTCL.EQ.KWD.AND.CAT(2).EQ.0) GO TO 30
IC ELSE
I       CAT(A)=DCAT(D)
I       ATOM(A)=DNAME(D)
I       IF(DICTCL.NE.OPER) RETURN
IC CAN TEST FOR DOUBLE OPERATOR SEQUENCE HERE
I    20 CONTINUE
IC       CALL OPOP
I       RETURN
IC PUT KWD IN A=2
I    30 CAT(2)=DCAT(D)
I       ATOM(2)=DNAME(D)
I       IF(A.GT.2) GO TC 80
I       RETURN
IC BACKUP BY ONE ATOM
I    80 ATOM(A)=0
```

```
      A=A-1
      RETURN
      END
C*****************************************************************
C
      SUBROUTINE FRMOIN
C TO READ PRINT INPUT TOKENS AND CAT'S
C USES INBUF FOR HOLDING OUTPUT LINE
      COMMON/ULCAT1/LENALF,DUMMY(81),FNUMS,INUMS
      INTEGER LENALF,FNUMS,INUMS
C
      COMMON /INBUF/INBUF(80),MAXI,IPTR
      COMMON /TOKEN/ ATOM(63),CAT(63),A,MAXA
      INTEGER ATOM,CAT,A,TOKEN
C ---------------------------------------------
C
      MAXI=0
      DO 200 A=1,MAXA
      IF(CAT(A).EQ.FNUMS) GO TO 600
      IF(CAT(A).EQ.INUMS) GO TO 700
C EXTRACT CHARACTERS AND HOLD IN N1
C ASSUMING 6 BITS PER CHAR
      DO 123 K=1,LENALF
      NBITS=-6*(LENALF-K)
      N1=SHIFT(ATOM(A),NBITS).AND.63
      IF(N1.EQ.0) GO TO 130
      MAXI=MAXI+1
      INBUF(MAXI)=N1
  123 CONTINUE
      GO TO 130
C ALPHIC
  600 ENCODE(10,611,TOKEN) ATOM(A)
  611 FORMAT(I10)
      GO TO 210
C REAL
  700 ENCODE(10,711,TOKEN) ATOM(A)
  711 FORMAT(F10.3)
C
  720 DO 720 K=1,LENALF
      NBITS=-6*(LENALF-K)
      N1=SHIFT(TOKEN,NBITS).AND.63
      IF(N1.EQ.1R ) GO TO 720
      MAXI=MAXI+1
      INBUF(MAXI)=N1
  720 CONTINUE
C
C PUT SPACE BETWEEN WORDS,
  130 MAXI=MAXI+1
      INBUF(MAXI)=1R
  200 CONTINUE
C PRINT OUT
      PRINT 11,(INBUF(K),K=2,MAXI)
   11 FORMAT(1H0,5X,63R1)
      PRINT 13,(CAT(A),A=2,MAXA)
   13 FORMAT(1H ,5X,1E14)
      RETURN
      END
C*****************************************************************
C
C
C
      SUBROUTINE GETCHAR
C OBTAINS THE NEXT CHARACTER FROM INPUT BUFFER
C
      COMMON/ULCAT1/LENALF,LENTOK,CHARC0,CHARCL(63),KL0,KLE,EOL,MXTOK
     1,MXTOK
     2,ALPHA,SEPAR,DASH,DESCR,DIGIT,DECIM,QUOTE,COLON,TERMIN
     3,EOSS,LABELS,STRINGS,FNUMS,INUMS,SEPOP,PLUSS,MINUSS
      INTEGER LENALF,LENTOK,CHARC0,CHARCL,KL0,KLE,EOL,MXTOK
     2,ALPHA,SEPAR,DASH,DESCR,DIGIT,DECIM,QUOTE,COLON,TERMIN
     3,EOSS,LABELS,STRINGS,FNUMS,INUMS,SEPOP,PLUSS,MINUSS
C
      COMMON /INBUF/ INBUF(80),MAXI,IPTR
      COMMON/TOKEN/ATOM(63),CAT(63),A,MAXA,LEN,LIMTOK,
     1 INCAR,INCLAS,DICLAS
      INTEGER DICLAS
C ---------------------------------------------
C
      IPTR=IPTR+1
      IF(IPTR.GT.MAXI) GO TO 100
      INCAR=INBUF(IPTR)
      DICLAS=CHARCL(INCAR)
      INCLAS=DICLAS/100
      RETURN
C END OF LINE
  100 INCAR=EOL
      DICLAS=EOSS
      INCLAS=DICLAS/100
      RETURN
      END
C*****************************************************************
C
C
C
      SUBROUTINE SPELL
C TO REDUCE A NAME TO STD FORM
      COMMON/ULCAT1/LENALF,LENTOK,CHARC0,CHARCL(63),KL0,KLE,EOL,MXTOK
     1,MXTOK
     2,ALPHA,SEPAR,DASH,DESCR,DIGIT,DECIM,QUOTE,COLON,TERMIN
      INTEGER LENALF,LENTOK,CHARC0,CHARCL,KL0,KLE,EOL,MXTOK
     2,ALPHA,SEPAR,DASH,DESCR,DIGIT,DECIM,QUOTE,COLON,TERMIN
      COMMON/TOKEN/ATOM(63),CAT(63),A,MAXA,LEN,LIMTOK,INCAR
     1,INCLAS,DICLAS
      INTEGER ATOM,A
C ---------------------------------------------
C
```

```
      IF(ATOM(A).EQ.ALPHA) GO TO 100
      IF(INCLAS.EQ.SEPAR.OR.INCLAS.EQ.DASH) GO TO 200
      RETURN
C NAME
  100 IF(LEN.GE.LIPTOK) RETURN
      LEN=LEN+1
      ATOM(A)=SHIFT(ATOM(A),6).OR.INCAR
      RETURN
C NONFUNC KEPT SEPARATOR
  200 IF(LINTCK.GE.LENALF) RETURN
      LINTCK=LIPTOK+2
      RETURN
C-----------------------------------------------
C
C     ENTRY FORCE
C TO FORCE LENGTH UP TO LENALF WITH EOL
C
      IF(LEN.GE.LENALF) GO TO 20
      L1=LEN+1
      DO 10 L=L1,LENALF
   10 ATOM(A)=SHIFT(ATOM(A),6).OR.EOL
C REINITIALISE
   20 LEN=0
      LINTCK=LEXTOK
      RETURN
      END
```

```
      PROGRAM ULPREP(INPUT,OUTPUT,TAPE5=INPUT,TAPE1=512)
C ULANG PREPROCESSOR CONTROL
C-----
      CALL SECOND(SEK1)
      CALL TABLCR
      IEOF=0
C INPUT LOOP
   10 CALL INPUT(IEOF)
      IF(IEOF.EC.1) GO TO 100
      CALL ULSCAN
      CALL LLKNE(NAME)
C ---------
C CALLS TO USER SUBROUTINES TO BE INSERTED HERE
      IF(NAME.EC.4LCALC) CALL CALC
      IF(NAME.EC.4LDISP) CALL DISPLAY
C ---------
      GOTO 10
C END OF REQUESTS
  100 CALL SECOND(SEK2)
      SEK=SEK2-SEK1
      PRINT 101, SEK
  101 FORMAT(1H0,40X,F8.3,5H SEC.)
      CALL TABCLO
      STOP
      END
C*****************************************************************
C
C
C     SUBROUTINE ACX
C TO PARSE EXPRESSIONS
C       CALLED BY USERCOM
C F,G ARE OPERATOR PRECEDENCE FUNCTIONS
C S,R ARE INDICES OF STACK SYMBOLS IN F,G
C A IS INDEX IN F,G OF INCOMING SYMBOL.
C I IS STACK POINTER
C J IS TAIL OF PRIME PHRASE PTR IN STACK
C JC IS HEAD OF PRIME PHRASE PTR IN STACK
C
      COMMON /ULOAT2/ DUMMY(19)
     3,EOS,STRING,FNUM,INUM,OPER,KWO,DESCON, KWCCL,ITNAME,CLNAME
     4,EXNAME,CANE,EXEC,REQU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TC,CF
      INTEGER
     1 EOS,STRING,FNUM,INUM,OPER,KWO,DESCON,KWCCL,ITNAME,CLNAME
     2,EXNAME,CNAME,EXEC,REQU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TC,CF
C
      COMMON /TCKEN/ATOM(63),CAT(63),A,MAXA,K,KK
      INTEGER ATOM,CAT,A,MAXA,K,KK
      INTEGER STK(20),STKA(20),F(12),G(12),C,R,S,VT,VN,SA,XTYP
      INTEGER THEN,DO
      DATA THEN,DO/463,464/
C DEFINITIONS FOR PRECEDENCE FUNCTIONS
      DATA LEFPAR,MARKER,VT,VN/9,10,11,12/
      DATA F/ 5, 7, 7, 8,11,13,13,15, 3, 2,16, 1/
      DATA G/ 4, 6, 8, 8,10,12,14, 3,14, 0,16, 1/
C-----------------------------------------------
C
C ADD EXPRESSION NAME TO SPECIAL-NAMES
      XTYP=2
      M=0
      MVALUE=0
      CALL INSERP(ATOM(A-1),MVALUE,CAT(A-1),M,DUM)
C
C INITIALISE
    1 STK(1)=MARKER
      I=1
C GET NEXT ATOM
    8 CALL NEXTA
      IF(K.EQ.0) GOTO 90
      GOTO(10,90,90,40,50,60,60,60,90,90),K
C NAME
   10 IF(KK.NE.ITNAME.AND.KK.NE.EXNAME) GOTO 90
      IF(R.EQ.VT) GOTO 90
      IT=0
      CALL INSERP(ATOM(A),XTYP,KK,IT,DUM)
      ATOM(A)=IT
      GOTO 60
C DESCRIPTOR, MAYBE THEN, DO
   40 IF(CAT(A).EQ.THEN.OR.CAT(A).EC.DO) CALL NEXTA
      GOTO 90
C OPERATOR
   50 R=KK-50
      GOTO 110
C NAME OR VALUE
   60 IF(R.EQ.VT) GOTO 90
      R=VT
      GOTO 110
C END OF EXPRESSION, INSERT MARKER
   90 R=MARKER
C-----------------------------------------------
C LOOK FOR TAIL OF PRIME-PHRASE IN STACK
C
  110 J=I
      IF(STK(I).EQ.VN) J=I-1
  200 S=STK(J)
      IF(F(S).GT.G(R)) GOTO 300
C STACK THIS ATOM
      I=I+1
      IF(I.LE.20) GOTO 204
      CALL ERROR(I,1013)
      GOTO 8
  204 STK(I)=R
      STKA(I)=A
      GOTO 8
C LOOK FOR HEAD OF PRIME-PHRASE IN STACK
  300 Q=STK(J)
      JC=J
      J=J-1
```

```
      IF(STK(J).EQ.VN) J=J-1
      S=STK(J)
      IF(F(S).GE.G(O)) GOTO 300
C REDUCE PRIME-PHRASE IN STACK FROM (J+1) TO I
      IF(C.EQ.LPPAR) GOTO 500
C FORM POLISH STRING IN PCELL
      SA=STKA(JC)
      CALL APPENP(P,ATOM(SA),CAT(SA))
C ADJUST STACK POINTER
  900 I=J+1
      STK(I)=VN
      IF(I.GT.2.OR.P.NE.MARKER) GOTO 200
      RETURN
C
C ---------------------------------------
C
      ENTRY REX
C TO PARSE BOOLEAN EXPRESSIONS ... IF CONDITION THEN ...
C       CALLED BY USERCOM
      XTYP=4
      M=0
      MVALUE=0
      MCAT=EXNAME*10+1
C ADD CONDITION-NAME TO SPECIAL NAMES
      CALL INSERP(CLULCCNO,MVALUE,MCAT,M,DUM)
      GOTO 1
      END
*****************************************************************

      SUBROUTINE APPENP(C,PVALUE,PCAT)
C TO APPEND ANOTHER VALUE TO PARAM = C
C   CALLED BY AEX,INSERP,LIM,QUAL,SEMIMPL,USERCOM,USERSEM
C       MEANING OF ARGUMENTS
C   C   IN. INDEX OF PARAM IN IN INAME OR SPNAME
C           C MUST BE GT 0
C PVALUE IN. PARAM VALUE
C       OUT. PARAM INDEX IN PCELL
C PCAT   IN. PARAM TYPE
C
      COMMON/ULCAT2/TYP,NXT,BRO,SON,NPR,FAM,RES,MVAL,CV,LIM,VALUE
     2,STAT,TYP2, KNAME, MXDICT,MXSPNAM,MXTNAM,MXTCEL,MXPCEL
      INTEGER C,PVALUE,PCAT,P
      COMMON/CTSTEP/ MAXP,SPNAME(10),PCELL(400)
C
      IF(C.GT.0) GOTO 10
      CALL ERROR(PVALUE,401)
      RETURN
C FIND LAST VALUE
   10 NXV=C
      IF(F(C,TYP).EQ.0) GOTO 70
   20 NPTR=F(NXV,NXT)
      IF(NPTR.EC.0) GOTO 50
      NXV=NPTR
      GOTO 20
C ADD NEXT VALUE
   50 MAXP=MAXP+1
      IF(MAXP.LE.MXPCEL) GOTO 60
      CALL ERROR(MAXP,1012)
      RETURN
   60 CALL FSET(NXV,NXT,MAXP)
      NXV=MAXP
C
170   K1=PCAT/100
      K2=MOD(PCAT,100)
      CALL FSET(NXV,TYP,K1)
      IF(K2.GT.0) CALL FSET(NXV,TYP2,K2)
      CALL FSET(NXV,VALUE,PVALUE)
      CALL FSET(NXV,CV,C)
C RETURN VALUE-PTR IN PVALUE
      PVALUE=NXV
      RETURN
      END
*****************************************************************


      SUBROUTINE DUMPP
C TO DUMP COMMAND TABLE CONTENTS
C       CALLED BY ULKWD
C
      COMMON/ULCAT2/TYP,NXT,BRO,SON,NPR,FAM,RES,MVAL,CV,LIM,VALUE
     2,STAT,TYP2, KNAME, MXDICT,MXSPNAM,MXTNAM,MXTCEL,MXPCEL
     3,EOS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWDCL,ITNAME,CLNAME
C
      INTEGER
     1 EOS,STRING,FNUM,INUM,OPER,KWD,DESCON,KWDCL,ITNAME,CLNAME
      INTEGER BYTE(5),I,P,PNAME,TNAME,SPNAME,PVALUE,DTYPE
      COMMON /TEMPL/MAXM,MAXV,TNAME(100)
      COMMON/CTSTEP/ MAXP,SPNAME(10)
C
      PRINT 11
   11 FORMAT (1H )
      MAXMSP=MAXM+MXSPNAM
      DO 100 I=1,MAXMSP
      IF(F(I,STAT).EQ.0) GOTO 100
      IF(F(I,TYP).GE.10) GOTO 100
      ITYPE=T(I,TYP)
      IF(F(I,TYP).EQ.0.AND.(ITYPE.EC.DESCON.OR.ITYPE.EQ.KWD))
     1 GOTO 100
C ACTIVE PARAM
      IV=I
      ISP= I-MAXM
      IF(I.LE.MAXM) PNAME=TNAME(I)
      IF(I.GT.MAXM) PNAME=SPNAME(ISP)
C PRINT LOOP
   10 DO 14 IB=1,5
```

```
   14 BYTE(IB)=F(IV,IB)
      PVALUE=F(IV,VALUE)
      OTYPE=1H
      IF(BYTE(1).EC.DESCON.OR.BYTE(1).EC.KWD) GOTO 16
      IF(BYTE(1).EC.FNUM) GOTO 24
      IF(BYTE(1).EQ.STRING) GOTO 26
      GOTO 22
C FOR DESC CONS OBTAIN NAME
   16 IT=I
   18 IT=T(IT,NXT)
      IF(IT.EC.0) GOTO 22
      IF(PVALUE.NE.T(IT,MVAL)) GOTO 18
      OTYPE=TNAME(IT)
C
   22 PRINT 21,IV,PNAME,BYTE,PVALUE,OTYPE
   21 FORMAT(1X,I4,1X,A8,5I4,1X,I10,1X,A8)
      GOTO 30
   24 PRINT 23,IV,PNAME,BYTE,PVALUE
   23 FORMAT(1X,I4,1X,A8,5I4,1X,F10.3)
      GOTO 30
   26 PRINT 25,IV,PNAME,BYTE,PVALUE
   25 FORMAT(1X,I4,1X,A8,5I4,1X,A10)
C
   30 IF(BYTE(2).EC.0) GOTO 100
      IV=BYTE(2)
      PNAME=1H
      GOTO 18
  100 CONTINUE
      RETURN
      END
*****************************************************************

      SUBROUTINE ECHOUT
* TO ECHO PRINT PARAMETERS OF A COMMAND
*   USER-DEFINED AND DEFAULT VALUES
*
      COMMON/LLCAT2/TYP,NXT,PRO,SCR,NPR,FAM,RES,MVAL,CV,LIM,VALUE
     2,STAT,TYP2, KNAME, MXDICT,MXSPNAM,MXTNAM,MXTCEL,MXPCEL
     3,EOS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWDCL,ITNAME,CLNAME
      INTEGER
     1 EOS,STRING,FNUM,INUM,OPER,KWD,DESCON,KWDCL,ITNAME,CLNAME
C
      COMMON /TEMPL/MAXM,MAXV,TNAME(109)
      COMMON/CTSTEP/ MAXP,SPNAME(10)
      INTEGER MCAT(9),F,P,T,PTYPE,TNAME,SPNAME
      DATA MCAT/1HV,1HC,1HK,1HC,1HO,1HI,1HE,1HA,1H*/
*
* ---------------------------------------------------
*
      PRINT 11
   11 FORMAT(1H0,6X,17HACTIVE PARAMETERS/
     .1H ,20HNAME   TYPE  VALUE    MEANING/)
I       F=0
I       F=NEXTF(F)
I    10 IF(P(F,STAT).EQ.0) GOTO 200
I* ACTIVE CLASS
I       PTYPE=P(F,TYP)
I       M=F
I       IF(PTYPE.GE.10) GOTO 100
I*
I* DESC CONS OR KWD
I       NAME=TNAME(F)
I       KTYPE=1H
I       IF(PTYPE.GT.0) KTYPE=MCAT(PTYPE)
I       K=M
I*
I    20 KVALUE=P(K,VALUE)
I* FIND MEANING OF VALUE
I       IT=F
I    30 IT=T(IT,NXT)
I       IF(IT.EQ.0) GOTO 32
I       IF(KVALUE.NE.T(IT,MVAL)) GOTO 30
I       KMEANS=TNAME(IT)
I    32 PRINT 13,NAME,KTYPE,KVALUE,KMEANS
I    13 FORMAT(1X,A8,1X,A2,I10,1X,A8)
I       K=P(K,NXT)
I       IF(K.EQ.0) GOTO 200
I       NAME=1H
I       GOTO 20
I*
I* ITEM-VALUE
I   100 M=T(M,NXT)
I       IF(M.EQ.0) GOTO 200
I       IF(P(M,STAT).EQ.0) GOTO 100
I       NAME=TNAME(M)
I       K=M
I*
I   120 PTYPE=P(K,TYP)
I       KTYPE=1H
I       IF(PTYPE.GT.0) KTYPE=MCAT(PTYPE)
I       KVALUE=P(K,VALUE)
I       IF(PTYPE.EQ.FNUM) GOTO 104
I       IF(PTYPE.EQ.STRING) GOTO 106
I       IF(PTYPE.EQ.4) GOTO 110
I*
I       PRINT 13,NAME,KTYPE,KVALUE
I       GOTO 130
I   104 PRINT 15,NAME,KTYPE,KVALUE
I    15 FORMAT(1X,A8,1X,A2,G10.3)
I       GO TO 130
I   106 PRINT 17,NAME,KTYPE,KVALUE
I    17 FORMAT(1X,A8,1X,A2,A10)
I       GOTO 130
I* DESCRIPTOR
I   110 K2=P(K,TYP2)
I       KVALUE=1H
```

```
      IF(K2.EC.31) KVALUE=4HFROM
      IF(K2.EC.32) KVALUE=2HTO
      IF(K2.EC.33) KVALUE=2HBY
      PRINT 17,NAME,KTYPE,KVALUE
*
  130 K=R(K,NXT)
      IF(K.EQ.0) GOTO 180
      NAME=1H
      GOTO 120
*
* NEXT CLASS
  200 F=NEXTF(F)
      IF(F.GT.0) GOTO 10
*
* PRINT SPECIAL-NAME SECTION
*
      DO 400 I=1,MXSPNAM
      IF(SPNAME(I).EQ.0) RETURN
      NAME=SPNAME(I)
      KMEANS=I
*
  220 KTYPE=F(K,TYP)
      KTYPE=1H
      IF(ETYPE.GT.0) KTYPE=HCAT(ETYPE)
      KVALUE=F(K,VALUE)
      KMEANS=1H
      IF(KTYPE.EQ.1) GOTO 220
      IF(KTYPE.EQ.INUM) GOTO 260
      IF(KTYPE.EQ.FNUM) GOTO 270
      PRINT 17,NAME,KTYPE,KVALUE
      GOTO 300
  220 IF(S(K,TYP2).EQ.10) KMEANS=TNAME(KVALUE)
      PRINT 13,NAME,KTYPE,KVALUE,KMEANS
      GOTO 300
  260 PRINT 17,NAME,KTYPE,KVALUE
      GOTO 300
  270 PRINT 15,NAME,KTYPE,KVALUE
*
  300 K=R(K,NXT)
      IF(K.EC.0) GOTO 400
      NAME=1H
      GOTO 210
  400 CONTINUE
      RETURN
      END
*****************************************************************

      SUBROUTINE ERRORP(IT,K,NERR)
C TO SIGNAL USER PARAMETER ERRORS
C       CALLED BY PROUNDS, USERSEM
      COMMON /TEMPL/MAXM,MAXV,TNAME(109)
C
      IF(NERR.EC.401) GOTO 401
      IF(NERR.EQ.402) GOTO 402
      IF(NERR.EQ.404) GOTO 404
      RETURN
  401 PRINT 1401,TNAME(IT)
 1401 FORMAT(5X,A8,* USED IN WRONG CONTEXT*)
      RETURN
  402 PRINT 1402,TNAME(IT),K
 1402 FORMAT(5X,A8,* NOT ENOUGH PARAMETERS *,I4,* REQU*)
      RETURN
  404 PRINT 1404,TNAME(IT)
 1404 FORMAT(5X,A8,* INPUT VALUES OUT OF RANGE*)
      RETURN
      END
*****************************************************************


      SUBROUTINE INSERP(PNAME,PVALUE,PCAT,C,F)
C TO INSERT FIRST VALUE OF A MEMBER IN PCELL
C   CALLED BY AEX,QUAL,SEMCLA,SEMIMPL,USERCOM,USERSEM
C
C         MEANING OF ARGUMENTS
C PNAME   IN, PARAM NAME
C PVALUE IN, PARAM VALUE
C            IF PARAM USED IN AEX PVALUE=2
C            IF PARAM USED IN BEX PVALUE=4
C       ' OUT, PCELL INDEX OF VALUE (IF APPENP CALLED)
C PCAT    IN, PARAM TYPE
C             PCAT=0 MEANS NO VALUE (APPENP NOT CALLED)
C   C     IN, OUT PARAM INDEX IN TNAME OR SPNAME
C             IF C=0 THEN FIND, ELSE IF C GT 0 THEN IN
C   F     OUT, PARAM CLASS INDEX
C
      COMMON/ULCAT2/TYP,NXT,BRO,SCN,NPR,FAM,RES,MVAL,CV,LIM,VALUE
     2,STAT,TYP2, KNAME, MXCICT,MXSPNAM,MXTNAM,MXTCEL,MXPCEL
     3,ECS,STRING,FNUM,INUM,OPER,KWC,DESCON, KWUCL,TTNAME,CLNAME
     4,EXNAME,ONAME,EXEC,RECU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TO,OF
     5,RESTYP,ULTYP,LLTYP, EOV,EOVCAT,LIMIT,KOND
      INTEGER
     1 EOS,STRING,FNUM,INUM,OPER,KWC,DESCON,KWUCL,TTNAME,CLNAME
C
      INTEGER PNAME,PVALUE,PCAT,C,FIND,STATUS,T,P,F,PSTAT,TTYP
      INTEGER SPNAME,CSTAT,FTYP
      COMMON /TEMPL/MAXM,MAXV,TNAME(109)
      COMMON/CTSTEP/ MAXP,SPNAME(10)
C ---------------------------------------------
C
C PARAM NAME
      IF(C.GT.0) GOTO 10
      C=FIND(TNAME,MAXM,PNAME,STATUS)
      IF(STATUS.EQ.1) GOTO 10
C NAME NOT FOUND IN T ,CHECK SPECIAL NAMES
      F=0
```

```
      DO 6 IC=1,MXSPNAM                                  |    80 IF(PVALUE.EQ.2) CSTAT=CSTAT.AND.5.OR.2
      C=MAXP+IC                                          |       IF(PVALUE.EQ.4) CSTAT=CSTAT.AND.3.OR.4
      IF(SPNAME(IC).EQ.0) GOTO 6                         |       CALL FSET(C,STAT,CSTAT)
      IF(SPNAME(IC).NE.PNAME) GOTO 4                     |       IF(F.EQ.0) RETURN
      CSTAT=P(C,STAT)                                    |       FTYP=T(F,TYP)
      IF(PCAT.EQ.EXNAME) GOTO 88                         |       CALL FSET(F,TYP,FTYP)
      GOTO 98                                            |       RETURN
    4 CONTINUE                                           |       END
      C=0                                                |*****************************************************************
      CALL ERROR(IC,1811)                                |
      RETURN                                             |
C ADD KEYWORD IN SPECIAL NAMES                           |
    6 SPNAME(IC)= PNAME                                  |       SUBROUTINE LIM(M)
      CALL FSET(C,STAT,1)                                |IC TO PARSE VALUE RANGES ... FROM X TO Y BY Z
      IF(PCAT.GT.0) CALL APPENP(C,PVALUE,PCAT)           |IC        CALLED BY USERCOM
      RETURN                                             |IC
C                                                        |       COMMON /ULDAT2/ DUMMY(19)
C RENAME NAME EXISTS (ASSUME IT IS NOT A CLASS-NAME)     |      3,EOS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWDCL,ITNAME,CLNAME
C    (NO CHECKS ARE MADE FOR CLASS-NAMES)                |      4,EXNAME,ONAME,EXEC,RECU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TO,OF
   18 TTYP=T(C,TYP)                                      |       INTEGER
      F=T(C,FNM)                                         |      1 EOS,STRING,FNUM,INUM,OPER,KWD,DESCON,KWDCL,ITNAME,CLNAME
      CSTAT=P(C,STAT)                                    |      2,EXNAME,ONAME,EXEC,RECU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TO,OF
      IF(CSTAT.GT.0) GOTO 14                             |IC
C ACCUMULATE AC OF ACTIVE PRS                            |       COMMON /ICKEN/ATOM(63),CAT(63),A,MAXA,K,KK
      IF(F.LE.0) RETURN                                  |       INTEGER ATOM,CAT,A,MAXA,K,KK
      PSTAT=P(F,STAT)+1                                  |IC
      CALL PSET(F,STAT,PSTAT)                            |IC LIMIT MARKER
   14 IF(PCAT.EQ.ITNAME) GOTO 80                         |       CALL APPENP(M,0,CAT(A))
C 8K INPUT PARAMETER                                     |       CALL NEXTA
      IF(MOD(CSTAT,2).EQ.1) GOTO 50                      |       IF(K.EQ.INUM.OR.K.EC.FNUM) GO TO 100
      CSTAT=CSTAT.OR.1                                   |       IF(K.EQ.DESCON.OR.K.EC.STRING) GOTO 100
      CALL PSET(C,STAT,CSTAT)                            |       CALL ERROR(A,305)
      IF(TTYP.EC.DESCON.OR.TTYP.EQ.KWD) GOTO 38          |       RETURN
C VALUE                                                  |IC LIMIT VALUE
      FTYP=T(F,TYP)                                      |  100 CALL APPENP(M,ATOM(A),CAT(A))
      CALL FSET(F,TYP,FTYP)                              |       KKS=KK
      IF(PCAT.GT.0) CALL APPENP(C,PVALUE,PCAT)           |       CALL NEXTA
      RETURN                                             |       IF(KK.EQ.KKS+1) GOTO 100
C DESCRIPTION OR KWD                                     |       RETURN
38    PVALUE=T(C,MVAL)                                   |       END
      CALL APPENP(F,PVALUE,PCAT)                         |*****************************************************************
      RETURN                                             |
C                                                        |
C NAME ALREADY EXISTS                                    |
   50 IVALUE=PVALUE                                      |       INTEGER FUNCTION P(K,I)
      IF(TTYP.EC.DESCON) C=F                             |IC TO EXTRACT FIELD I OF CELL 2*K OR WORD 2*K-1
      CALL APPENP(C,EQV,EQVCAT)                          |IC        CALLED BY MOST ROUTINES OF ULPREP
      IF(PCAT.EC.0) RETURN                               |       COMMON/CISIEP/ MAXP,SPNAME(10),PCELL(400)
      CALL APPENP(C,IVALUE,PCAT)                         |       INTEGER PCELL,MASKG(5),NBITS(5)
      PVALUE=IVALUE                                      |       DATA MASKG/15,255,255,31,127/
      RETURN                                             |       DATA NBITS/0,-4,-12,-20,-25/
C                                                        |IC
C ITEM IS PART OF EXPRESSION OR CONDITION                |       IF(I.EQ.20) GOTO 20
C SET STATUS TO 2 OR TO 4                                |IC RIGHTSHIFT
```

```
      P=SHIFT(PCELL(2*K),NBITS(I)).AND.MASKG(I)
      RETURN
C GET THE 1ST WORD
   20 P=PCELL(2*K-1)
      RETURN
      END
C*****************************************************************

      SUBROUTINE PBOUNDS(M,V)
C TO CHECK LIMITS OF ALL USER VALUES OF A PARAM M
C AND TO CHECK PDEFS
C       CALLED BY USERSEM
C
      COMMON/LLCAT2/TYP,NXT,PRC,SOA,APR,PAR,RES,PVAL,CV,LIM,VALLE
     2,STAT,TYP2, KNAME, MXDICT,MXSPNAM,MXTNAM,MXTCEL,MXPCEL
     3,ECS,STRING,FNUM,INUM,OPER,KWC,DESCOA, KWCCL,ITNAME,CLNAME
     4,DUMMY(12),ULTYP,LLTYP,EOV,EOVCAT,LIMIT
      INTEGER*2
     1 ECS,STRING,FNUM,INUM,OPER,KWC,DESCOA,KWCCL,ITNAME,CLNAME
     2,ULTYP,LLTYP,EOV,EOVCAT
      INTEGER*2 VL,T,P,PVALUE,PTYP,V
      EQUIVALENCE (PVALUE,FPVAL),(LVALUE,FLVAL)
C ---------------------------------------------
C
      MTYP=T(P,TYP)
      MV=P
C LOOK AT EACH PARAM VALUE
   10 PVALLE=F(MV,VALUE)
      PTYP=P(PV,TYP)
C CHECK MODES
      IF(PTYP.EQ.MTYP) GOTO 40
      IF(PTYP.EQ.INUM.AND.MTYP.EQ.FNUM) GOTO 20
      IF(PTYP.EQ.FNUM.AND.MTYP.EQ.INUM) GOTO 30
      GOTO 200
C CONVERT TO REAL
   20 FPVAL=PVALUE
      CALL PSET(MV,TYP,MTYP)
      CALL FSET(MV,VALUE,PVALUE)
      GOTO 40
C CONVERT TO INTEGER - TRUNCATE
   30 PVALLE=FPVAL
      CALL FSET(MV,TYP,MTYP)
      CALL FSET(MV,VALUE,PVALUE)
C
C CHECK LIMITS
C
   40 IF(V.EQ.0) GOTO 200
      VL=V
   50 VL=T(VL,LIM)
      IF(VL.EQ.0) GOTO 200
      LVALLE=T(VL,VALUE)
      IF(MTYP.EQ.FNUM) GOTO 70
      IF(T(VL,RES).EQ.ULTYP) GOTO 60
C INTEGER OR ALPHA LL
      IF(PVALUE.GE.LVALUE) GOTO 50
      GOTO 90
C INTEGER OR ALPHA UL
   60 IF(PVALUE.LE.LVALUE) GOTO 50
      GOTO 90
C
   70 IF(T(VL,RES).EQ.ULTYP) GOTO 80
C REAL LL
      IF(FPVAL.GE.FLVAL) GOTO 50
      GOTO 90
C REAL UL
   80 IF(FPVAL.LE.FLVAL) GOTO 50
C
C SET PVALUE TO LIMIT
   90 CALL ERRORP(M,PVALUE,404)
      CALL FSET(MV,VALUE,LVALUE)
C
C DO CHECKING FOR ALL INPUT VALUES
  200 MV=F(PV,NXT)
      IF(MV.GT.0) GOTO 10
      RETURN
      END
C*****************************************************************
C
C
C
      SUBROUTINE PSET(K,I,ITEM)
C TO INSERT ITEM IN FIELD I OF CELL 2*K OR IN WORD 2*K-1
C       CALLED BY APPENP, INSERP, PBOUNDS
      COMMON/CTSTEP/ MAXP,SPNAME(10),PCELL(400)
      INTEGER PCELL,MASKS(5),NBITS(5)
      DATA MASKS/37777777608,37777770017B,377740077778
     *,37603777777B,1777777778/
      DATA NBITS/0,4,12,20,25/
C
      IF(I.EQ.20) GOTO 20
C LEFTSHIFT
      KK=2*K
      PCELL(KK)=PCELL(KK).AND.MASKS(I).OR.SHIFT(ITEM,NBITS(I))
      RETURN
C PUT THE 1ST WORD
   20 PCELL(2*K-1)=ITEM
      RETURN
      END
C*****************************************************************
C
C
C
      SUBROUTINE QUAL(M,F)
C TO PROCESS QUALIFIERS  OF, WITH
C       CALLED BY USERCOM
C
```

```
      COMMON /LLDAT2/ DUMMY(19)
     3,EOS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWCCL,ITNAME,CLNAME
     4,EXNAME,CNAME,FXEC,REQU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TC,CF
     5,RESTYP,ULTYP,LLTYP, EOV,EOVCAT,LIMIT,KCNG
      INTEGER
     1 EOS,STRING,FNUM,INUM,OPER,KWD,DESCON,KWCCL,ITNAME,CONAME
     2,EXNAME,CNAME,EXEC,REQU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TC,CF
     3,RESTYP,ULTYP,LLTYP, EOV,EOVCAT
C
      COMMON /TOKEN/ATOM(63),CAT(63),A,MAXA,K,KK
      INTEGER ATOM,CAT,A,MAXA,K,KK
      INTEGER PMA,PMC,PMOFF
C    CLALIFIER CONSTANT DEFINITIONS ONLY
      INTEGER POSMARK,THIS,NEXT,PREV,DIRMARK,AGC
      DATA POSMARK,THIS,NEXT,PREV,DIRMARK,AGC
     1/41,411,412,413,42,422/
C ---------------------------------------------
C
C CF ...
      CALL NEXTA
      IF(KK.EC.ITNAME) GOTO 200
      IF(KK.EC.POSMARK) GOTO 500
      IF(K.EC.INUM) GOTO 600
      RETURN
C CF C ...
  200 MC=0
      CALL INSERP(ATOM(A),PVALUE,0,MC,F)
      CAT(A)=CNAME*10
      CALL NEXTA
      IF(CAT(A).EQ.EQUAL) CALL NEXTA
      IF(KK.EC.POSMARK) GOTO 300
      IF(K.EC.INUM.OR.K.EQ.FNUM) GOTO 210
      IF(K.EC.DESCON.OR.K.EQ.STRING) GOTO 210
      RETURN
C CF C = A
210   CALL APPENP(MC,ATOM(A),CAT(A))
      KKS=KK
      CALL NEXTA
      IF(KK.EC.KKS+1) GOTO 210
      CALL APPENP(M,MC,OF)
      RETURN
C ---------------------------------------------
C CF C THIS ...
C        PMC IS POSMARK CATEGORY
C        PMOFF IS POSMARK OFFSET
300   PMC=CAT(A)
      PMOFF=0
      CALL NEXTA
      IF(KK.EC.AOP) GOTO 400
      IF(K.EC.INUM) GOTO 310
      GOTO 1000
C CF C THIS =A
310   PMOFF=ATOM(A)
      CALL NEXTA
I       GOTO 1000
IC CF C THIS AOP ...
I400    CALL NEXTA
I       IF(K.EQ.INUM) GOTO 420
I       GOTO 1000
IC CF C THIS AOP M
I420    IF(CAT(A-1).EQ.MINUS) ATOM(A)= -ATOM(A)
I       PMOFF=ATOM(A)
I       CALL NEXTA
I       GOTO 1000
IC
IC ---------------------------------------------
IC 2ND FORMAT    OF THIS ...
I500    PMC=CAT(A)
I       PMOFF=0
I       CALL NEXTA
I       IF(KK.EQ.ITNAME) GOTO 510
I       CAT(A-2)=0
I       CAT(A-1)=0
I       RETURN
IC OF THIS C
I510    MC=0
I       CALL INSERP(ATOM(A),PVALUE,0,MC,F)
I       CAT(A)=CNAME*10
I       CALL NEXTA
I       GOTO 1000
IC ---------------------------------------------
IC  3RD FORMAT  OF M ...
I600    PMOFF=ATOM(A)
I       CALL NEXTA
I       IF(KK.EQ.ITNAME) GOTO 700
I       CAT(A-2)=0
I       RETURN
IC CF M C
I700    MC=0
I       CALL INSERP(ATOM(A),PVALUE,0,MC,F)
I       CAT(A)=CNAME*10
I       CALL NEXTA
I       IF(KK.EQ.DIRMARK) GOTO 710
I       CAT(A-3)=0
I       A=A-2
I       RETURN
IC OF M C ACC
I710    IF(CAT(A).EQ.AGC) PMOFF= -PMOFF
I       CALL NEXTA
I       GOTO 1010
IC ADJUST OFFSET
I1000   IF(PMC.EC.THIS) GOTO 1010
I       IF(PMC.EQ.NEXT) PMOFF=PMOFF+1
I       IF(PMC.EC.PREV) PMOFF=PMOFF-1
IC FORM LINKS
I1010   PMA=4LTHIS
I       CPM=0
I1020   CALL INSERP(PMA,0,THIS,CPM,CUM)
```

```
      CALL APPENP(MQ,CP%,THIS)
      CALL APPENP(MQ,PNOFF,INUM)
      CALL APPENP(P,MC,OP)
      RETURN
      END
*****************************************************************

      SUBROUTINE SEMCLA(F)
C TO EXPAND CLASS NAME
C     <  CALLED BY USERCOM
C
      COMMON/LLCAT2/TYP,NXT,BRO,SON,NPR,FAM,RES,MVAL,CV,LIM,VALLE
      COMMON /TEMPL/MAXM,MAXV,TNAME(109),TCELL(400)
      COMMON /TCKEN/ATOM(63),CAT(63),A,MAXA,K,KK
      INTEGER ATOM,CAT,A,MAXA,K,KK
      INTEGER F,STATUS,FIND,T,TNAME
C
      FMFIND(TNAME,MAXM,ATOM(A),STATUS)
      IF(STATUS.EQ.1) GOTO 1218
      CALL ERROR(A,306)
      RETURN
C
1218  M=T(F,NXT)
      CAT(A)=8
      A=A-1
C EXPAND LOOP
1220  IF(P.EQ.0) RETURN
      CALL EXPAND(1,KCMD)
      IF(MCHC.EC.8) RETURN
      A=A+1
      ATOM(A)=TNAMF(M)
      CAT(A)=T(P,TYP)*10
C INSERT ITEM NAME IN P
      CALL INSERP(ATOM(A),PVALUE,0,P,F)
      M= T(M,NXT)
      GOTO 1220
      END
*****************************************************************

      SUBROUTINE SEMIMPL(M,F)
C FINDS IMPLICIT ITEM NAMES
C       CALLED BY USERCOM
C
      COMMON/LLCAT2/TYP,NXT,BRO,SON,NPR,FAM,RES,MVAL,CV,LIM,VALLE
     2,STAT,TYP2, KNAME, MXDICT,MXSPNAM,MXTNAP,MXTCEL,MXPCEL
     3,EOS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWCCL,ITNAME,CLNAME
     4,BUFFY(16),LIMIT
      INTEGER
     1 EOS,STRING,FNUM,INUM,OPER,KWD,DESCON,KWCCL,ITNAME,CLNAME
C
      COMMON /TCKEN/ATOM(63),CAT(63),A,MAXA,K,KK
      INTEGER ATOM,CAT,A,MAXA,K,KK
      COMMON /TEMPL/MAXM,MAXV,TNAME(109),TCELL(400)
      INTEGER F,C,P,TTYP,TNAME,FNAME,FBEG,FIND,STATUS,T
C -------------------------------------------------
C
      FBEG=F
      IF(M.GT.0) GOTO 110
C LOOK AT NEXT CLASS
100   F=NEXTF(F)
      IF(F.EQ.FBEG) GOTO 104
      IF(F.EQ.0) GOTO 100
      M=T(F,NXT)
      GOTO 110
C BACK FULL CIRCLE
104   CALL ERROR(A,307)
      CALL NEXTA
      RETURN
C ARE MORE PARAMS NEEDED
110   NPRQ=T(F,NPR)
      IF(NPRQ.LE.MAXM) GOTO 120
      NP=T(NPRQ,2)
      NPRQ=P(NP,STAT)
120   IF(P(F,STAT).GE.NPRQ) GOTO 100
150   IF(M.EQ.0) GOTO 100
C
      IF(P(M,TYP).GT.0) GOTO 156
C CONDITIONS OF ACCEPTING IMPLICIT PARAM
      TTYP=T(M,TYP)
      IF(TTYP.EQ.K) GOTO 160
      IF(TTYP.EC.INUM.AND.K.EQ.FNUM) GOTO 160
      IF(TTYP.EQ.FNUM.AND.K.EQ.INUM) GOTO 160
C THIS NEXT CASE IS DEBATABLE
      IF(KK.EQ.LIMIT) GOTO 160
      IF(TTYP.EC.0) GOTO 160
C LOOK AT NEXT M
156   M=T(M,NXT)
      GOTO 150
C FOUND ONE
160   PNAME=TNAME(M)
      CALL INSERP(PNAME,ATOM(A),CAT(A),M,F)
C CHECK FOR VALUE STRING
170   CALL NEXTA
      IF(TTYP.EC.K) GOTO 172
      IF(TTYP.EC.INUM.AND.K.EQ.FNUM) GOTO 172
      IF(TTYP.EQ.FNUM.AND.K.EQ.INUM) GOTO 172
      IF(KK.EQ.LIMIT) GOTO 172
      RETURN
172   CALL APPENP(M,ATOM(A),CAT(A))
      GOTO 170
      END
*****************************************************************
```

```
      SUBROUTINE TABCLO
C TO CLOSE CUT FILES
C       CALLED BY ULPREP
C       USES DIRECT ACCESS READ/WRITE
      COMMON /CICT/ONAME(63),OCAT(63),MAXO,O,CSTAT
     1,KWCNAM,KWI,IABIND(21)
      INTEGER ONAME,OCAT,MAXO,O,CSTAT
C
      IF(CSTAT.GT.0) CALL WRITNS(1,CNAME,127,SLLCICT)
      RETURN
C
C ---------------------------------------------
C
      ENTRY TABLOP
C TO OPEN CICTICNARY
C       CALLED BY ULPREP
C       USES DIRECT ACCESS READ/WRITE
      CALL OPENRS(1,IABIND,21,1)
      CSTAT=KWCNAM*8
      CALL READNS(1,ONAME,127,SLLOICT)
      LABEL=0
      PRINT 101
  101 FORMAT(1H1)
      RETURN
      END
C*****************************************************************
C
C
      SUBROUTINE ULKWC(NAME)
C TO LOOK FOR KEYWORD AND TO ANALYSE REQUEST
C       CALLED BY ULPREP
C       USES DIRECT ACCESS READ/WRITE
C
      COMMON/LLCAT2/TYP,NXT,BRO,SON,NPR,FAM,RES,MVAL,CV,LIM,VALLE
     2,STAT,TYP2, KNAME, MXDICT,PXSPNAM,PXINAM,PXICEL,MXPCEL
     3,EOS,STRING,FAUP,INUM,OPER,KWD,DESCOR, KWECL,ITNAME,CLNAME
     4,EXNAME,CNAME,KEYUL
      INTEGER ECS
C
      COMMON /CICT/ONAME(63),OCAT(63),MAXO,O,CSTAT
     1,KWCNAM,KWI,IABIND(21)
      INTEGER ONAME,OCAT,MAXO,O,CSTAT
      COMMON /TCHEK/ATOM(63),CAT(63),A,MAXA,K,KK,FILL(3)
      INTEGER ATOM,CAT,A,MAXA
      COMMON /TEMPL/MAXM,MAXV,TNAME(109),TCELL(400)
      COMMON/CTSTEP/ MAXP,SPNAME(10),PCELL(400)
      DATA KCPT1,KCPT2,KOPT3/3*0/
C ---------------------------------------------
C
      A=1
      CALL NEXTA
      IF(KK.EC.KEYUL) GOTO 50
      IF(KCPT1.EQ.1) CALL ECHOIN
IC FIND TEMPLATE
I       IF(KWCNAM.NE.0) GOTO 12
IC FIRST TEMPLATE
I       IF(K.EQ.KWD) GOTO 14
I       CALL ERROR(A,310)
I       RETURN
IC A TEMPLATE IS OPEN
I    12 IF(KWCNAM.EQ.ATOM(2)) GOTO 24
I       IF(K.NE.KWD) GOTO 24
IC READ NEW TEMPLATE
I    14 KWCNAM=ATOM(2)
I       CALL READNS(1,MAXM,511,KWCNAM)
I       MXP=PXPCEL*2
IC
IC INITIALIZE COMTAB
I    24 DO 30 I=1,MXP
I    30 PCELL(I)=0
I       MAXP=MAXM+MXSPNAM
I       DO 34 I=1,MXSPNAM
I    34 SPNAME(I)=0
I       LABEL=LABEL+10
I       IF(ATOM(1).GT.0) LABEL=ATOM(1)
I       NAME=KWONAM
I       ATOM(2)=KWONAM
I       CAT(2)=KWC*100
IC
I       A=1
I       CALL USERCOM
I       CALL USERSEM
I       IF(KCPT2.EQ.1) CALL ECHOUT
I       IF(KCPT3.EQ.1) CALL DUMPP
I       RETURN
IC
IC ULANG SYSTEM KEYWORD
I    50 IF(CAT(A).NE.4LCPTI) RETURN
IC ONLY *OPTIONS* IS HANDLED NOW
I    60 CALL NEXTA
I       IF(K.EQ.ECS) RETURN
I        IF(ATOM(A).EQ.6LECHOIN) KOPT1=1
I        IF(ATOM(A).EQ.6LECHOOU) KOPT2=1
I        IF(ATOM(A).EQ.5LDUMPP ) KOPT3=1
I        IF(ATOM(A).EQ.6LNOECIN) KOPT1=0
I        IF(ATOM(A).EQ.6LNOECOU) KOPT2=0
I        IF(ATOM(A).EQ.5LNODUP ) KOPT3=0
I        IF(ATOM(A).NE.4LVTES) GOTO 60
IC VTEST IS A TEST PROCEDURE
I       CALL VTEST
I       RETURN
I       END
I*****************************************************************
I
I
I
```

```
      SUBROUTINE USERCOM
C USER COMMAND ANALYSER
C       CALLED BY ULKNO
C
      COMMON /ULDAT2/ DUMMY2(19)
     3,EOS,STRING,FNUM,INUM,OPER,KWC,DESCON, KWCCL,ITNAME,CLNAME
     4,EXNAME,CNAME,EXEC,REQU,LPAREN,RPAREN,MINUS,ACP,EQUAL,TC,CF
     5,RESTYP,ULTYP,LLTYP, EOV,ECVCAT,LIMIT,KCNC
      INTEGER
     1 EOS,STRING,FNUM,INUM,OPER,KWC,DESCON,KWCCL,ITNAME,CLNAME
     2,EXNAME,CNAME,EXEC,REQU,LPAREN,RPAREN,MINUS,ACP,EQUAL,TC,CF
     3,RESTYP,ULTYP,LLTYP, EOV,ECVCAT
C
      COMMON /TOKEN/ATOM(63),CAT(63),A,MAXA,K,KK,DUMMY(3)
      INTEGER ATOM,CAT,A,MAXA,K,KK
      INTEGER REM
C ---------------------------------------------
C
C INITIALIZE
      LBM=RBM=MM=0
      P=M=PVALUE=0
C
C SCAN INPUT ATOM STRING
10    CALL NEXTA
20    IF(K.EC.0) GO TO 10
C CASE CAT(A) OF
      GO TO(100,200,200,400,500,600,700,700, 10,1000),K
C
C NAME
100   IF(KK.EC.ITNAME) GOTO 110
      IF(KK.EC.CLNAME) GOTO 190
      IF(KK.EC.EXNAME) GOTO 190
      CALL ERROR(A,301)
      GOTO 10
C
C SIMPLE ITEM NAME
110   M=0
      CALL INSERP(ATOM(A),PVALUE,0,M,F)
      IF(LBM.GT.0) RBM=A
      CALL NEXTA
C QUALIFIER MAY FOLLOW
114   IF(CAT(A).NE.OF) GOTO 118
      CALL CLAL(M,F)
      GOTO 114
C EQUAL MAY FOLLOW
118   IF(CAT(A).EQ.EQUAL.OR.K.EC.0) CALL NEXTA
C
C VALUE MAY FOLLOW
120   IF(K.EQ.INUM) GOTO 126
      IF(K.EC.FNUM.OR.K.EQ.STRING) GOTO 130
      IF(KK.EC.LIMIT) GOTO 140
      GOTO 20
C VALUE FOLLOWS
C REPEAT BRACKET MAY FOLLOW
126   IF(CAT(A+1).NE.LPAREN) GOTO 130
      CALL NEXTA
      CALL REPEAT
C FIND M TO WHICH THIS VALUE BELONGS
130   IF(LBM.EQ.0) GO TO 134
C ITEM NAME LIST
132   IF(MM.LT.LBM) MM=LBM-1
      MM=MM+1
      IF(MM.GT.RBM) GOTO 20
      IF(CAT(MM)/10.NE.ITNAME) GOTO 132
      M=MM
C STORE VALUE IN P
134   CALL APPENP(M,ATOM(A),CAT(A))
      KKS=KK
      CALL NEXTA
      IF(KK.EQ.KKS+1) GOTO 134
      GOTO 120
C LIMIT FOLLOWS
140   CALL LIM(M)
      GOTO 120
C
C CLASS NAME
160   IF(LBM.LE.0) LBM=A
      CALL SEMCLA(F)
      RBM=A
      CALL NEXTA
      GOTO 110
C
C EXPRESSION NAME
190   CONTINUE
      GOTO 10
C
C DESCRIPTIVE CONSTANT OR COMMAND
200   M=0
      CALL INSERP(ATOM(A),PVALUE,CAT(A),M,F)
      GOTO 10
C
C DESCRIPTOR
400   IF(KK.EQ.LIMIT) GOTO 430
      IF(KK.EQ.KOND) GOTO 420
      GOTO 10
420   CALL EEX
      GOTO 20
430   CALL SEMIMPL(M,F)
      GOTO 20
C
C OPERATOR
500   IF(CAT(A).EQ.LPAREN) GO TO 520
      IF(CAT(A).EQ.RPAREN) GOTO 530
      IF(CAT(A).EQ.EQUAL) GO TO 540
      CALL ERROR(A,302)
      GOTO 10
C LEFT PAREN
520   LBM=-1
```

```
      CALL NEXTA
      IF(KK.EQ.ITNAME.OR.KK.EQ.CLNAME) GO TO 524
      CALL ERRCR(A-1,382)
      LBR=0
      GO TO 28
524   LBR=A
      GOTO 28
C RIGHT PAREN
530   IF(LBR.NE.0) GOTO 534
      CALL ERROR(A,384)
      GOTO 10
C END OF ITEM GROUP
534   LBR=REP=PP=0
      GOTO 10
C EQUAL
540   IF(CAT(A-1).EQ.0) GOTO 550
      CALL ERRCR(A,383)
      GO TO 10
550   CAT(A-1)=EXNAME*10
      CALL AEX
      GOTO 28
C
C INTEGER
      IF(CAT(A+1).EQ.LPAREN) GOTO 650
      CALL SEMIPPL(M,F)
      GOTO 28
C REPEAT BRACKET
650   CALL NEXTA
      CALL REPEAT
      GOTO 28
C DECIMAL, STRING
700   CALL SEMIPPL(M,F)
      GOTO 28
C
C END OF SENTENCE
1000  RETURN
      END
*****************************************************************

      SUBROUTINE USERSEM
C TO CHECK USER SUPPLIED PARAMETERS
C AND TO SUPPLY MISSING ONES BY DEFAULT
C       CALLED BY ULKNO
C
      COMMON/LLCAT2/TYP,NXT,BRO,SON,NPR,FAM,RES,PVAL,CV,LIM,VALLE
     2,STAT,TYP2, KNAME, MXDICT,MXSENAM,MXINAM,MXICEL,MXPCEL
     3,EOS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWDCL,ITNAME,CLNAME
      INTEGER
     1 EOS,STRING,FNUM,INUM,OPER,KWD,DESCON,KWDCL,ITNAME,CLNAME
C
      COMMON /TEMPL/MAXM,MAXV,TNAME(109),TCELL(400)
      INTEGER F,T,P,V,V2,VL,PVALLE,PCAT
IC ------------------------------------------
IC
I     F=0
IC GET NEXT CLASS
I10   F=NEXTF(F)
I     IF(F.GT.0) GOTO 20
I     RETURN
I  20 M=T(F,NXT)
I     IF(M.EQ.0) GOTO 600
IC LOOK AT MEMBERS OF THIS CLASS
I200  IF(P(M,STAT).EQ.0) GOTO 500
IC M IS ACTIVE
I     MR=T(M,RES)
I     IF(MR.EQ.0) GOTO 300
IC M HAS RESTRICTIONS, SEE IF PRESENT
I     DO 240 I=2,8
I     KR=T(MR,I)
I     IF(KR.EQ.0) GOTO 250
I     IF(P(KR,STAT).GT.0) GOTO 300
I 240 CONTINUE
IC M SHOULD NOT BE ACTIVE THEN
I 250 CALL ERRORP(M,F,401)
I     GOTO 500
IC
IC CHECK VALUES
I 300 MT=T(M,TYP)
I     IF(MT.EQ.DESCON.OR.MT.EQ.KWD) GOTO 500
IC ITEM VALUES ARE CHECKED
I     V=T(M,MVAL)
IC IF M HAS VALUES
I 310 IF(V.LT.MAXV) GOTO 360
IC CHECK FOR ACTIVE RESTRICTIONS, OR NO RESTRICTIONS
I     VR=T(V,RES)
I     IF(VR.EQ.0) GOTO 350
I     IF(P(VR,STAT).GT.0) GOTO 350
IC LOOK AT NEXT V
I     V=T(V,NXT)
I     GOTO 310
IC
IC HAS USER SUPPLIED A P
I 350 IF(P(M,TYP).GT.0) GOTO 360
IC GET DEFAULT VALUE
I     PVALLE=T(V,VALUE)
I     PCAT=T(M,TYP)*100 + T(V,TYP)
I     CALL APPENP(M,PVALUE,PCAT)
I     GOTO 500
IC CHECK MODES AND LIMITS
I 360 IF(T(M,TYP).GT.0) CALL PBOUNDS(M,V)
IC LOOK AT NEXT M
I 500 M=T(M,NXT)
I     IF(M.GT.0) GOTO 200
IC
IC ------------------------------------------
IC CHECK NO OF REQUIRED PARAMS
```

```
  600 APRC=T(F,APR)
      IF(APRC.LE.MAXM) GOTO 620
C RPE OF ANOTHER CLASS
      NP=T(APRC,2)
      NPRC=P(NP,STAT)
C
  620 IF(P(F,STAT).GE.NPRO) GOTO 10
C
C EXTRA PARAMS NEEDED
      N=T(F,NXT)
      IF(N.EQ.0) GOTO 890
      NCOLAT=0
  650 NCOUNT=NCOUNT+1
      IF(P(N,STAT).GT.0) GOTO 800
C N INACTIVE
      NR=T(N,RES)
      IF(NR.EQ.0) GOTO 700
C SEE IF RESTRICTIONS ACTIVE
      DO 670 I=2,8
      NR=T(NR,I)
      IF(NR.EQ.0) GOTO 680
      IF(P(KN,STAT).GT.0) GOTO 700
  670 CONTINUE
C RESTRICTION INACTIVE
  680 N=T(N,NXT)
      IF(N.EQ.0) GOTO 10
      GOTO 650
C
C GET A VALUE
  700 NT=T(N,TYP)
      IF(NT.EQ.DESCON.OR.NT.EQ.KND) GOTO 750
C
C ITEP DEFAULT
      V=T(N,NVAL)
  710 IF(V.LT.MAXV) GOTO 800
      NR=T(V,RES)
      IF(NR.EQ.0) GOTO 720
      IF(P(NR,STAT).GT.0) GOTO 720
C LOCK AT NEXT V
      V=T(V,NXT)
      GOTO 710
C GET DEFAULT VALUE
  720 PVALUE=T(V,VALUE)
      PCAT=T(N,TYP)*100 + T(V,TYP)
      CALL INSERP(TNAME(N),PVALUE,PCAT,P,FCUM)
      GOTO 800
C
C DESCON DEFAULT
  750 PCAT=T(N,TYP)*100
      CALL INSERP(TNAME(N),PVALUE,PCAT,P,FCUM)
C
C SEE IF MORE PARAMS NEEDED
  800 IF(P(F,STAT).GE.NPRO) GOTO 10
      N=T(N,NXT)
      IF(N.GT.0)GOTO 650
      IF(P(F,STAT).GE.MIN0(MCOUNT,NPRO)) GOTO 10
IC NOT ENOUGH
  890 CALL ERRORP(F,NPRO,402)
      GOTO 10
      END
```

```
      INTEGER FUNCTION FINDP(ARG,STATUS)
* TO FIND PARAM NO IN TNAME
C        CALLED BY V-FUNCTIONS
      INTEGER ARG,STATUS,TNAME
      COMMON /TEMPL/MAXM,MAXV,TNAME(100)
*
      STATUS=1
      IF(IABS(ARG).LE.MAXM) GOTO 500
* SEARCH TABLE
      KEY=I=1
      J=MAXP
C SEARCH LOOP
  100 IF(J.LT.I) GO TO 300
      KEY= (I+J)/2
      IF(ARG.LT.TNAME(KEY)) J=KEY-1
      IF(ARG.GT.TNAME(KEY)) I=KEY+1
      IF(ARG.NE.TNAME(KEY)) GOTO 100
C FOUND
      FINDP=KEY
      RETURN
C ELSE NOT FOUND
  300 FINDP=I
      STATUS=0
      RETURN
* NUMERIC ARG
  500 FINDP=ARG
      IF(FINDP.LT.1) STATUS=0
      RETURN
      END
*****************************************************************


      INTEGER FUNCTION V(ARG)
C        CALLED BY USER PROGRAMS
* TO RETRIEVE PARAM VALUES TO USER PROGRAM
C VPREV POINTS TO PREVIOUS VALUE, VFIRST TO FIRST
      INTEGER ARG,ARGPR,P,EOV,FINDP,STATUS,VLIM,VPREV,VFIRST
      COMMON /RLNDAT/EOV,TYP,NXT,CV,STAT,TYP2,VALUE,VPREV,VFIRST
      DATA ECV,TYP,NXT,CV,STAT,TYP2,VALUE,VPREV,VFIRST
     ./999999,1,2,3,4,5,20,-1,1/
      DATA ARGPR/0/
*
      ENTRY IV
      IF(ARG.EQ.ARGPR) GOTO 50
      IF(ARG.EQ.VPREV) NXV=NXVP
      IF(ARG.EC.VFIRST) NXV=M
      IF(ARG.EC.VPREV.OR.ARG.EQ.VFIRST) GOTO 50
      P=FINDP(ARG,STATUS)
      IF(STATUS.EQ.0) GOTO 500
*
      IF(P(P,STAT).EQ.0) GOTO 500
* ACTIVE PARAM
      ARGPR=ARG
      NXV=P(M,CV)
*
   50 IF(NXV.EQ.0) GOTO 500
      IF(P(NXV,TYP).EC.4) GOTO 400
* NEXT VALUE
      V=P(NXV,VALUE)
      NXVP=NXV
      NXV=P(NXV,NXT)
      CALL PSET(M,CV,NXV)
      IV=V
      RETURN
*
  400 IF(P(NXV,TYP2).EQ.1) GOTO 420
* FROM, TO, BY
      V=VLIM(ARG,NXV)
      CALL PSET(M,CV,NXV)
      IF(V.EQ.EOV) GOTO 50
      IV=V
      RETURN
* QUALIFIER
  420 MQ=P(NXV,VALUE)
      CALL PSET(MQ,CV,MQ)
      NXV=P(NXV,NXT)
      CALL PSET(M,CV,NXV)
      GO TO 50
*
* NOT ACTIVE, OR VALUES EXHAUSTED
  500 V=ECV
      ARGPR=0
      IV=V
      RETURN
      END
*****************************************************************


      INTEGER FUNCTION VLIM(ARG,NXV)
*TO GET CURRENT VALUE , SPECIFIED AS  FROM LLV TO ULV BY INC
C        CALLED BY V-FUNCTIONS
      INTEGER ARG,ARGPR,ULV,ULSTAT,P,PTYP,ECV
      EQUIVALENCE (LLV,FLLV),(ULV,FULV),(INC,FINC)
      COMMON /RLNDAT/EOV,TYP,NXT,CV,STAT,TYP2,VALUE
      DATA ARGPR/0/
*
      IF(ARG.EQ.ARGPR) GOTO 400
* INITIALISE LIMIT
      LLV=LLV=INC=0
      LLSTAT=ULSTAT=INCSTAT=0
* INIPTR POINTS TO INIVES (=1 IF INITIALIZED)
      INIPTR=P(NXV,NXT)
      INIVES=P(INIPTR,STAT)
      IF(INIVES.EQ.1) LLV=P(NXV,VALUE)
      ARGPR=ARG
      NXP=NXV
```

```
C
C LOOP OVER LIMIT SPECS
  310 LTYP=P(NXP,TYP2)
      NXP=P(NXP,NXT)
      IF(NXP.EQ.0) GOTO 500
      PTYP=P(NXP,TYP)
      IF(PTYP.EQ.6.OR.PTYP.EQ.7) GOTO 320
      IF(PTYP.EQ.8.OR.PTYP.EQ.2) GOTO 320
      GOTO 500
C
C LIMIT MARKER
  320 IF(LTYP.EQ.31) GOTO 340
      IF(LTYP.EQ.32) GOTO 350
      IF(LTYP.EQ.33) GOTO 360
      GOTO 500
C LOWER LIMIT FROM
  340 IF(LLSTAT.EQ.1) GOTO 360
      LLSTAT=1
      IF(INIVES.EQ.1) GOTO 370
      LLV=P(NXP,VALUE)
      CALL PSET(INIPTR,STAT,1)
      GOTO 370
C UPPER LIMIT TO
  350 IF(ULSTAT.EQ.1) GOTO 360
      ULSTAT=1
      ULV=P(NXP,VALUE)
      GOTO 370
C INCREMENT BY
  360 IF(INCSTAT.EQ.1) GOTO 380
      INCSTAT=1
      INC=P(NXP,VALUE)
C LOOK AT NEXT VALUE SPEC
  370 NXP=P(NXP,NXT)
      IF(NXP.EQ.0) GOTO 380
      IF(P(NXP,TYP).EQ.4) GOTO 310
C
C CHECK INCREMENT
  380 IF(PTYP.EQ.7) GOTO 362
      IF(INC.EQ.0) INC=1
      IF(INC.LT.0.AND.LLV.LT.ULV) INC=IABS(INC)
      IF(INC.GT.0.AND.LLV.GT.ULV) INC=-INC
      GOTO 400
  362 IF(FINC.EQ.0) FINC=1.0
      IF(FINC.LT.0.AND.FLLV.LT.FULV) FINC=ABS(FINC)
      IF(FINC.GT.0.AND.FLLV.GT.FULV) FINC=-FINC
C
C RETURN VALUE AND ADVANCE CV
  400 CONTINUE
      VLIM=LLV
      IF(PTYP.NE.6) GOTO 410
      IF(INC.GT.0.AND.LLV.GT.ULV) GOTO 500
      IF(INC.LT.0.AND.LLV.LT.ULV) GOTO 500
      LLV=LLV+INC
      CALL PSET(NXV,VALUE,LLV)
      RETURN
C REAL VALUES
  410 IF(PTYP.NE.7) GOTO 430
      IF(FINC.GT.0.AND.FLLV.GT.FULV) GOTO 500
      IF(FINC.LT.0.AND.FLLV.LT.FULV) GOTO 500
      FLLV=FLLV+FINC
      CALL FSET(NXV,VALUE,FLLV)
      RETURN
C FOR STRINGS AND DC#S RETURN LL AND UL ONLY, NO INCREMENTS
  430 IF(PTYP.NE.8.AND.PTYP.NE.2) GOTO 500
      IF(LLV.EQ.EOV) GOTO 500
      IF(LLV.LT.ULV) LLV=LLV
      IF(LLV.EQ.ULV) LLV=EOV
      CALL FSET(NXV,VALUE,LLV)
      RETURN
C NO MORE
  500 VLIM=EOV
      NXV=NXP
      ARGPR=0
      RETURN
      END
C*****
C
C
      INTEGER FUNCTION VM(ARG,TABLE,N)
C        CALLED BY USER PROGRAM
C TO RETRIEVE ALL VALUES OF ARG IN TABLE(N)
C VM IS SET TO NO OF VALUES RETRIEVED
C CV POINTER IS NOT MOVED
      INTEGER TABLE(1),ARG,F,EOV,FINDP,STATUS,VLIM
      COMMON /RLNDAT/EOV,TYP,NXT,CV,STAT,TYP2,VALUE
C
      I=0
      IF(N.LE.0) GOTO 500
      M=FINDP(ARG,STATUS)
      IF(STATUS.EQ.0) GOTO 500
      IF(P(M,STAT).EQ.0) GOTO 500
C ACTIVE ARG
      NXV=M
C
   50 IF(NXV.EQ.0) GOTO 500
      IF(P(NXV,TYP).EQ.4) GOTO 400
C NEXT VALUE
      I=I+1
      IF(I.GT.N) GOTO 510
      TABLE(I)=P(NXV,VALUE)
      NXV=P(NXV,NXT)
      GOTO 50
C
  400 IF(P(NXV,TYP2).EQ.1) GOTO 420
C        LIMIT LOOP
  410 IV=VLIM(ARG,NXV)
      IF(IV.EQ.EOV) GOTO 50
```

```
      I=I+1
      IF(I.GT.N) GOTO 510
      TABLE(I)=IV
      GOTO 410
C QUALIFIER
  420 NXV=F(NXV,NXI)
      GOTO 50
C NOT ACTIVE, CR VALUES EXHAUSTED
  500 VN=I
      RETURN
C TABLE FULL
  510 I=I-1
      RETURN
      END
*****************************************************************

      INTEGER FUNCTION VP(ARG,INDEX)
C        CALLED BY USER-PROGRAMS
C TO SET CV POINTER TO THE INDEX-TH VALUE
      INTEGER ARG,F,EOV,FINDP,STATUS,VLIM
      COMMON /RLNDAT/EOV,TYP,NXT,CV,STAT,TYP2,VALUE
C
      NXV=0
      F=FINDP(ARG,STATUS)
      IF(STATUS.EQ.0) GOTO 500
      NXV=F
      ICT=1
      IF(INDEX.LE.1) INDEX=1
C LOOP AND COUNT  IN ICT NO OF VALUES ENCOUNTERED
   50 IF(ICT.EQ.INDEX) GOTO 500
      NV=F(NXV,NXT)
      IF(NV.EQ.0) GOTO 500
      ICT=ICT+1
      NXV=NV
      IF(F(NXV,TYP).EQ.4) GOTO 400
      GOTO 50
C
  400 IF(F(NXV,TYP2).EQ.1) GOTO 50
C A LIMIT ECCF COUNTS AS ONE
  410 IV=VLIM(ARG,NXV)
      IF(IV.NE.EOV) GOTO 410
      GOTO 50
C
  500 CALL FSET(M,CV,NXV)
      VP=NXV
      RETURN
      END
*****************************************************************

      INTEGER FUNCTION VS(ARG,AVALUE,ACAT,ACAT2)
C        CALLED BY USER PROGRAMS
C TO SET CV OF ARG TO AVALUE, ACAT IS ITS TYPE
      INTEGER F,FINDP,STATUS,AVALUE,ACAT,ACAT2,ASTAT,FNUM
      COMMON /RLNDAT/EOV,TYP,NXT,CV,STAT,TYP2,VALUE
      DATA FNUM/7/
C
      M=FINDP(ARG,STATUS)
      IF(STATUS.EQ.0) GOTO 500
      NXV=F(M,CV)
      IF(NXV.EQ.0) NXV=M
C  SET VALUE
      CALL FSET(NXV,VALUE,AVALUE)
C SET TYPES, FLOATING PT IS DEFAULT
      K1=ACAT
      IF(K1.LE.0.OR.K1.GT.15) K1=FNUM
      CALL FSET(NXV,TYP,K1)
      K2=ACAT2
      IF(K2.LE.0.OR.K2.GT.99) K2=0
      CALL FSET(NXV,TYP2,K2)
C CHANGE STATUS OF M
      ASTAT=P(M,STAT)
      ASTAT=ASTAT.OR.8
      CALL FSET(M,STAT,ASTAT)
      CALL FSET(M,CV,NXV)
      VS=NXV
      RETURN
C  ERROR EXIT, VS SET TO ZERO
  500 VS=0
      RETURN
      END
*****************************************************************


      FUNCTION VXA(INDEX)
C        CALLED BY USER PROGRAM
C TO EVALUATE USER-DEFINED EXPRESSION(INDEX)
C INDEX IS SET TO POSITION IN SPNAME
C        RETURNS VXA=VXB=EOV IF UNDEFINED
C                VXB=1 IF TRUE, VXB=0 IF FALSE
C                VXA= FLT PT RESULT IF COMPUTABLE
C
      INTEGER EXNAME,XTYP,FNUM
      INTEGER STK(20),KIND(20),F
      DIMENSION FSTK(20)
      EQUIVALENCE(STK,FSTK)
      COMMON /TEMPL/MAXM
      COMMON /RLNDAT/EOV,TYP,NXT,CV,STAT,TYP2,VALUE
      DATA MXSPNM,EXNAME,KCNAME,INUM,FNUM/10,30,31,6,7/
C
      XTYP=2
      GOTO 2
C
```

```
C BOOLEAN EXPRESSION ENTRY POINT
      ENTRY VXB
      XTYP=4
C
    2 IF(INDEX.LE.0.OR.INDEX.GT.MXSPNAM) GOTO 900
      MSP=0
      I=0
C FIND EXP IN SPNAMES
      DO 10 K=1,MXSPNAM
      M=MAXM+K
      PTYP=P(M,TYP)
      IF(PTYP.NE.1) GOTO 10
      IF(XTYP.EQ.2.AND.P(M,TYP2).EQ.EXNAME) MSP=MSP+1
      IF(XTYP.EQ.4.AND.P(M,TYP2).EQ.KONAME) MSP=MSP+1
      IF(MSP.EQ.INDEX) GOTO 20
   10 CONTINUE
      INDEX=0
      GOTO 900
C FOUND, M IS INDEX IN SPNAME
   20 INDEX=M
      K=P(M,NXT)
      IF(K.EQ.0) GOTO 900
C
C LOOP OVER POLISH STRING OF PSTACK
   50 KTYP=P(K,TYP)
      IF(KTYP.EQ.0) GOTO 900
      GOTO(100,900,900,900,500,100,100,100,900),KTYP
C ITEM-NAME, INDEX IS STORED IN VALUE
  100 KIT=P(K,VALUE)
      IF(P(KIT,STAT).EQ.0) GOTO 900
      KXV=P(KIT,CV)
      IF(KXV.EQ.0) GOTO 900
      IF(P(NXV,TYP).EQ.0) GOTO 900
C STACK IT
      I=I+1
      IF(I.GT.20) GOTO 900
      STK(I)=P(KXV,VALUE)
      KIND(I)=P(NXV,TYP)
      GOTO 198
C INTEGER, REAL, STRING VALUE
  100 I=I+1
      IF(I.GT.20) GOTO 900
      STK(I)=P(K,VALUE)
      KIND(I)=P(K,TYP)
C NEXT PSTACK ENTRY
  198 K=P(K,NXT)
      IF(K.GT.0) GOTO 50
      GOTO 900
C OPERATOR
C CONSIDER EVERYTHING FLOATING-POINT
  500 IF(KIND(I-1).EQ.INUM) FSTK(I-1)=STK(I-1)
      IF(KIND(I).EQ.INUM) FSTK(I)=STK(I)
      KIND(I-1)=KIND(I)=FNUM
C BRANCH ON OPERATOR TYPE
      TRU=0
      K2=P(K,TYP2)
      IF(K2.EQ.10) GOTO 510
      IF(K2.EQ.20) GOTO 520
      IF(K2.EQ.41) GOTO 541
      IF(K2.EQ.43) GOTO 543
      IF(K2.EQ.45) GOTO 545
      IF(K2.EQ.51) GOTO 551
      IF(K2.EQ.52) GOTO 552
      IF(K2.EQ.61) GOTO 561
      IF(K2.EQ.62) GOTO 562
      IF(K2.EQ.70) GOTO 570
      GOTO 900
C LOGICAL OPERATORS
  510 IF(FSTK(I-1).EQ.1..OR.FSTK(I).EQ.1.) TRU=1
      GOTO 600
  520 IF(FSTK(I-1).EQ.1..AND.FSTK(I).EQ.1.) TRU=1
      GOTO 600
C RELATIONAL OPERATORS
  541 IF(FSTK(I-1).LT.FSTK(I)) TRU=1
      GOTO 600
  543 IF(FSTK(I-1).EQ.FSTK(I)) TRU=1
      GOTO 600
  545 IF(FSTK(I-1).GT.FSTK(I)) TRU=1
      GOTO 600
C ARITHMETIC OPERATORS
  551 FSTK(I-1)=FSTK(I-1)+FSTK(I)
      GOTO 610
  552 FSTK(I-1)=FSTK(I-1)-FSTK(I)
      GOTO 610
  561 FSTK(I-1)=FSTK(I-1)*FSTK(I)
      GOTO 610
  562 FSTK(I-1)=FSTK(I-1)/FSTK(I)
      GOTO 610
C EXPONENTIATION
C  INTEGER POWERS ONLY FOR NOW
  570 IEXP=FSTK(I)
      FSTK(I-1)=FSTK(I-1)**IEXP
      GOTO 610
C ADJUST STACK
  600 FSTK(I-1)=TRU
  610 I=I-1
      K=P(K,NXT)
      IF(K.GT.0) GOTO 50
C FINISHED
      IF(I.NE.1) GO TO 900
      IF(XTYP.EQ.2) VXA=FSTK(1)
      IF(XTYP.EQ.4) VXB=FSTK(1)
      RETURN
C ERROR EXIT
  900 VXA=ECV
      VXB=ECV
      RETURN
      END
```

```
      PROGRAM ULSETUP(INPUT,OUTPUT,TAPE5=INPUT,TAPE1=512)
C ULANG SETUP MODULE
C
      COMMON/ULCAT3/DICMAN,TEMMAN,ITCAT,UL,LL,FOR,SETMARK
     2,MAXDS,SDNAME(22),SDCAT(22)
      INTEGER DICMAN,TEMMAN,ITCAT,UL,LL,FOR,SETMARK
     2,MAXDS,SDNAME,SDCAT
C
      INTEGER KWDTYP,STATUS,FIND
      COMMON /TOKEN/ATOM(63),CAT(63),A,MAXA,K,KK,FILL(3)
      INTEGER ATOM,CAT,A,MAXA
      COMMON /DICT/DNAME(63),DCAT(63),MAXD,D,DSTAT
      INTEGER DNAME,DCAT,MAXD,D,DSTAT
C ---------------------------------------------
C
      CALL SECOND(SEK1)
      CALL STABCP
      IEOF=0
      KWDACT=0
C INPUT LOOP
   10 CALL INPUT(IEOF)
      IF(IECF.EQ.1) GO TO 100
      CALL ULSCAN
C
C SETUP ROUTINES
      A=1
      IF(CAT(2).GT.0) GOTO 20
C IF NO KWD INPUT, USE PREVIOUS ONE
      ATOM(2)=KWDACT
      IC=FIND(DNAME,MAXD,KWDACT,STATUS)
      IF(STATUS.EQ.1) CAT(2)=DCAT(IC)
   20 CALL NEXTA
      KWDTYP=PCE(CAT(A),10)
      KWDACT=ATOM(2)
      IF(KK.EQ.DICMAN) GOTO 3800
      IF(KK.EQ.TEMMAN) GOTO 3900
      GOTO 10
C
C DICTIONARY CONSTRUCTION
 3800 IF(KWDTYP.LT.1.OR.KWDTYP.GT.3) GOTO 10
      GOTO(810,820,830), KWDTYP
  810 CALL CDBUILD
      GOTO 10
  820 CALL UDBUILD
      GO TO 10
  830 CALL DICSYN
      GOTO 10
C
C TEMPLATE CONSTRUCTION
 3900 IF(KWDTYP.EQ.0) CALL DEFINE(2)
      IF(KWDTYP.LT.1.OR.KWDTYP.GT.8) GOTO 10
      GOTO(910,920,930,940,950,960,970,980),KWDTYP
  910 CALL DISPLAY
      GOTO 10
  920 CALL CLINK
      GOTO 10
  930 CALL CLASS(1)
      GOTO 10
  940 CALL CLASS(2)
      GOTO 10
  950 CALL CVALUE
      GOTO 10
  960 CALL CATEG
      GOTO 10
 970  GOTO 10
  980 CALL NEXTA
      CALL DUMPTC(ATOM(A))
      GOTO 10
C END OF REQUESTS
  100 CALL SECOND(SEK2)
      SEK=SEK2-SEK1
      PRINT 101, SEK
  101 FORMAT(40X,F8.3,' SEC. ')
      CALL STABCLO
      STOP
      END
C*****************************************************************
C
C
      BLOCKDATA CONSET
C DEFINES CONSTANTS USED BY SETUP ONLY
      COMMON/ULCAT3/DICMAN,TEMMAN,ITCAT,UL,LL,FOR,SETMARK
     2,MAXDS,SDNAME(22),SDCAT(22)
      INTEGER DICMAN,TEMMAN,ITCAT,UL,LL,FOR,SETMARK
     2,MAXDS,SDNAME,SDCAT
C ---------------------------------------------
C                    DEFINITIONS FOR ULCAT3
C MISCEL CATEGORIES
      DATA  DICMAN,TEMMAN,ITCAT,UL,LL,FOR,SETMARK
     2/38,39,49,482,481,450,561/
C
C INITIAL SETUP DICTIONARY OF SIZE MAXDS
      DATA MAXDS /22/
      DATA SDNAME
     1/1LA,3LALL,4LCATE,4LCLAS,4LDCLA,4LDEFI,4LDICT,4LDISP
     1,5LDUMPT,1LF,3LFOR,1LI,2LIS,4LLINK,2LLL,2LOF,4LSEQU,4LSYNO
     2,2LTC,4LUDIC,2LLL,4LVALU/
      DATA SDCAT/498,443,396,393,394,390,381,391,398,497,450,496
     1,543,392,481,401,397,383,432,382,482,395/
      END
C*****************************************************************
```

```
      SUBROUTINE CATEG                                                  I     4,EXNAME,CNAME,EXEC,REQU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TC,CF
C TO ASSIGN TYPES TO MEMBERS                                            I      INTEGER
C       CALLED BY ULSETUP                                               I     1 ECS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWDCL,ITNAME,CLNAME
C                                                                       I     2,EXNAME,CNAME,EXEC,REQU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TC,CF
      COMMON/ULCAT2/TYP,NXT,BRO,SON,NPR,FAM,RES,MVAL,PNO,LIM,VALUE      I      COMMON/ULDAT3/DICMAN,TEMMAN,ITCAT,UL,LL,FCR,SETMARK
     2,STAT,TYP2, KNAME, MXDICT,MXSPNAM,MXTNAM,MXTCEL,MXPCEL           I      INTEGER DICMAN,TEMMAN,ITCAT,UL,LL,FCR,SETMARK
     3,ECS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWDCL,ITNAME,CLNAME        IC
     4,EXNAME,CNAME,EXEC,REQU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TC,CF       I      INTEGER F,FIND,STATUS,T,REA,CLTYPE,SWALUE,FCAT
      INTEGER                                                           I      COMMON /TOKEN/ATOM(63),CAT(63),A,MAXA,K,KK
     1 ECS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWDCL,ITNAME,CLNAME        I      INTEGER ATOM,CAT,A,K,KK
     2,EXNAME,CNAME,EXEC,REQU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TC,CF       I      COMMON /TEMPL/MAXM,MAXV,TNAME(109),TCELL(400)
      COMMON/ULCAT3/DICMAN,TEMMAN,ITCAT,UL,LL,FCR,SETMARK               IC ---------------------------------------------
C                                                                       IC
      COMMON /TOKEN/ATOM(63),CAT(63),A,MAXA,K,KK                        IC CLASS-NAME
      INTEGER ATOM,CAT,A,K,KK                                           I      CALL NEXTA
      COMMON /TEMPL/MAXM,MAXV,TNAME(109),TCELL(400)                     I      F= FIND(TNAME,MAXM,ATOM(A),STATUS)
      INTEGER FIND,STATUS,T                                             I      IF(STATUS.EQ.1.AND.T(F,TYP)/10.EQ.1) GOTO 20
C ---------------------------------------------                        I      CALL ERROR(A,501)
C                                                                       I      RETURN
      CALL NEXTA                                                        IC
      IF(KK.NE.ITCAT) GOTO 90                                           I   20 FCAT=T(F,TYP)
C GET TYPE                                                              I      CALL NEXTA
   20 MTYP=MOD(CAT(A),10)                                               I      IF(CAT(A).NE.EQUAL) A=A-1
      IF(CAT(A+1).EQ.EQUAL) CALL NEXTA                                  I      M=LBA=RBA=LBF=0
C MEMBER NAME LOOP                                                      IC
  100 CALL NEXTA                                                        IC MEMBER NAMES
      IF(K.EC.ECS) RETURN                                               I100   CALL NEXTA
      IF(KK.EQ.ITCAT) GOTO 20.                                          I      IF(K.EQ.KWD) K=0
C                                                                       I      IF(K.EQ.0) GOTO 200
      M=FIND(TNAME,MAXM,ATOM(A),STATUS)                                 I      IF(CAT(A).EQ.FOR) GOTO 400
      IF(STATUS.EQ.1) GOTO 140                                          I      IF(CAT(A).EQ.LPAREN) GOTO 290
      CALL ERROR(A,506)                                                 I      IF(CAT(A).EQ.RPAREN) GOTO 280
      GOTO 100                                                          I      IF(K.EQ.ECS) RETURN
  140 MTYP=T(F,TYP)                                                     I      CALL ERROR(A,502)
      IF(MTYP.GE.10) GOTO 190                                           I      RETURN
      CALL ISET(M,TYP,KTYP)                                             IC NAME
      GOTO 100                                                          I  200 MTYPE=0
C ERRORS                                                                I      IF(CLTYPE.EQ.2) MTYPE=DESCON
   90 CALL ERROR(A,513)                                                 I      IF(FCAT.EC.KWDCL) MTYPE=KWD
      RETURN                                                            I      M=INSERT(ATOM(A),MTYPE,F)
  190 CALL ERROR(A,502)                                                 I      IF(M.GT.0) GOTO 206
      GOTO 100                                                          I      CALL ERROR(A,503)
      END                                                               I      RETURN
C-----------------------------------------------------------------------IC
                                                                        I206   IF(LBA.LT.0) LBA=A
                                                                        I      RBA=A
                                                                        I      CALL TSET(M,FAM,F)
      SUBROUTINE CLASS(CLTYPE)                                          I      MB=I(F,NXT)
C DEFINE MEMBERS OF A CLASS WITH RESTRICTIONS                           I      IF(MB.GT.0) GOTO 210
C       CALLED BY ULSETUP                                               IC FIRST MEMBER IN CLASS
C                                                                       I      CALL TSET(F,NXT),M)
      COMMON/ULCAT2/TYP,NXT,BRO,SON,NPR,FAM,RES,MVAL,PNO,LIM,VALUE      I      IF(CLTYPE.EQ.1.AND.FCAT.NE.KWDCL) GOTO 230
     2,STAT,TYP2, KNAME, MXDICT,MXSPNAM,MXTNAM,MXTCEL,MXPCEL           IC FIRST DESCON OR KWD
     3,ECS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWDCL,ITNAME,CLNAME        I      CALL TSET(M,MVAL,1)
```

```
      CALL TSET(M,TYP,MTYPE)
      GOTO 100
C FIND LAST MEMBER IN CLASS
210   MXPE= T(PE,NXT)
      IF(MXPE.EQ.0) GOTO 220
      MR=MXPE
      GOTO 210
220   CALL TSET(MR,NXT,M)
C FOR DES CCAS SET VALUE
      IF(CLTYPE.NE.2.AND.FCAT.NE.KWDCL) GOTO 230
      SVALUE= T(MR,MVAL) + 1
      CALL TSET(M,MVAL,SVALUE)
      CALL TSET(M,TYP,MTYPE)
      GOTO 100
C FOR ITEMS SET TYPE, F IS DEFAULT-TYPE
230   MTYPE=FNUM
      IF(CAT(A+1).NE.SETMARK) GOTO 240
      CALL NEXTA
      IF(NR.EQ.ITCAT) MTYPE=MOD(CAT(A),10)
240   CALL TSET(M,TYP,MTYPE)
      GOTO 100
C
C LEFT PAREN
400   IF(LBA.GT.0) CALL ERROR(A,504)
      LBA=-1
      GOTO 100
C RIGHT PAREN
280   IF(LBA.EQ.0) CALL ERROR(A,504)
      GOTO 100
C
C RESTRICTION MARKER
485   IF(P.NE.0) GOTO 500
      CALL ERROR(4,505)
      RETURN
C ARRAY RESTRICTION
500   CALL NEXTA
      IF(K.EQ.KWD) K=0
      IF(P.EQ.0) GOTO 600
      IF(CAT(B).EQ.LPAREN) GOTO 690
      IF(CAT(B).EQ.RPAREN) GOTO 680
      IF(K.EQ.ECS) RETURN
      CALL ERROR(A,502)
      RETURN
C FIND RESTRICTION
600   MR= FIND(TNAME,MAXM,ATOM(A),STATUS)
      IF(STATUS.EQ.1) GOTO 610
      CALL ERROR(A,506)
      RETURN
610   IF(T(MR,TYP)/10.EQ.1) GOTO 720
      CALL CUALR(MR,LBA,RBA)
      IF(LBF.NE.0) GOTO 500
      LBA=0
      GOTO 100
C LEFT PAREN
690   IF(LBF.LT.0) CALL ERROR(A,504)
      LBF=-1
      GOTO 500
C RIGHT PAREN
680   IF(LBF.EQ.0) CALL ERROR(A,504)
      LBF=LBA=RBA=0
      GOTO 100
      END
C CLASS-NAME RESTRICTION
720   MR= T(MR,NXT)
      IF(MR.EQ.0) GOTO 730
      CALL CUALR(MR,LBA,RBA)
      GOTO 720
730   IF(LBF.NE.0) GOTO 500
      LBA=0
      GOTO 100
C LEFT PAREN
690   IF(LBF.LT.0) CALL ERROR(A,504)
      LBF=-1
      GOTO 500
C RIGHT PAREN
680   IF(LBF.EQ.0) CALL ERROR(A,504)
      LBF=LBA=RBA=0
      GOTO 100
      END
C*****************************************************************
C
C
C
C     SUBROUTINE CLINK
C TO FORM LINKAGES BETWEEN CLASSES AND TO ASSIGN FCELLS
C       CALLED BY ULSETUP
C
      COMMON/ULDAT2/TYP,NXT,BRO,SCN,NPR,FAM,RES,MVAL,PNO,LIM,VALUE
     2,STAT,TYP2, KNAME, MXDICT,MXSFNAM,MXTNAM,MXTCEL,MXPCEL
     3,ECS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWDCL,ITNAME,CLNAME
     4,EXNAME,GNAME,EXEC,RECU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TO,OF
     INTEGER
     1 EOS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWDCL,ITNAME,CLNAME
     2,EXNAME,GNAME,EXEC,RECU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TO,OF
C
      INTEGER F,FS,T,FIND,STATUS
      COMMON /TOKEN/ATOM(63),CAT(63),A,MAXA,K,KK
      INTEGER ATOM,CAT,A,K,KK
      COMMON /TEMPL/MAXM,MAXV,TNAME(109),TCELL(400)
C ---------------------------------------------
C
      IF(MAXM.GT.0) GOTO 100
C NO ENTRIES, KWD CLASS IS FIRST BY DEFAULT
      F=INSERT(KNAME,KWDCL,0)
      CALL TSET(F,TYP,KWDCL)
      CALL TSET(F,NPR,1)
C
C CLASS-1, F IS INDEX
100   CALL NEXTA
      IF(K.EQ.KWD) K=0
      IF(K.NE.0) RETURN
      F=FIND(TNAME,MAXM,ATOM(A),STATUS)
      IF(STATUS.EQ.1) GOTO 120
C NAME NOT YET ENTERED, ATTACH TO F= (KWD CLASS)
      F=FIND(TNAME,MAXM,KNAME,STATUS)
      CALL INSERE(F,FS,ATOM(A))
```

```
      F=FS
      GOTO 130
C NAME EXISTS ALREADY
  120 IF((F.IVS)/10.NE.1) GOTO 1120
      CALL NEXTA
C TO
  130 IF(CAT(A).NE.TO) RETURN
C
C FORM LINKAGES  CLASS=N
  200 CALL NEXTA
      IF(K.EC.KWD) K=0
      IF(K.NE.0) GOTO 250
      CALL INSERP(F,FS,ATOM(A))
      GOTO 210
C TO
  250 IF(CAT(A).NE.TO) RETURN
      F=FS
      GOTO 200
C
C ERROR
 1120 CALL ERROR(A,501)
      RETURN
      END
C-----------------------------------------------------------------
C
C SUBROUTINE BUILD
C TO BUILD DICTIONARIES
C     CALLED BY ULSETUP
      COMMON /ULDAT2/ DUMMY(14),PXDICT
C
      COMMON /TOKEN/ATOM(63),CAT(63),A,MAXA,FILL(5)
      INTEGER ATOM,CAT,A,MAXA
      COMMON /DICTION/NAME(63),DCAT(63),MAXD,D,DSTAT
      INTEGER NAME,DCAT,MAXD,D,ARG(2),DSTAT
      COMMON /LDICT/UDNAME(63),LDCAT(63),MAXUD,LD,UDSTAT
      INTEGER KWDNAM,KWI,TABIND(21)
      INTEGER LDNAME,UDCAT,UD,UDSTAT,TABIND
C-----------------------------------------------------------------
C
C INSERT ATOMS IN DICT
      DO 188 A=3,MAXA,2
      ARG(1)=ATOM(A)
      ARG(2)=ATOM(A+1)
      D=INSERX(LDNAME,PXDICT,MAXD,2,ARG)
  188 CONTINUE
      DSTAT=1
      RETURN
C -----------------------------------------------------------------
C
      ENTRY LDBUILD
C TO BUILD USER DICTIONARY
      DO 288 A=3,MAXA,2
      ARG(1)=ATOM(A)
      ARG(2)=ATOM(A+1)
      D=INSERX(UDNAME,MXDICT,MAXUD,2,ARG)
  288 CONTINUE
      UDSTAT=1
      RETURN
      END
C*****************************************************************
C
C
C
      SUBROUTINE DEFINE(KEY)
C TO OPEN AND CLOSE TEMPLATES
C    KEY=1 CLOSE ONLY
C    KEY=2 CLOSE FIRST THEN OPEN
C        CALLED BY ULSETUP, STABCLO.  USE DIRECT ACCESS READ/WRITE
C
      COMMON/ULDAT2/IYP,NXT,BRO,SON,NPR,FAM,RES,MVAL,PNO,LIM,VALUE
     2,STAT,TYP2, KNAME, MXDICT,MXSPNAM,MXTNAM,MXTCEL,MXPCEL
     3,ECS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWDCL,ITNAME,CLNAME
      INTEGER
     1 ECS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWDCL,ITNAME,CLNAME
C
      COMMON /UDICT/UDNAME(63),UDCAT(63),MAXUD,UD,UDSTAT
     1,KWDNAM,KWI,TABIND(21)
      INTEGER UDNAME,UDCAT,UD,UDSTAT,TABIND
      COMMON /TEMPL/MAXM,MAXV,TNAME(109),TCELL(400)
      COMMON /TOKEN/ATOM(63),CAT(63),A,MAXA,K,KK
      INTEGER ATOM,CAT,A,K,KK
      INTEGER FIND,STATUS,I
C-----------------------------------------------------------------
C
C CLOSE OLT OPEN TEMPLATE
      IF(KWDNAM.EQ.0) GOTO 50
      M=FIND(TNAME,MAXM,KWDNAM,STATUS)
      IF(STATUS.EQ.1) GOTO 24
      CALL ERROR(1,512)
      RETURN
   24 CALL WRITMS(1,MAXM,511,KWDNAM)
      MF=T(M,FAM)
      IF(MF.GT.0) GOTO 26
      RETURN
   26 MF=T(MF,NXT)
      IF(M.NE.MF) GOTO 50
C IT IS THE FIRST KWD
   30 M=T(M,NXT)
      IF(M.EQ.0) GOTO 50
      DO 34 I=6,20,2
      IF(TABIND(I).EQ.0) GOTO 40
      IF(TABIND(I).EQ.TNAME(M)) GOTO 30
   34 CONTINUE
C
   40 CALL WRITMS(1,MAXM,511,TNAME(M))
      GOTO 30
```

```
C
C OPEN NEW TEMPLATE IF ASKED TO
 90 IF(KEY.EQ.1) RETURN
    CALL NEXTA
    IF(K.EQ.EOS) RETURN
    IF(K.EQ.E.OR.K.EQ.KWD) GOTO 100
    CALL ERROR(A,510)
    RETURN
100 DO 120 KWI=6,20,2
    IF(TABIND(KWI).EQ.0) GOTO 140
    IF(TABIND(KWI).EQ.ATOM(A)) GOTO 130
120 CONTINUE
    CALL ERROR(A,511)
    RETURN
C THIS TEMPLATE EXISTS
130 CALL RESERPS(1,[illegible],511,ATOM(A))
C [illegible]
[illegible] ATOM(A)
    PRINT 141,KWDNAM
141 FORMAT(1H1,5X,*TEMPLATE *,A10,8H IS OPEN)
    RETURN
    END
*************************************************************************


    SUBROUTINE DISPLAY
C TO DISPLAY TEMPLATE CONTENTS
C
    COMMON/ALLCAT2/TYP,NXT,BRO,SON,NPR,FAM,RES,MVAL,MNO,LIM,VALUE
   1,STAT,TYPE, NAME, MXDICT,MXSPNAM,MXTNAM,MXTCEL,MXPCEL
   2,EQL,STRING,FNUM,INUM,OPER,KWD,DESCON, KWCCL,ITNAME,CLNAME
   4,EXNAME,QNAME,EXEC,REQU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TC,CE
   5,OPSTYP,ULTYP,LLTYP, EOV,EQVCAT
    INTEGER
   1 EQL,STRING,FNUM,INUM,OPER,KWD,DESCON, KWCCL,ITNAME,CLNAME
   2,EXNAME,QNAME,EXEC,REQU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TC,CF
   3,OPSTYP,ULTYP,LLTYP, EOV,EQVCAT
    COMMON/ALLCAT3/DICNAM,TEMNAM,ITCAT,UL,LL,FOR,SETMARK
    COMMON /UDICT/UDNAME(63),UDCAT(63),MAXUD,UD,UDSTAT
   1,KWDNAM,KWI,TABIND(21)
    INTEGER DICNAM,TEMNAM,ITCAT,UL,LL,FOR,SETMARK
C
    COMMON /TEMPL/MAXM,MAXV,TNAME(109),TCELL(400)
    INTEGER ANAME(7),F,T,TNAME,V,VR,VL,MCAT(9)
    DATA MCAT/1HS,1HC,1HK,1HO,1HC,1HI,1HF,1HA,1HP/
C -----------------------------------------------
C
    PRINT 3,KWDNAM
  3 FORMAT(1H1,5X,*TEMPLATE *,A10/)
    NL=2
    F=0
    F=NEXTF(F)
    IF(F.GT.0) GOTO 10
```

```
I        CALL ERROR(2,501)
I        RETURN
IC CLASS HEADER
I    10 ANAME(1)=ANAME(2)=1H
I        K=T(F,SON)
I        IF(K.GT.0) ANAME(1)=TNAME(K)
I        K=T(F,BRO)
I        IF(K.GT.0) ANAME(2)=TNAME(K)
I        K=NN=T(F,NPR)
I        IF(K.GT.MAXM) NN= -T(K,2)
I        PRINT 11,F, TNAME(F),NN,ANAME(1),ANAME(2)
I    11 FORMAT(1H0,*C = *,I3,1X,A10,7H NPR = ,I4,5H S = ,A10,
I       1* B = *,A10)
I        NL=NL+2
I        IF(NL.GT.56) PRINT 3,KWDNAM
I        IF(NL.GT.56) NL=2
IC
IC MEMBERS
I        ANAME(1)=ANAME(2)=1H
I        NN=0
I        M=T(F,NXT)
I        IF(M.EQ.0) GOTO 1000
I   200 KT=T(M,TYP)
I        IF(KT.EQ.DESCON.OR.KT.EQ.KWD) NN=T(M,MVAL)
I        MTYP=1H*
I        IF(KT.GT.0.AND.KT.LT.10) MTYP=MCAT(KT)
I        PRINT 21,M,MTYP,TNAME(M),NN
I    21 FORMAT(1H ,4X,*M = *,I3,1X,A1,1X,A10,3X,*IV =*,I3)
I        MR=T(M,RES)
I        IF(MR.EQ.0) GOTO 300
IC RESTRICTIONS
I        DO 220 I=1,7
I        KR=T(MR,I+1)
I        IF(KR.EQ.0) GOTO 230
I        ANAME(I)=TNAME(KR)
I   220 CONTINUE
I   230 PRINT 23, (ANAME(K),K=1,I)
I    23 FORMAT(1H+,35X,*FOR *,7A10)
I        DO 232 I=1,7
I   232 ANAME(I)=1H
IC
IC ITEM VALUES
I   300 NL=NL+1
I        IF(NL.GT.56) PRINT 3,KWDNAM
I        IF(NL.GT.56) NL=2
I        IF(KT.EQ.DESCON.OR.KT.EQ.KWD) GOTO 500
I        V=T(M,MVAL)
I        IF(V.LT.MAXV) GOTO 500
I310     IVAL=T(V,VALUE)
I        IF(KT.EQ.FNUM) GOTO 320
I        IF(KT.EQ.STRING) GOTO 330
I        PRINT 311,IVAL
I   311 FORMAT(1H ,8X,*V = *,I10)
I        GOTO 340
```

```
      PRINT 321 ,IVAL
  321 FORMAT(1H ,8X,*V = *,F10.3)
      GOTO 348
      PRINT 321,IVAL
      FORMAT(1H ,8X,*V = *,A10)
      VR=T(V,PRE)
      IF(VR.EQ.0) GOTO 350
C VALUE RESTRICTION
      PRINT 341, TRANE(VR)
      FORMAT(1H+,24X,*FOR *,A10)
C VALUE LIMITS
  350 VL=T(V,LIM)
      IF(VL.EQ.0) GOTO 400
      LTYP=SHLL
      IF(T(VL,PRE).EQ.ULTYP) LTYP=SHUL
      IVAL=T(VL,VALUE)
      IF(LTYP.EQ.FNUM) GOTO 352
      IF(LTYP.EQ.STRING) GOTO 354
      PRINT 351,LTYP,IVAL
  351 FORMAT(1H+,38X,A5,I10)
      GOTO 356
      PRINT 353,LTYP,IVAL
      FORMAT(1H+,38X,A5,F10.3)
      GOTO 356
      PRINT 355,LTYP,IVAL
      FORMAT(1H+,38X,A5,A10)
      VL=T(VL,LIM)
      IF(VL.EQ.0) GOTO 400
      LTYP=SHLL
      IF(T(VL,PRE).EQ.ULTYP) LTYP=SHUL
      IVAL=T(VL,VALUE)
      IF(LTYP.EQ.FNUM) GOTO 362
      IF(LTYP.EQ.STRING) GOTO 364
      PRINT 361,LTYP,IVAL
      FORMAT(1H+,54X,A5,I10)
      GOTO 400
      PRINT 363,LTYP,IVAL
      FORMAT(1H+,54X,A5,F10.3)
      GOTO 400
      PRINT 365,LTYP,IVAL
  365 FORMAT(1H+,54X,A5,A10)
C GET NEXT VALUE
  400 NL=NL+1
      IF(NL.GT.56) PRINT 3,CHONAM
      IF(NL.GT.56) NL=2
      V=T(V,NXT)
      IF(V.GE.MAXV) GOTO 318
      PRINT 401
  401 FORMAT(1H )
      NL=NL+1
```

```
C
C GET NEXT MEMBER
  500 M=T(M,NXT)
      IF(M.GT.0) GOTO 200
C
C GET NEXT CLASS
 1000 F=NEXTF(F)
      IF(F.GT.0) GOTO 10
      RETURN
      END
C*****************************************************************
C
C
C     SUBROUTINE DUMPTC(PART)
C TO DUMP TCELLS, 0=BOTH PARTS, 1=M-CELLS, 2=V-CELLS
C       CALLED BY ULSETUP
      COMMON/ULDAT2/ DUMMY(17),MXTCEL
      INTEGER FIELD(5),T,PART
      COMMON /TEMPL/MAXM,MAXV,TNAME(109),TCELL(400)
C
      PRINT 21
   21 FORMAT(1H )
      IF(PART.EQ.2) GOTO 120
      DO 100 M=1,MAXM
      DO 30 J=1,5
   30 FIELD(J) = T(M,J)
      PRINT 31, M,TNAME(M),(FIELD(J),J=1,5)
   31 FORMAT(* M= *,I3,2X,A8,5I4)
  100 CONTINUE
      IF(PART.EQ.1) RETURN
      PRINT 21
C
120   IF(MAXV.GT.MXTCEL) RETURN
      DO 200 M= MAXV,MXTCEL
      DO 130 J=1,5
  130 FIELD(J) = T(M,J)
      PRINT 131, M,FIELD
  131 FORMAT(* V= *,I3,10X,5I4)
  200 CONTINUE
      RETURN
      END
C*****************************************************************
C
C
C     SUBROUTINE DVALUE
C TO DEFINE DEFAULT VALUES AND LIMITS OF ITEM-NAMES
C       CALLED BY ULSETUP
C
      COMMON/ULDAT2/TYP,NXT,BRO,SCN,NPR,FAM,RES,MVAL,PNO,LIM,VALUE
     2,STAT,TYP2, KNAME, MXCICT,MXSFNAM,MXTNAM,MXTCEL,MXPCEL
     3,EOS,STRING,FNUM,INUM,OPER,KWC,DESCON, KWDCL,ITNAME,CLNAME
     4,EXNAME,ONAME,EXEC,RECU,LPAREN,RPAREN,MINUS,ALP,EQUAL,TO,CF
```

```
      S,RESTYP,ULTYP,LLTYP, EOV,ECVCAT                                  I260   IF(T(V,TYP).EQ.RESTYP) GOTO 280
       INTEGER*                                                         I      NXV= T(V,NXT)
     1 EOS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWCCL,ITNAME,CLNAME        I      IF(NXV.EQ.0) GOTO 270
     2,EXNAME,CNAME,EXEC,REQU,LPAREN,RPAREN,MINUS,ADD,EQUAL,TC,CF       I      V=NXV
     3,RESTYP,ULTYP,LLTYP, EOV,ECVCAT                                   I      GOTO 260
      COMMON/ILCAT3/DICHAN,TEMHAN,ITCAT,UL,LL,FOR,SETMARK               IC ADD NEW VALUE CELL
      INTEGER DICHAN,TEMHAN,ITCAT,UL,LL,FOR,SETMARK                     I270   CALL NEXTV(NV)
C                                                                       I      CALL TSET(V,NXT,NV)
      INTEGER FIND,STATUS,T,V,VCAT,VL,REM                               I      V=NV
      COMMON /ICKED/ATOM(63),CAT(63),A,MAXA,K,KK                        IC PUT VALUE
      INTEGER ATOM,CAT,A,K,KK                                           IC   POSSIBLY CONVERT I, F
      DIMENSION FATOM(63)                                               I  280 MTYP=T(M,TYP)
      EQUIVALENCE (ATOM,FATOM)                                          I      IF(PTYP.EQ.INUM.AND.K.EQ.FNUM) ATOM(A)=FATOM(A)
      COMMON /TEMPL/MAXM,MAXV,TNAME(100),TCELL(400)                     I      IF(PTYP.EQ.FNUM.AND.K.EQ.INUM) FATOM(A)=ATOM(A)
C ------------------------------------------                           I      CALL TSET(V,VALUE,ATOM(A))
C                                                                       I      VCAT=MOD(CAT(A),100)
      LBV=RBV=LEP=V=0                                                   I      CALL TSET(V,TYP,VCAT)
C SYNTAX LOOPS TO WHICH VALUE APPLIES                                   I      IF(LBV.EQ.0) LBV=V
      CALL NEXTA                                                        I      GOTO 200
      IF(CAT(A).EQ.OF) CALL NEXTA                                       IC
      M=FIND(TNAME,MAXM,ATOM(A),STATUS)                                 IC ---------- LL AND UL ----------
      IF(STATUS.EQ.0) GOTO 1010                                         IC LIMITS
      MTYP=T(M,TYP)                                                     I300   CALL NEXTA
      IF(MTYP.EQ.KWDCL.OR.MTYP.EQ.CLNAME) GOTO 1180                     I      IF(K.EQ.ENUM.OR.K.EQ.STRING) GOTO 310
      IF(MTYP.EQ.OFSCON.OR.MTYP.EQ.KWD) GOTO 1180                       I      IF(K.EQ.INUM) GOTO 306
C                                                                       I      IF(CAT(A).EQ.LL) GOTO 302
C  FIND THE VALUE OR LIMIT                                              I      IF(CAT(A).EQ.UL) GOTO 304
200   CALL NEXTA                                                        I      IF(K.EQ.EOS) RETURN
      IF(CAT(A).EQ.EQUAL) CALL NEXTA                                    I      IF(CAT(A).EQ.EQUAL) GOTO 300
      IF(K.EQ.ENUM.OR.K.EQ.STRING) GOTO 216                             I      GOTO 1100
      IF(K.EQ.INUM) GOTO 206                                            IC LOWER LIM
      IF(CAT(A).EQ.LL) GOTO 302                                         I302   LTYP=LLTYP
      IF(CAT(A).EQ.UL) GOTO 304                                         I      IF(LBV.GT.0) V=LBV
      IF(CAT(A).EQ.FOR) GOTO 500                                        I      GOTO 300
      IF(K.EQ.EOS) RETURN                                               IC UPPER LIM
      GOTO 1180                                                         I304   LTYP=LLTYP
C ---------- VALUE ----------                                          I      IF(LBV.GT.0) V=LBV
C INTEGER, REPEAT BRACKET MAY FOLLOW                                    I      GOTO 300
206   IF(CAT(A+1).NE.LPAREN) GOTO 210                                   IC INTEGER, REPEAT BRACKET MAY FOLLOW
      CALL NEXTA                                                        I  306 IF(CAT(A+1).NE.LPAREN) GOTO 310
      CALL REPEAT                                                       I      CALL NEXTA
C                                                                       I      CALL REPEAT
C DEFAULT VALUES                                                        IC LIMIT VALUE
  210 IF(LBV.EQ.0) GOTO 220                                             I  310 IF(V.GT.0) GOTO 330
      V=LBV                                                             I      V=T(M,MVAL)
      GOTO 260                                                          I      IF(V.GT.0) GOTO 330
220   V=T(P,PVAL)                                                       IC GET FIRST V-CELL
      IF(V.GT.0) GOTO 260                                               I      CALL NEXTV(V)
C FIRST VALUE, FORM M-V LINK                                            I      CALL TSET(M,MVAL,V)
      CALL NEXTV(V)                                                     I      IF(T(P,TYP).EQ.0) CALL TSET(M,TYP,K)
      CALL TSET(M,PVAL,V)                                               I      IF(LBV.EQ.0) LBV=V
      IF(T(P,TYP).EQ.0) CALL TSET(M,TYP,K)                              I      GOTO 300
      GOTO 280                                                          IC
C FURTHER VALUES                                                        IC VALUE CELL EXISTS
```

```
      LV= T(V,LIM)                                          I       LBV=0
      IF(LV.EQ.0) GOTO 700                                  I       GOTO 200
C     EXIT CELL EXISTS                                      IC CLASS-NAME RESTRICTION
      IF(T(LV,RES).NE.LTYP.AND.T(LV,LIM).EQ.0) GOTO 370     I720    MR= T(MR,NXT)
C                                                           I       IF(MR.EQ.0) GOTO 730
C     LOOK AT NEXT V-CELL                                   I       CALL QUALV(MR,M,LBV)
      LBV=T(V,NXT)                                          I       GOTO 720
      IF(LBV.EQ.0) GOTO 350                                 I  730  IF(LBF.NE.0) GOTO 500
      V=LBV                                                 I       LBV=0
      GOTO 330                                              I       GOTO 200
C     NEW V-CELL                                            IC LEFT PAREN
  350 CALL GETV(LBV)                                        I690    IF(LBF.LT.0) CALL ERROR(A,506)
      CALL TSET(LV,NXT,BV)                                  I       LBF=-1
      V=LBV                                                 I       GOTO 500
      IF(LBV.EQ.0) LBV=V                                    IC RIGHT PAREN
      GOTO 300                                              I680    IF(LBF.EQ.0) CALL ERROR(A,504)
C                                                           I       LBF=LEV=REM=0
C     NOT LIMIT-CELL                                        I       GOTO 200
  370 CALL GETV(VL)                                         IC ERRORS
      CALL TSET(LV,LIM,VL)                                  I1010   CALL ERROR(A,505)
      GOTO 390                                              I       RETURN
C     NEW LIMIT CELL                                        I1100   CALL ERROR(A,509)
  380 CALL NEXTV(VL)                                        I       RETURN
      CALL TSET(VL,LIM,VL)                                  I       END
      IF(T(V,TYP).EQ.0) CALL TSET(V,TYP,K)                  I**********
C                                                           I
C     CONVERT F,I,IF NECESSARY                              I
  390 TYPE=T(F,TYP)                                         I
      IF(TYP.EQ.INUM.AND.K.EQ.FNUM) ATOM(A)=FATOM(A)        I       SUBROUTINE INSERT(F,FS,LINK)
      IF(TYP.EQ.FNUM.AND.K.EQ.INUM) FATOM(A)=ATOM(A)        IC TO APPEND A CLASS.  CALLED BY CLINK
      CALL TSET(VL,VALUE,ATOM(A))                           IC F IS INDEX OF CLASS TO WHICH CLASS FS IS APPENDED
      VCAT=MOD(CAT(A),100)                                  IC
      CALL TSET(VL,TYP,VCAT)                                I       COMMON/ULDAT2/TYP,NXT,BRO,SON,NPR,FAM,RES,MVAL,PNO,LIM,VALUE
      CALL TSET(VL,RES,LTYP)                                I      2,STAT,TYP2, KNAME, MXDICT,MXSPNAM,MXTNAM,MXTCEL,MXPCEL
      GOTO 380                                              I      3,ECS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWDCL,ITNAME,CLNAME
C                                                           I      4,EXNAME,CNAME,LXEC,REGU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TO,OF
C     ---------- FOR ----------                            I      5,RESTYP,ULTYP,LLTYP, EOV,EOVCAT
C     APPLY RESTRICTION                                     I       INTEGER
  400 CALL NEXTP                                            I      1 ECS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWDCL,ITNAME,CLNAME
      IF(P.EQ.MPO) K=0                                      I      2,EXNAME,CNAME,LXEC,REGU,LPAREN,RPAREN,MINUS,AOP,EQUAL,TO,OF
      IF(P.EQ.0) GOTO 600                                   I      3,RESTYP,ULTYP,LLTYP, EOV,EOVCAT
      IF(CAT(A).EQ.LPAREN) GOTO 600                         I       COMMON/ULDAT3/CICMAN,IEMMAN,ITCAT,UL,LL,FOR,SETMARK
      IF(CAT(A).EQ.RPAREN) GOTO 600                         I       INTEGER CICMAN,IEMMAN,ITCAT,UL,LL,FOR,SETMARK
      IF(K.EQ.ECS) RETURN                                   IC
      CALL ERROR(A,502)                                     I       INTEGER F,FS,FB,T,FINC,STATUS,SONNY,BROTH,V
      RETURN                                                I       COMMON /TOKEN/ATOM(63),CAT(63),A,MAXA,K,KK
C FIND RESTRICTION                                          I       INTEGER ATOM,CAT,A,K,KK
  600 MR= FIND(INAME,MAXM,ATOM(A),STATUS)                   I       COMMON /TEMPL/MAXM,MAXV,TAAP(109),IC(400)
      IF(STATUS.EQ.1) GOTO 610                              IC ----------
      CALL ERROR(A,506)                                     IC
      RETURN                                                I       FS=INSERT(LINK,CLNAME,F)
  610 IF(T(MR,TYP)/10.EQ.1) GOTO 720                        I       IF(FS.GT.0) GOTO 40
      CALL QUALV(MR,M,LBV)                                  I       CALL ERROR(A,507)
      IF(LBF.NE.0) GOTO 500                                 I       RETURN
```

```
C SET TYPE= CLASS
50    CALL TSET(FS,TYP,CLNAME)
      SONAY= T(F,SCN)
      IF(SCNNY.GT.0) GOTO 100
C ADD THE SON-LINK
      CALL TSET(F,SON,FS)
      GOTO 200
C ADD A BROTHER-LINK
100   FB=SCNNY
110   BROTH= T(FB,BRO)
      IF(BROTH.EQ.0) GOTO 120
      FB=BROTH
      GOTO 110
120   CALL TSET(FB,BRO,FS)
C SET NO OF ASSC PARAMS, DEFAULT IS NPR=1
200   NPRC=1
      CALL NEXTA
      IF(ATOM(A).NE.SETMARK) GOTO 350
C IT IS =
      NPRC=0
      CALL NEXTA
      IF(K.LE.KWD) K=0
      IF(K.NE.INUM) GOTO 310
C NUMERIC
      NPRC= ATOM(A)
      CALL NEXTA
      GOTO 350
C NO OF PARAMS OF ANOTHER CLASS PAYEE
310   IF(K.NE.0) GOTO 350
      M=FIND(INAME,MAXM,ATOM(A),STATUS)
      IF(STATUS.EQ.0.OR.T(MF,TYP)/10.NE.1) GOTO 350
      CALL NEXTA(V)
      NPRC=V
      CALL TSET(V,TYP,RESTYP)
      CALL TSET(V,2,MF)
      CALL NEXTA
C SET NPR.
350   CALL TSET(FS,NPR,NPRC)
      IF(K.EC.KWD) K=0
      RETURN
      END
```

```
      FUNCTION INSERT(INAME,ITYPE,ACPTR)
C TO INSERT A CLASS-NAME OR MEMBER-NAME IN TNAME AND IN USER DICT
C AND TO ADJUST TCELLS AND POINTERS
C       CALLED BY CLASS, CLINK, INSERF
C
      COMMON/LLCAT2/TYP,NXT,BRO,SON,NPR,FAM,RES,MVAL,PNO,LIM,VALUE
     2,STAT,TYP2, MNAME, MXDICT,MXSPNAM,MXTNAM,MXTCEL,MXPCEL
     3,ECS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWOCL,ITNAME,CLNAME
     4,EXNAME,CNAME,EXEC,RECU,LPAREN,RPAREN,MINUS,ACP,EQUAL,TO,OF
```

```
      INTEGER ECS,STRING,FNUM,INUM,OPER,KWD,DESCON, KWOCL,ITNAME,CLNAME
     1,RESTYP,ULTYP,LLTYP, EOV,EOVCAT
      INTEGER
     2,EXNAME,CNAME,EXEC,RECU,LPAREN,RPAREN,MINUS,AOP,_QUAL,TO,OF
     3,RESTYP,ULTYP,LLTYP, EOV,EOVCAT
C
      COMMON /TEMPL/MAXM,MAXV,TNAME(100),TCELL(400)
      INTEGER FIND,STATUS,PTR,T,TNAME,TCELL,ARG(2),ACPTR
      COMMON /UDICT/UDNAME(63),UDCAT(63),MAXUD,UD,UDSTAT
     1,KWDNAM,KWI,TABIND(21)
      INTEGER UDSTAT
C ---------------------------------------------------
C
      IF(MAXM.LT.MAXV.AND.MAXM.LT.MXTNAM) GOTO 10
      CALL ERROR(MAXV,1001)
      RETURN
C FIND INDEX FOR THIS "NAME"
10    M=0
      M=FIND(TNAME,MAXM,INAME,STATUS)
      IF(STATUS.EQ.1) GOTO 200
      IF(M.GT.MAXM) GOTO 40
C MAKE ROOM
      DO 30 MM=M,MAXM
      K=MAXM + M - MM
30    TNAME(K+1) = TNAME(K)
40    MAXM= MAXM+1
      IF(ITYPE.EQ.KWD.AND.KWDNAM.EQ.0) KWDNAM=INAME
C INSERT NAME
      TNAME(M) = INAME
      ARG(1)=INAME
      IF(ITYPE.EQ.0) ITYPE=ITNAME
      ARG(2)=ITYPE*10
      IF(ARG(2).LT.100) ARG(2)=ARG(2)*10
      UD=INSERX(UDNAME,MXDICT,MAXUD,2,ARG)
      UDSTAT=1
      IF(M.EQ.MAXM) GOTO 200
C PUSH DOWN M-CELLS
      MAXM1=MAXM-1
      DO 120 MM=M,MAXM1
      K= 2*(MAXM1 + M - MM)
      TCELL(K+2) = TCELL(K)
120   TCELL(K+1) = TCELL(K-1)
C ZERO OUT TCELL(M)
      K=2*M
      TCELL(K)=TCELL(K-1)=0
C ADJUST POINTERS
      DO 150 K=1,MAXM
      DO 140 I=2,3
      PTR=T(K,I)
      IF(PTR.LT.M) GOTO 140
      PTR=PTR+1
      CALL TSET(K,I,PTR)
140   CONTINUE
```

```
      IF(T(K,TYP)/10.NE.1) GOTO 150
      PTR=T(K,4)
      IF(PTR.LT.M) GOTO 158
      PTR=PTR+1
      CALL TSET(K,4,PTR)
  150 CONTINUE
C ADJUST ACTIVE POINTER
      IF(ACPTR.EQ.0) GOTO 200
      IF(ACPTR.GE.M) ACPTR=ACPTR+1
C ADJUST RESTRICTION POINTERS
      IF(MAXV.GT.MXTCEL) GOTO 200
      DO 170 K=MAXV,MXTCEL
      IF(T(K,TYP).NE.RESTYP) GOTO 164
      DO 160 I=2,8
      PTR=T(K,I)
      IF(PTR.LT.M) GOTO 160
      PTR=PTR+1
      CALL TSET(K,I,PTR)
  160 CONTINUE
      GOTO 170
  164 PTR=T(K,RES)
      IF(PTR.LT.M.OR.PTR.GT.MAXM) GOTO 170
      CALL TSET(K,RES,PTR+1)
  170 CONTINUE
C RETURN INDEX
  200 INSERT=F
      RETURN
      END
*****************************************************************

      FUNCTION INSERX(T,NRMAX,NR,NC,ARG)
C TO INSERT ARG(NC) INTO TABLE T(NR,NC) AT POSITION K
C NRMAX = MAX NC OF ROWS (A CONSTANT)
C NR = CURRENT NO OF ROWS, NC = CURRENT NO OF COLUMNS
C       CALLED BY DBUILD, UDBUILD, INSERT
      INTEGER T(1),ARG(1),FIND,STATUS
C
      IF(NR.LT.NRMAX) GOTO 10
      CALL ERROR(NF,1002)
      RETURN
C
   10 K=0
      K=FIND(T,NR,ARG(1),STATUS)
      IF(STATUS.EQ.1) GOTO 300
C NOT FOUND
      IF(K.GT.NR) GOTO 220
C MAKE ROOM FOR ARG
      DO 210 PP=K,NR
      NR=NR+K-PP
      DO 210 MC=1,NC
      M= NRMAX*(MC-1) + NR
  210 T(M+1)=T(M)
  220 NR=NR+1
C INSERT OR REPLACE ARG
  300 DO 310 MC=1,NC
      M=NRMAX*(MC-1)+K
      IF(ARG(MC).NE.0) T(M)=ARG(MC)
  310 CONTINUE
      INSERX=K
      RETURN
      END
*****************************************************************

      SUBROUTINE NEXTV(V)
C TO ALLOCATE NEXT V-CELL
C         CALLED BY DVALUE,INSERF, QUALR, QUALV
      COMMON /TEMPL/MAXM,MAXV,TNAME(109),TCELL(400)
      INTEGER V
      MAXV= MAXV-1
      V=MAXV
      IF(MAXM.LT.MAXV) RETURN
      CALL ERROR(V,1002)
      RETURN
      END
*****************************************************************

      SUBROUTINE QUALR(MR,LBA,RBA)
C TO APPLY RESTRICTION MR TO MEMBERS FROM LBA TO RBA
C RA IS CURRENT PTR TO ATOMS
C         CALLED BY CLASS
      COMMON/ULOAT2/TYP,NXT,BRO,SON,NPR,FAM,RES,MVAL,PNO,LIM,VALUE
     2,DUMMY(29),RESTYP
      INTEGER RBA,T,V,M,STATUS,FIND,RA
      COMMON /TOKEN/ATOM(63),CAT(63),A,MAXA,K,KK
      INTEGER ATOM,CAT,A,K,KK
      COMMON /TEMPL/MAXM,MAXV,TNAME(109),TCELL(400)
C ---------------------------------------------
C
      IF(LBA.EQ.0) LBA=RBA
      RA=LBA
  100 M=FIND(TNAME,MAXM,ATOM(RA),STATUS)
      IF(STATUS.EQ.1) GOTO 110
      CALL ERROR(RA,506)
      RETURN
  110 V= T(M,RES)
      IF(V.GT.0) GOTO 200
C FIRST RESTRICTION
      CALL NEXTV(V)
      CALL TSET(M,RES,V)
      CALL TSET(V,TYP,RESTYP)
      CALL TSET(V,2,MR)
      GOTO 500
```

```
C 2ND TO 7TH RESTRICTIONS
288   DO 298 I=3,8
      IF( T(V,I).EQ.0) GOTO 388
298   CONTINUE
C FULL
      CALL ERROR(R4,588)
      GOTO 588
C ADD NEW RESTRICTION
388   CALL TSET(V,I,MR)
C LOOK OVER PEPERS
588   IF(RA.GE.RRA) RETURN
      RA=RA+1
      GOTO 188
      END
*****************************************************************
      SUBROUTINE QUALV(MR,M,LBV)
C TO APPLY VALUE RESTRICTION MR TO M
C       CALLED BY QVALUE
      COMMON/LLCAT2/TYP,NXT,BRO,SON,NPR,FAP,RES,MVAL,PNO,LIM,VALUE
     2, DUMMY(29),RESTYP
      INTEGER T,V
C
      V=T(P,MVAL)
      IF(V.GT.0) GOTO 288
C FIRST VALUE RESTP
      CALL NEXTV(NV)
      CALL TSET(M,MVAL,NV)
      GOTO 388
C FURTHER VALUE RESTP
288   NXV=T(V,NXT)
      IF(NXV.EC.0) GOTO 258
      V=NXV
      GOTO 288
258   CALL NEXTV(NV)
      CALL TSET(V,NXT,NV)
C SET RESTRICTION
388   CALL TSET(NV,RES,MR)
      CALL TSET(NV,TYP,RESTYP)
      IF(LBV.EC.0) LBV=NV
      RETURN
      END
*****************************************************************
      SUBROUTINE STARCLC
C TO CLOSE CCT TABLES
C       CALLED BY ULSETUP
C       USES DIRECT ACCESS READ/WRITE
C
      COMMON /ULDAT2/ DUMMY(116),MXTNAM,MXTCEL
      COMMON /ULDAT3/ DUMMY3(7),MAXDS,SDNAME(22),SDCAT(22)
      INTEGER SDNAME,SDCAT

      COMMON /DICT/DNAME(63),DCAT(63),MAXD,D,DSTAT
      INTEGER DNAME,DCAT,MAXD,D,DSTAT
      COMMON /UDICT/UDNAME(63),UDCAT(63),MAXUD,UD,UDSTAT
     1,KWDNAM,KWI,TABIND(21)
      INTEGER UDNAME,UDCAT,UD,UDSTAT,TABIND
      COMMON /TEMPL/MAXM,MAXV,TNAME(109),TCELL(400)
C ---------------------------------------------------

      IF(DSTAT.EQ.1) CALL WRITMS(1,DNAME,127,SLSDICT)
   20 CONTINUE
      IF(UDSTAT.EQ.0) GOTO 30
      PRINT 21
   21 FORMAT(1H1,6X,*USER DICTIONARY*/)
      PRINT 13,(UD,UDNAME(UD),UDCAT(UD),UD=1,MAXUD)
   13 FORMAT(1X,I4,2X,A8,I5)
      CALL WRITMS(1,UDNAME,127,SLUDICT)

130   IF(KWDNAM.EQ.0) GOTO 40
      CALL DEFINE(1)
   40 CONTINUE
      RETURN

C ---------------------------------------------------
      ENTRY STAROP
C OPEN TABLES
C       CALLED BY ULSETUP
      CALL OPENMS(1,TABIND,21,1)
      OSTAT=UDSTAT=KWDNAM=0
      MAXUD=KWI=MAXM=0
      MAXV=MXTCEL+1
      DO 120 I=1,MXTNAM
  120 TNAME(I)=0
      MXT=MXTCEL*2
      DO 130 I=1,MXT
  130 TCELL(I)=0
      PRINT 101
  101 FORMAT(1H1)
C READ DICTIONARIES
      IF(TABIND(2).EQ.SLSDICT) GOTO 150
C SYS DICT DOES NOT EXIST YET, GET S DEFAULT SDICT
      MAXD=MAXDS
      DO 140 I=1,MAXD
      DNAME(I)=SDNAME(I)
  140 DCAT(I)=SDCAT(I)
      DSTAT=1
      GOTO 200
C READ EXISTING SYS DICT
  150 CALL READMS(1,DNAME,127,SLSDICT)
C READ USER DICT
200   IF(TABIND(4).EQ.SLUDICT) CALL READMS(1,UDNAME,127,SLUDICT)
      RETURN
```

```
      END
C*****************************************************************
C
C
C
      FUNCTION ISET(NX,IPOS,ITEM)
C     SET ITEM AT BYTE I OF CELL N OF TCELL
C     ITEM IS 2 WORDS LONG
      COMMON /BD/ CLASS,FLINK,OVALUE,INSERF,INSERT,GLALR,CUALV
      COMMON /TEMPL/MAXM,MAXV,TNAME(100),TCELL(480)
      INTEGER TCELL,MASK(4)
      DATA MASK/#FF000000,#00FF0000,#0000FF00,#000000FF/
C
      ENTRY
      IVEC=1
      IF(IPOS.LE.4) GOTO 10
      IVEC=2
      IPOS=IPOS-4
   10 IF(IPOS.EQ.1) GOTO 20
      IPOS=5-IPOS
C     SHIFT N BYTES
      DO 30 I=1,IPOS
      ITEM=ITEM/256
   30 CONTINUE
C     PUT THE WHOLE 2ND WORD
   20 TCELL(N)=ITEM
      RETURN
C     WORK BYTE SPEC
      IF(CELL.EQ.MASK(I)) GOTO 40
      RETURN
      END
```