

AD-A116 503

ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA

F/G 9/2

VIRTUAL INFORMATION FACILITY OF THE INFOPLEX SOFTWARE TEST VEHIC--ETC(U)

MAY 82 P LU

N00039-81-C-0663

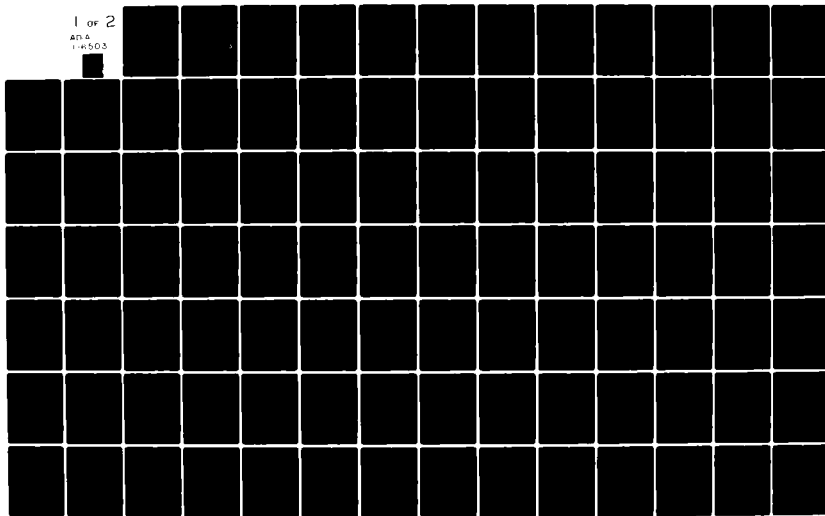
UNCLASSIFIED

M010-8205-11

NL

1 of 2

AD-A
1-850-3

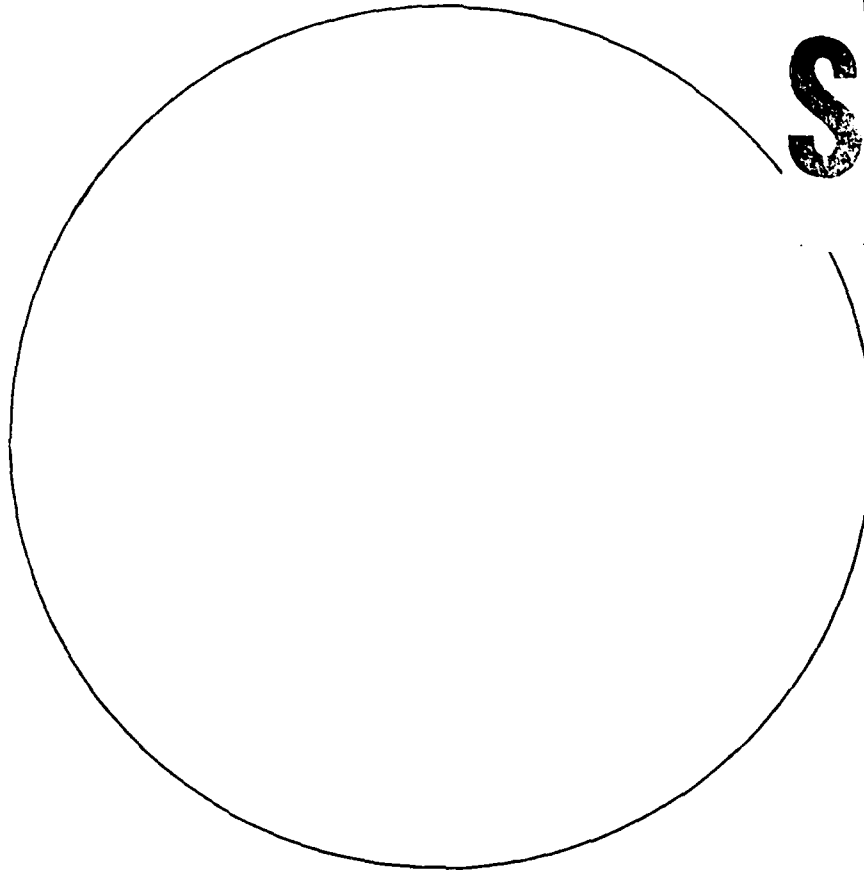




12

AD A116503

DTIC
ELECT
S JUL 7 1982
H



DTIC FILE COPY

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution unlimited

Center for Information Systems Research

Massachusetts Institute of Technology
Sloan School of Management
77 Massachusetts Avenue
Cambridge, Massachusetts 02139

Contract Number N00039-81-0663 (MIT # 91445)
Internal Report Number M010-8205-11
Deliverable Number 6

12

VIRTUAL INFORMATION FACILITY
OF THE INFOPLEX SOFTWARE TEST VEHICLE
(PART II)

Technical Report #11

BY
Peter Lu
May, 1982

NAVY
JUL 7 1982
D

Principal Investigator:
Professor Stuart E. Madnick

Prepared for:
Naval Electronics Systems Command
Washington, D.C.

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Technical Report #11	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Virtual Information Facility of the INFOPLEX Software Test Vehicle (Part II)		5. TYPE OF REPORT & PERIOD COVERED
7. AUTHOR(s) Peter Lu		6. PERFORMING ORG. REPORT NUMBER M010-8205-11
9. PERFORMING ORGANIZATION NAME AND ADDRESS Sloan School of Management, MIT 50 Memorial Drive, Cambridge, MA 02139		8. CONTRACT OR GRANT NUMBER(s) N0039-81-C-0663
11. CONTROLLING OFFICE NAME AND ADDRESS		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office)		12. REPORT DATE May 1982
		13. NUMBER OF PAGES 144
		15. SECURITY CLASS. (of this report) unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) database computer, database management system, Software Test Vehicle, Virtual Information hierarchical system		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report describes the software design and implementation of the rear- end for the Virtual Information Facility of the INFOPLEX database computer. It is part of a major effort to develop a software simulation, STV, for the underlying architecture of INFOPLEX. The virtual information facility is a single level of operations situated within the Functional Hierarchy. It supports the use of virtual information, a virtual entity based on procedural relationships and derivations from		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

physically recorded data. Upon completion, this facility will be integrated within the current implementation of the STV for the INFOPLEX Functional Hierarchy which lacks the support for virtual information processing.

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Avail and/or	
Dist	Special
A	



Virtual Information Facility
of the INFOPLEX Software Test Vehicle

by

PETER LU

Submitted to the Department of Electrical
Engineering and Computer Science in May,
1982, in partial fulfillment of the
requirements for the degree of
Bachelor of Science

Abstract

This thesis is a software design and implementation of the rear-end for the Virtual Information Facility of the INFOPLEX data base computer. It is part of a major effort to develop a software simulation, so called a Software Test Vehicle, STV , for the underlying architecture of INFOPLEX.

INFOPLEX is a hierarchical architecture for data base computers, based on functional decomposition of data base operations. It is a current research project of the Information Systems Group at M.I.T.'s Sloan School of Management. Within the INFOPLEX architecture, a functional hierarchy of information management functions is built on top of a storage hierarchy of information storage functions. These two independent hierarchies are further divided into many sub-levels, each of which is devoted to a more specific function of data base activities.

The virtual information facility is a single level of operations situated within the functional hierarchy. It supports the use of virtual information, a virtual entity based on procedural relationships and derivations from physically recorded data. Upon completion, this facility will be integrated within the current implementation of the the STV for the INFOPLEX functional hierarchy which lacks the support for virtual information processing.

Thesis Supervisor: Professor Stuart E. Madnick
Sloan School of Management, M.I.T.

Contents

	Page
Title.....	1
Abstract.....	2
Acknowledgement.....	4
Contents.....	5
Chapter 1 Introduction.....	8
1.1.0 INFOPLEX Overview.....	8
1.1.1 Concept.....	9
1.1.2 Infoplex Architecture.....	9
1.1.3 Functional Hierarchy.....	10
1.1.4 Research Issues.....	10
1.2.0 Thesis Objectives.....	10
1.2.1 Background.....	13
Chapter 2 Virtual Information.....	14
2.1.0 Concept.....	14
2.2.0 Classification.....	14
2.2.1 Factored Facts.....	15
2.2.2 Computed Facts.....	15
2.2.3 Inferred Facts.....	16
2.3.0 Specification.....	17
2.4.0 Merits.....	17
2.5.0 Approach.....	19
Chapter 3 Functionalities.....	21
3.1.0 Underlying Data Model.....	21
3.2.0 Active Workspace.....	22
3.3.0 Permanently Defined Virtual Information.....	22
3.4.0 Adhoc Virtual Information.....	23
3.5.0 Notion of a Transaction.....	23
3.6.0 Virtual Attributes.....	24

3.7.0	Conditions on Real or Virtual Attributes.....	25
3.8.0	Virtual Entity Sets.....	25
3.9.0	Generalized Macro Facility.....	27
3.10.0	Extended Functionalities.....	27
3.10.1	User Dependent Virtual Definitions.....	27
3.10.2	Inferred Facts of Undesignated Indirection....	28
Chapter 4	Overview of Virtual Information Sub-System.....	29
4.1.0	Division by Infoplex Sub-Levels.....	29
4.1.1	User-Interface Level.....	29
4.1.2	Virtual Information Level.....	30
4.2.0	Internal Interfaces.....	30
Chapter 5	Storage Architecture and Manipulating Methods....	32
5.1.0	Inter-Level Query Structures.....	32
5.1.1	Entities.....	32
5.1.2	Attribute Values and Levels.....	33
5.2.0	Inter-Level Condition Structures.....	34
5.2.1	Execution Tree.....	34
5.2.2	Transition Rules Machine.....	34
5.2.3	Additional Information Table.....	34
5.3.0	Extra-Level Query Structures.....	35
5.3.1	Query Requester.....	35
5.3.1	Query Information Return.....	35
5.4.0	Pseudo Extra-Level Query Structures.....	35
5.4.1	The Dictionary.....	35
Chapter 6	Software Implementation.....	37
6.1.0	Dictionary.....	37
6.2.0	Machine Definer.....	38
6.3.0	Language Parser.....	38
6.4.0	Simplification of Request.....	46
6.5.0	Conversion of Storage.....	48
6.6.0	Execution of Discriminator.....	48
6.7.0	Final Processing of Data.....	54
Chapter 7	Design Considerations and Overview.....	55

7.1.0	Modularity, Modifiability, and Efficiency.....	55
7.2.0	Adaptability in Overall Infoplex Environment.....	56
7.3.0	Potential Power of Virtual Information.....	57
Chapter 8	Conclusion.....	58
	Bibliography.....	60
	Appendix A (Diagrams and Illustrations).....	61
Sections		
1.1.2	Infoplex Architecture.....	61
3.1.0	Underlying Data Model.....	62
3.2.0	Active Workspace.....	62
3.3.0	Permanently Defined Virtual Information.....	63
3.6.0	Virtual Attributes.....	64
3.8.0	Virtual Entity Sets.....	65
4.1.0	Division by Infoplex Sub-Levels.....	66
5.1.2	Attribute Values and Levels.....	67
6.0.0	Overview of Software Implementation.....	68
	Appendix B (Program Listing).....	69
Contents		
	All Copy (List of Macros).....	69
	Dctnry Listing (Dictionary Simulator).....	72
	Defmch Listing (Machine Definer).....	89
	Parse Listing (Finite-State Parser).....	98
	Smplfy Listing (Request Simplifier).....	125
	Cnvert Listing (Data Relocator).....	147
	Xecute Listing (Discriminator).....	158

1.0.0 INTRODUCTION

INFOPLEX DATA BASE COMPUTER is a current research project of the Information Systems Group at M.I.T.'s Sloan School of Management. It proposes a new architecture whose objectives are to provide substantial improvements in information management performance over conventional computer architectures, and to provide highly reliable support for very large and complex data bases.

1.1.0 INFOPLEX OVERVIEW

Progress of modern society has put increasingly more new and challenging demands upon the capability and performance of information storage, retrieval, and management. Conventional computers, whose architecture is designed primarily for computational objectives, are not suited to meet the requirements of these new demands. Efforts have been made in four different areas to build computer systems which will suit our information needs today, and in the future: (1) new instructions through microprogramming, (2) intelligent controllers, (3) dedicated computers for data base operations, and (4) data base computers. INFOPLEX is a research project belonging to the fourth category.

1.1.1 CONCEPT

INFOPLEX employs the concept of hierarchical decomposition which organizes information management functions into a functional hierarchy, and the physical memory management functions into a storage hierarchy (Madnick 78); both hierarchies consist of many independent levels of operation, each of which supports a different set of information or storage management functions through the use of multiple microprocessors.

1.1.2 INFOPLEX ARCHITECTURE

As stated previously, INFOPLEX is an architecture for data base computers based on hierarchical decomposition. A functional hierarchy of information management functions is built on top of a hierarchy of information storage functions. Both hierarchies are further divided into many functionally independent levels of operation, each of which is to be supported by a set of micro-processors operating in parallel with one another. A global Communication Bus coordinates inter-level transmission of data. This hierarchical architecture exploits the advantages of functional modularity of operations, and of parallel processing of micro-processors to systemize data base activities and to achieve a prescribed level of efficiency. A graphical illustration of this architecture is presented in appendix A (1.1.2).

1.1.3 FUNCTIONAL HIERARCHY

Current architecture of the functional hierarchy (Hsu 1982) with respect to data abstraction consists of four separate levels: (1) external level, (2) conceptual level, (3) entity level, and (4) internal level. A part of the conceptual level is a virtual information facility (Hsu 1982). These four levels of information management are highly independent of one another, and each is responsible for a different but necessary phase of information processing in a data base computer.

1.1.4 RESEARCH ISSUES

Major efforts of INFOPLEX research are devoted to the design, modeling, and evaluation of an optimal decomposition strategy for both the functional and memory hierarchy of information management and storage operation, and also to the study of an associated distributed control mechanism. This control mechanism would be used to coordinate the activities of and inter-level communications within the hierarchies.

1.2.0 THESIS OBJECTIVE

This thesis shares a joint mission with a concurrent thesis by Jameson Lee. The two theses are entirely separate in functionalities, but closely related and dependent upon one

another for a complete software simulation of the virtual information facility on the INFOPLEX data base computer architecture. This facility would incorporate the design and implementation of two sub-levels of the INFOPLEX functional hierarchy, the virtual information level, and an user interface level which is tailored for the use of virtual information processing.

Jameson Lee's thesis is responsible for the fulfillment of the front-end objectives of the joint mission; his objectives include the design and implementation of the following:

- a) A data base language to support virtual information
- b) A finite state machine to parse data base statements written in this language
- c) A user-interface tailored to the use of virtual information.
- d) A processor to process the creation, listing, and modifications of virtual definitions, as well as the substitution of these definitions into data base statements in actual use.

This processor would also be responsible for transforming data base statements into a chain of tokens, each of which

would include an indicator describing the classification of the token according to a prescribed classification scheme.

My objectives involve the implementation of the following:

- a) An execution tree of expressions and conditions within retrieval statements.
- b) A parser which implements a finite state machine that takes the match-action-next_state rules as input from the front-end.
- c) An entity set table of real and virtual entity sets within retrieval statements.
- d) An interface with the entity level in regard to the passing of values of real attributes.
- e) An extraction of the data which the user specified in his/her database statements.
- f) A virtual definition catalogue which consists of two separate dictionaries, permanent and adhoc.

The combined objectives of my "back-end" and Jameson Lee's "front-end" would fulfill our joint mission as mentioned ear-

lier, namely, to construct in software a virtual information facility with its own user interface, from here on referred to as VIFI, the Virtual Information Interpreter.

1.2.1 BACKGROUND

In the three short months in which VIFI was developed, we labored and wished to exhibit a certain degree of professionalism in its design and implementation. The merits of modular programming, of innovative algorithms, of performance efficiency, of functional capabilities, of user-friendliness of the proposed data based language, of program organization and flexibility, and even of consistencies in programming style were evaluated against time and labor limitations. A serious attempt was made to incorporate all of these characteristics into our Virtual Information Interpreter.

2.0.0 VIRTUAL INFORMATION

2.1.0 Concept

The concept of virtual information in data base systems has been developed and examined in earlier research of the Information Systems Group. Basically, there is a spectrum of the kinds of information which may be retrieved from a data base. Along this spectrum, pure data occupy an extreme on one end, and pure algorithms occupy the extreme on the other. In between these two extremes are the information which may be derived from a combination of data and algorithms; such information are dynamic and procedural in nature, and are referred to as Virtual Information.

2.2.0 CLASSIFICATION

Virtual information may be categorized into three major classes: factored facts, inferred facts, and computed facts. Together, these three classes of virtual information and combinations there of, constitute the portion of the information spectrum between the two extremes of pure data and pure algorithms.

2.2.1 FACTORED FACTS

Factored facts, subsets of data elements, based on certain prescribed conditions, or so called predicates, of attribute values, are often very valuable in structuring information in a useful manner. For instance, if a certain data base maintains records of weight, hair color, and salary for a group of employees, it may be useful to select from this group those individuals who share a certain condition on their attribute values, such as having black hair, making a salary greater than 8 dollars per hour, or weighing over 300 pounds. It is important that users of information should be able to access information independent of the particular factoring involved; this would imply the ability to support multi-level factoring, or repeated factoring of data.

2.2.2 COMPUTED FACTS

Computed facts are those information which are obtainable through the application of particular computational algorithms and operators on data or groups of data. These operators include arithmetic, comparative, boolean, and other kinds of functions. In the very least, computed facts include those pure data manifested in a different form, with a different unit of measure, or an alias name. For instance: a user may define a virtual age attribute to be the difference between the current year and a person's birth-year, a virtual rectangular area attribute to be the length multiplied by the width, or an

attribute value in the unit of inches to be 12 times the attribute value in the unit of feet. In this sense, transformations between different units of measure are intrinsic to the operations of computed facts.

2.2.3 INFERRED FACTS

Inferred facts pertain to implicit relationships which the data base system may arrive at through certain levels of indirection. In other words, a path, although indirect, does exist which leads to the desired data in storage. There are two ways by which the system on its own can support this kind of virtual information. The first method is by an exhaustive search of all possible paths, and the second is the application of a certain degree of artificial intelligence to deduce a viable path to the target data. Well, the first method is unbounded in computing time, and even when a path is found, it may not be the correct path; the second method is far fetched at this time. Therefore, we will give our attention to a different but comparable set of inferred facts which is implementable, and we give it the name Pseudo Inferred Facts. Pseudo Inferred Facts are exactly the same as inferred facts except that all the indirections will be explicitly designated by the user. With this strategy, exhaustive search is not necessary, artificial intelligence is not necessary, and the specified path would always be the designated and correct path. For instance, the Uncle relationship may be defined as the application of the

Brother relationship after the application of the Mother relationship.

2.3.0 SPECIFICATION

Users of information, through the virtual information facility, define their own working environment and the manner in which they would like to use the physical and underlying data. Such definitions of virtual information may be accomplished through a virtual information definition language. The virtual information facility would accept virtual information definitions and their modifications in the definition language, and respond to virtual information retrieval requests through a separate virtual information retrieval language.

2.4.0 MERITS

There are several major merits in the support of virtual information in a data base system. It is dynamic in nature because its definition may be created, deleted, and modified readily; its definition applies to all instances of data where it may apply, and yet there is but only one copy of this definition stored in the system. By facilitating the ease of modification, it enhances data base flexibility, by eliminating redundant physical records, it contributes to more consistent data, and by being procedural in nature, it enhances

information accuracy through the delay in the evaluation of data which vary over time or other changing factors until their time of use. These kinds of merits are based on virtual information's association with procedural relationships. For instance: the stored algorithm for computing age would eliminate the need to update the age attribute day by day if it were physically stored, and would be applied to calculate anyone's age, thus eliminating redundancy of stored information.

Virtual information also conserves the use of vast amounts of physical storage. It makes unnecessary the storage and maintenance of those information which may be derived upon request. This raises the issue of Time/Space trade-off, which should be seriously considered when deciding which kinds of fundamental data are or are not to be physically stored. Derivation upon requests will have the added cost of derivation; therefore, those information which will be used many times and are also difficult to derive may be the best kind of data to be physically stored; those information which is seldomly used and easy to derive may be the best kind of data not to be physically stored. Furthermore, the situation is made even more complex as we realize that the definitions themselves will require the use of physical storage. Thus, it wouldn't be an easy task to decide which kinds of data are to be derived, or to be actually stored.

The definition of virtual information on a per user basis would simulate an entire virtual data base for each individual user. Each one would be free to tailor the data base to his own preferred view or use through the virtual information definitions. A particular set of virtual definitions may be very useful for one group of users, and another set for another group of users. In this sense, each one has gotten a data base suited for his own use while not affecting anybody else's usage of the data base. A logical extension of this scenario is to implement access control mechanisms such that users may establish a controlled sharing of sets of virtual information definitions with one another; the data base administrator may monitor all such sharing to prevent unauthorized access to a certain set of virtual information functions. However, in a scenario as such, a separate catalogue would have to be maintained for each and every user, and considerable catalogue management would be required. Such is the cost for this individually user-tailored data base functionality, a secondary merit of the use of Virtual Information.

2.5.0 APPROACH

The concept of virtual information leads directly to a functional approach to data bases. A virtual information facility would be treated as a collection of functions, and retrieved data would be regarded as functional values. Virtual information requests correspond to function invocations; this func-

tional approach to information readily supports procedural relationships on which based the concept of virtual information. As a result, a virtual information facility is likely to resemble very much a language interpreter which accepts functional definitions and responds to functional invocations with specified arguments.

3.0.0 FUNCTIONALITIES

There are numerous functionalities to a virtual information facility, each of which may be implemented to a varying degree of completeness. Although it may be desirable to implement all the functionalities there are wherever possible, it may be too impractical and less than meaningful for the initial version of the implementation. Thus, we have not implemented the One Data Base per user feature of virtual information capabilities which we have described in the previous chapter. Later portions of this chapter would describe the functionalities of virtual information which we did implement; surely, not all of these implementations would be without room for further refinement, even though they already include an extensive set of virtual information capabilities.

3.1.0 UNDERLYING DATA MODEL

The virtual information facility lies on top of the entity set level of the functional hierarchy. In this level, the data base is seen as a network of entity sets and their attributes. Each entity set may have a varying number of attributes, some of them being value attributes and others being entity attributes. (Hsu 1980) The value attributes include a set of attribute values, and the entity attributes represent

relationships leading to other entity sets. Illustrations are found in appendix A (3.1.0).

3.2.0 ACTIVE WORKSPACE

We have developed an active workspace which incorporates a line editor with full screen display, through which user commands may be issued. The workspace consists of two buffers, an execution buffer, and a transaction buffer. The transaction buffer holds many data base statements which will be executed sequentially when the transaction buffer is executed. The execution buffer holds a single data base statement and will be automatically executed when a data base statement is completed. A number of buffer commands is created to manipulate buffer contents. Examples of these commands as well those of the data base statements are illustrated in Appendix A (3.2.0) since their design is detailed in the front-end description of the VIFI.

3.3.0 PERMANENTLY DEFINED VIRTUAL INFORMATION

Permanent virtual information may be defined through the Define statement. Such definitions will be stored in a global dictionary, or catalogue, in the form of character string, and will remain there until explicitly removed or over-written by a different definition. Examples may be found in appendix A (3.3.0).

3.4.0 ADHOC VIRTUAL INFORMATION

Virtual information definitions may be derived for only the duration of a single transaction. When all statements within the transaction are executed, the adhoc dictionary would be erased. Within the transaction, adhoc definition may be created, deleted, as well as modified at any time. With this feature, each transaction would be associated with a catalogue of its own, and would not interfere with the concurrent activities of other transactions executing in parallel. At this stage, we do not support concurrent transactions, but adhoc definition capability is still useful in the principle of individual environments of transactions and the scoping of virtual variables. Naturally, the permanent dictionary would also be accessible from within each transaction.

3.5.0 NOTION OF A TRANSACTION

A transaction is a body of executable statements joined together within a single context. This context is provided by the adhoc dictionary associated to the particular transaction. A transaction is created within the transaction buffer, and will remain there until it is explicitly over-written, erased, or executed. Merits of this transaction concept are threefold: a) a group of statements which collectively does a certain task may be consolidated to exhibit logical unity. b) a shared context may be created and maintained for each transaction, a sign

of transactional modularity and independence from one another.
c) the execution of the consolidated operations in a transaction may be put off until a more opportune moment, by which time new permanent or adhoc virtual information definitions may be defined either to supplement or to replace existing definitions.

3.6.0 VIRTUAL ATTRIBUTES

Virtual attributes equated to the results of computational algorithms acting on available data or of designated indirect references may be explicitly defined through the Define data base statement. This feature incorporates the support for Computed Facts as well as for Pseudo Inferred Facts. For instance, the following is the definition and usage of two virtual attributes, income and ship-country, a computed fact, and a pseudo inferred fact.

```
Define income as salary - expenses ;  
Retrieve ( {teachers} ) by ( {V0} name, income) ;
```

The foregoing retrieve statement returns two vertical columns of data. The first column being teacher's name, and the second column being their corresponding incomes.

```
Define ship-country as company ( country (name)) ;  
Retrieve ( {ship} ) by ( {V0} name, ship-country) ;
```

This foregoing retrieve statement returns two columns of data, the first being individual ship names, and the second being the name of the country to which the ship belongs. More examples of this entity set is in appendix A (3.6.0).

3.7.0 CONDITIONS ON REAL OR VIRTUAL ATTRIBUTES

Arbitrary conditions on real or virtually defined attributes may be defined by INFOPLEX users as the shared 'condition' on their data values from which factored facts may be later constructed. For example:

```
Define old as age > 70 ;  
Define rich as assets > 1000000 ;  
Retrieve ( {people} where (rich and old))  
by ( {V0} name);
```

The foregoing retrieve statement would return a list of names of those people whose age > 70 and assets > 1000000.

3.8.0 VIRTUAL ENTITY SETS

Aside from virtual attributes, we also support a basic notion of virtual entity sets. We recognize two kinds of virtual entity sets:

a) Union, intersection, or cartesian of real or previously defined virtual entity sets based on their real and virtual attribute values.

b) Subsetting of real or virtual entity sets based on certain conditions on their real and virtual attribute values.

For instance:

Define ClassAB as $\{ \{ \text{ClassA} \} \text{ MU (Name) } \{ \text{ClassB} \} \}$;

ClassAB is defined as the result of a multiple-union operation on entity sets ClassA and ClassB, based on a common attribute called Name.

Define RichMen as $\{ \text{Men} \}$ where (assets > 1000000) ;

RichMen is defined as a virtual subset of the set Men, based on the values of its asset attributes.

The complete set of union and intersection operators as well as the cartesian product operator between entity sets is illustrated in appendix A (3.8.0). Also included are details of the capability to specify various conditional predicates on attribute values.

3.9.0 GENERALIZED MACRO FACILITY

Users will be able to give arbitrary definitions to specified names and later substitute for these names in the database statements. In this sense, the define statement may be used not only to define virtual attributes and virtual entity sets, but also random definitions that may seem to be incoherent without the proper context. When a retrieval statement is to be executed, each word within the statement is checked against a list of stored definition names; any word that matches with any definition name would be replaced from the dictionary by the definition. Furthermore the words in the definition substituted would likewise be substituted, if possible.

3.10.0 EXTENDABLE FUNCTIONALITIES

3.10.1 USER DEPENDENT VIRTUAL DEFINITIONS

This particular functionality is not difficult to implement, but it may be unnecessary at this stage of the project. It simply would require a separate catalogue for each user which includes an access control list, proper search rules including default situations, and adequate coordination and control mechanisms to manage the various catalogues. It would increase the cost in terms of time and space efficiency. Thus, we have not included this functionality in this version of virtual information implementation. Nevertheless, if circumstances in

later time are such that the support for user dependent catalogues is so desirable as to more than compensate for its cost of implementation, this functionality may be added readily. At this time, we can simulate this by tagging all definition names with the unique user id of the user and hence make the name unique.

3.10.2 INFERRED FACTS OF UNDESIGNATED INDIRECTION

Inferred facts with undesignated indirection, rather than pseudo inferred facts with designated indirection, is likely to have tremendous costs in system performance whenever it is to be implemented. As previously stated, this would require either an exhaustive search or a certain level of artificial intelligence, both of which require large amounts of resources in computing power, storage and time. Furthermore, in order to verify that the indirection the system chooses at each step along the way is correct, the user has to monitor the computer decisions interactively; this defeats the original purpose of not having the user to designate his intended path of indirection. Thus, it seems very doubtful that this functionality will ever be implemented unless the requirements for user monitoring of the decision process is somehow eliminated.

4.0.0 OVERVIEW OF VIRTUAL INFORMATION SUB-SYSTEM

4.1.0 DIVISION BY INFOPLEX SUB-LEVELS

The virtual information sub-system related to our project extends into two levels of INFOPLEX, due to necessity from the modular designer's point of view. The interaction between the user and the input peripherals of the sub-system must be isolated to one level, separate from that used to process the requests. Since part of our sub-system resides in a level external to the Virtual Information Level, data flow must be channeled through the INFOPLEX control structure data bus when passing to and from the input section to the processing section, in a proper implementation. A diagram of our entire sub-system and the surroundings is in appendix A (4.1.0).

4.1.1 USER INTERFACE LEVEL

The user interface level currently consists of a user interface activity coordinator and a virtual information interface buffer. When completely and correctly implemented, the user activity coordinator acts as a switchbox to activate either the virtual information interface buffer or a real information interface buffer. Likewise, it arbitrates between sending the input data to the virtual information processor or the real

information process in the Virtual Information Level. In addition to placing user inputs into proper storage, the interface buffer alters certain characters from the input so that the lower levels may safely use those characters as delimiters.

4.1.2 VIRTUAL INFORMATION LEVEL

This level is partitioned into two sections, one of which is the real information processor that creates and updates entity sets. We are not concerned with that portion of the design. The section of interest is the virtual information processor. Here we have a virtual information activity coordinator that sequences the data manipulators. First it has the input statements cleaned of virtual information by substitution from the dictionary; it then converts the statements into tokens. The tokens translate into database query commands when they are parsed using finite state transition rules. The coordinator then sends some queries to the lower INFOPLEX entity level for processing. The returned data are relocated to useable storage and processed further to extract the exact elements which the user specified by his/her command. Control then goes back to the User Interface Level.

4.2.0 INTERNAL INTERFACES

To avoid unnecessary conflicts and lack of determinism in data transfer between routines internal to a level, the coordina-

tors in the two levels, User Interface and Virtual Information, are capable of monitoring all data movements. This is a merit of the horizontal structuring of data accessing and processing paths. Each interface is well defined. The user interface coordinator transfers character buffers in both directions to the buffer processor. The buffer can be then altered minimally and sent down the INFOPLEX bus. The virtual information level activity coordinator sends a facsimile of the character buffer to the tokenizer and receives a list of tokens in return. The virtual level coordinator then passes the tokens to the parser and receives a set of filled tables which are copied into a simplified version to be used as a query request to the lower levels. The filled tables become the main means of transferring data from then on. There presently exists a dictionary which should be later fully embedded in the database storage system. In other words, we treat internal interfaces very much like the external ones which attach to the INFOPLEX bus, since we want modularity and simplicity of operation.

5.0.0 STORAGE ARCHITECTURE AND MANIPULATING METHODS

Since the data retrieval requests and returns are based on structures, a necessity exists to clarify what is involved in accessing these structures and to exhibit the kinds of data manipulation that can be performed on the structures.

5.1.0 INTER-LEVEL QUERY STRUCTURES

5.1.1 ENTITIES

The structure ENTITY consists of thirteen sets of vertically arranged entity set representations. Each entity has a NAME, DEPTH (number of values in a single occurrence attribute), VES_FN (entity set function), WHERE (discrimination tree pointer), N_PARENT (entity parents pointers), VES_PAR (virtual entity set pointer to attribute map between parent entities and child entity), and a set of fifteen attributes. Each attribute has a VES_KEY (designates a key attribute to a virtual entity set), CART_KEY (designates a cartesian key attribute to a cartesian virtual entity set), SING_OCC (designates a 1:1 correspondence attribute), A_PARENT (assigns attribute parent if non-zero), USES (attribute name), and LIST (locates all values of an attribute).

Real entity sets are created from the information available in the entity set representation. An abridged version of ENTITY is made and sent to the entity level of INFOPLEX and is returned with attribute values linked to it.

Virtual entity sets are created from real entity sets or from existing virtual entity sets. Whenever VES_FN specifies that an entity set is virtual, the N_PARENTs have their attributes mapped by the N_MAP attached to VES_PAR. Based on the proper relationship amongst the keys and labels, N_MAP determines which attributes are generated by which existing attributes; this map is extremely useful for future linking of attributes across entity sets. The VES_FN defines the particular scheme for creating the entity set.

5.1.2 ATTRIBUTE VALUES AND LEVELS

The ATTRIB structure of itself is very simple. It contains one attribute value and the level number of that value. The inter-relations of the ATTRIBs is flexible but complex. Attribute values are linked across the attribute table in the entity set by the level number; such a set is always copied or skipped as one since the correspondence is entire. An example can be seen in appendix A (5.1.2).

5.2.0 INTER-LEVEL CONDITION STRUCTURES

5.2.1 EXECUTION TREE

XTREE is structured such that it represents a tree with nodes and links. Traversal through the tree gives a natural sequence of performing operations and hence a correctly built tree can be used as the control structure of data processing. Each node uses the LABEL field once and each arc of a node uses one LINK. The number of links in a node is stored in CHILD so that tree traversal is easily achieved.

5.2.2 TRANSITION RULES MACHINE

MACH stores the information that is necessary to transform database statement tokens to query structures. In a full implementation of INFOPLEX, MACH should be totally static, since it is needed for any inputs to the VIFI. The control structure itself should boot up the entering of information into this table.

5.2.3 ADDITIONAL INFORMATION TABLE

Useful for storing more information than is allowed by a single node in XTREE, and as a workspace to generate attributes, XCHNGE should reside in the VIFI as a local storage area, in the same respect as ENTITY.

5.3.0 EXTRA-LEVEL QUERY STRUCTURES

5.3.1 QUERY REQUESTER

RETE_ARG is basically an extracted version of ENTITY, with control structure information added. Additionally, it has a COND sub-structure to store simplifiable conditions for lower level processing.

5.3.2 QUERY INFORMATION RETURN

RETE_RTN is essentially the storage carrier for ATTRIB. It contains control structure information.

5.4.0 PSUEDO EXTRA-LEVEL QUERY STRUCTURES

There is currently storage used to simulate storage that should be available from within the database itself. This is due to the fact that simulated retrievals are more easy to handle and check for errors.

5.4.1 THE DICTIONARY

The DICTON table is split into three sub-tables, to simulate a number-of-lookups CHECKer, an ADHOC dictionary, and a MAIN dictionary. With appropriate tagging and cataloging of definition names, the main dictionary should eventually be able to

take on this task. Certain amounts of security restriction can be made by this dictionary interface as well.

6.0.0 SOFTWARE IMPLEMENTATION

The software described is fully listed in appendix B. We used PL/I on the IBM 370 machine, CMS operating system. Only those rear-end routines are described; front-end implementation can be found in Jameson Lee's documentation. By relating to the overview diagram in appendix A, one can see the correlation of the software routines.

6.1.0 DICTIONARY

The dictionary routine, DCTNRY, is used only to simulate the storage of data in the main database. Since some necessary functions, such as template string matching or variable length strings, are not yet available in the database, a more powerful simulator was substituted. Due to the modularity of the system, it would be trivial to replace this with the real thing.

Our dictionary currently has a few special features. It can do pseudo parameter passing by replacing characters in the definition by characters in the definition name matching string. It has a counter that prevents unlimited circular definitions. Most of all, it has a defined search order that consists of the built-in function catalogue, the adhoc dictionary, and the main dictionary. The adhoc dictionary and the

counter can be cleared by a special command generated by the dictionary user.

6.2.0 MACHINE DEFINER

DEFMCH simply reads the transition rules from an input file into a table which can be accessed more readily by indexing.

6.3.0 LANGUAGE PARSER

The PARSE routine basically interpretes the linked list of tokens specified by TOKENS_PTR and produced by the TKNIZE routine, and translates this list to a request format to be later simplified and sent to the lower entity level for processing and retrieval. It does the interpretation based upon a set of finite-state transition rules that have been input by the DEFMCH routine into a transition rules table named MACH. The parameters filled by the parser are the execution order tree, XTREE, the additional information exchange table, XCHNGE, and the unabridged entity set query table, ENTITY. Of course, the PARSE routine can be put into a DEBUG mode to follow the transition states that are passed given a set of tokenized retrieval statement and the transition rules table.

The main data storage elements in PARSE in addition to the parameters aforementioned are the simulated stacks, STACK, and the virtual entity set map table, VES. The first two stacks are

used by the finite state parser in a push-down automaton scheme, in which they are respectively labeled as the operand stack and the operator stack, whereas the third one is used internally and exclusively by the parser to track the addresses to the execution tree.

A word of clarification is necessary here. In the description of PARSE to follow, the term 'item' refers to a multi-faceted abstraction of data in the sense that it could be anything from data on the level relevant to our processing, the same data with internally used update sources or labeling tags imbedded, to internally used mapping link numbers. Furthermore, the data being processed may be utilized by different levels, such as the finite automaton or the entity set request processor, in a related but differing way. For instance, the item '+:B', which means 'plus of type built-in function', is the composition of a symbol on the automaton level typed in by the user and an internally generated tag; the '+' portion is used by the automaton level as tokens to be matched and by the entity set retrieval processor as the operation 'plus' to be performed.

PARSE has a main driver loop which is traversed once for each transition state that is entered by any particular input statement. The loop coordinates the pairing of matches that fire according to MATCHING and the actions to be used and the next

transition state to be entered according to ACTING. The top level sub-routines of PARSE are:

MATCHING which determines whether the states of the input linked list and the stacks are matchable with any of the required conditions for firing in a given transition state.

ACTING which assigns the actions taken by the current transition state to arrive at another state, and designates this next state.

GETS which is a general string functions that breaks an argument item into two parts seperated by a chosen character in the item.

PUSH which pushes an item onto a specified stack.

POP which pops the top item off a specifed stack.

CHANGE which alters an item based upon special characters in the item selecting sources of update.

GENENT which takes items off the stacks and generates an entity request into ENTITY.

VIRTX which replaces the top item on the operand stack, which must be a virtual entity abstract map number, with the corresponding real map number.

VIRTA which adds a new entity map number into the virtual entity set catalogue.

GENNODE which takes items off the stacks and generates a node on the execution tree, with the appropriate links to other nodes in the same tree. The link to the new node is put onto the operand stack.

GENATTR which puts the specified attribute name into the entity set retrieval table, with the appropriate links to other attributes in the particular entity set of interest. The specified attribute name is converted to a reference number to the attribute in the entity set table.

EXCH which exchanges the top item on the operand stack with the map number to the next available space in the additional information table so that further items may be appended to the additional information table as part of the current top item. A tag is appended to the top item to identify what type of additional information, indirection of attributes or multiple comparison, is at issue.

ADDON which appends additional items onto the most currently addressed space in the additional information table.

Of these, MATCHING, ACTING, GENNENT, GENNODE and GENATTR are complex enough to deserve further explanation.

MATCHING simply decides whether or not a certain transition state conditions item matches the current existing conditions of inputs. The conditions item may be a set of any number of conditions each of which if matched will flag the match as positive. Each condition specifies what must be found in a set of sources consisting of the first item on the input linked list, and the top of stacks of the first two stacks. Furthermore, allowance has been made to allow translation of a single matching item to a multiple list of possible matching items to effect storage minimization in the transition rules table.

ACTING processes the list of actions found in the transition rules table. It takes the actions one by one and distributes the processing over several loops. Each action consists of an action name followed by arguments whose existence and typing vary dependent on each individual action. A short summary of the actions consists of:

'push' which pushes its second argument onto the stack specified by the first.

'pop' which pops off the top of the stack specified by the first argument, using POP.

'del' which eliminates the first item on the input linked list by advancing TOKENS_PTR and deallocating the first token.

'genent' which invokes the sub-routine GENENT to generate an entity set.

'virtx' which replaces the top item on the operand stack with the real map number indicated by the item, using VIRTX.

'virta' which adds a new entity set mapping into the virtual entity set catalogue, using VIRTa.

'attwhr' which sets the appropriate linkage between the entity set table, ENTITY, and the execution tree, XTREE.

'gennode' which generates a node on the execution tree, using GENNODE.

'indx' which exchanges the top item on the operand stack with the map number to the next available space in XCHNGE, using EXCH with the tag for indirection.

'mulx' which exchanges the top item on the operand stack with the map number to the next available space in XCHNGE, using EXCH with the tag for multiple comparison.

'addon' which adds the top item on the operand stack onto the additional information table, using ADDON.

After all actions have been performed, ACTING designates the next transition state to be entered, based on the next state entry in the rules table, which if negative would indicate a 'subroutine return' found on the operator stack.

GENENT does the generation of entity set requests using the ENTITY table. The appropriate items must already exist on the stacks to allow for the generation and hence the parsing rules must provide for the set up. The types of entity sets are:

'r' which is an explicitly user defined entity set referring to a real entity set existing inside the database.

's' which is an implicitly created entity set that refers to a composite entity set based on a single entity set, with additional constraints attached.

'mu', 'mi', 'su', 'si', 'cs' which are explicitly user defined entity sets referring to composite entities each of

which is based on two entity sets. The specifics of each entity type are described elsewhere in this document.

The necessary attribute keys in any key lists are catalogued by GENENT. An entity set real map number is also placed on the operand stack at the conclusion of a generation to allow proper linking of entity sets.

GENNODE generates nodes on XTREE by taking the top item on the operator stack and reserving sufficient number of locations on the tree for the children of the operator of interest. It puts the link numbers to these locations on the internal address stack and proceeds to process the children of the operator, which may be other non-terminal or terminal operators, or terminal constants or variables. The operators are represented on the operand stack as map links and the constants or variables exist on the stack as themselves with their respective taggings. The sub-routines PUTX and GENATTR are invoked whenever a terminal constant or variable is encountered.

GENATTR takes items pertaining to variables about to be put onto the execution tree or about to be recorded as keys to entity sets and places the decoded information into the entity set table. The data processed indicate the name of the attribute, any parent of the attribute, and whether the attribute is a key, either virtual information, cartesian, or both.

Hence we have described the operation of the finite-state, push-down automaton parser necessary to encode the user inputs into specifications comprehensible to the entity level for retrieval purposes. The specifications are encoded into pre-formatted tables.

6.4.0 SIMPLIFICATION OF REQUEST

The SMPLFY routine minimizes the amount of data flow across the boundary from the virtual information level to the entity level by abridging the entity request table, ENTITY, and creating the retrieval set, RETE_ARG. Only requests for real entity set information exist in RETE_ARG. In addition, the execution tree, XTREE is trimmed by the removal of certain easily manipulated conditions and the placement of these conditions in RETE_ARG for processing in the lower levels of INFOPLEX. Before any of this is done, however, SMPLFY's task is to assign all attributes correctly into the appropriate entity sets, using the propagation function PRPGATE.

PRPGATE sets up the proper relationships between entity sets and attributes, in the sense that attributes used in any entity sets which are composed from other entity sets must exist in the parent entity sets as well. An attribute map table is filled for each composite entity set to make the attribute relationships endure through the retrieval and execution phases of the data processing. All attribute names and parent

relationships are ensured when propagating so as not to create conflicts in naming. For instance, if entity B is based on entity A, then if entity A has attributes $X \rightarrow Y \rightarrow Z$ (here, we mean that X, Y, and Z are each attributes of A, with X being the parent of Y being the parent of Z) and entity B has attributes $X \rightarrow U \rightarrow Y \rightarrow Z$ then entity B, after propagation, would acquire the new attributes U, Y, and Z, with the parent relationship connected; thus B would have two attributes by the name Y and two by Z.

SEARCH goes through the trees attached to each real entity set and attempts to simplify their structure whenever and only whenever comparisons are made between pairs of variables and constants and which are at most joined by the 'and' joiner. The rationale for processing thus has to do with the power and efficiency of the unary level underlying the VIFI. The trick that SEARCH uses in amputating the tree involves the tagging of nodes so as to only prolong those branches which reach unresolvable nodes in terms of simplification. When a simplification is found, the SETREL sub-routine is invoked to set the relationship of interest into RETE_ARG for computation at the lower levels. An example of a partially resolvable tree would be (name = 'Bond' or id_num = '007' and job = 'spy'), where the name and id_num attributes cannot be taken out of the tree but the job attribute can because it is joined to the other conditions by a simple 'and'.

Once simplification is done, the control of the virtual information stage must be temporarily passed down to the entity level for request processing in order for further actions to be performable in the virtual information level. After all, virtualness has to be based upon some form of substance.

6.5.0 CONVERSION OF STORAGE

The CNVERT routine is a simple routine to transfer data from specialized storage necessary to the INFOPLEX control structure to storage more dedicated to the internal environment of the virtual information stage. This transfer allows uniformity of data processing in the VIFI. Of even greater significance is the amount of independence and flexibility the VIFI achieves from the external environments. Only CNVERT needs to be altered should any interface requirements be updated. Once the information tags and attributes' values of the retrieved real entity sets have found new shelter, CNVERT proceeds to eliminate their old home.

6.6.0 EXECUTION OF DISCRIMINATOR

EXECUTE is the meat of the data processing stage of the VIFI. The functions which it performs are creating virtual entity sets and applying additional constraints on completed entity sets for the purpose of discrimination. Here, discrimination is the subsetting of the entity sets based on the user speci-

fied conditioning clauses which were not simplified and sent to the entity level. The sub-routines of XECUTE are divided into two components: the set generation/condensing routines and the subset conditioning routines. They act on the retrieved data which have been attached to the ENTITY query table and base their actions on information in the ENTITY table itself, the execution tree, XTREE, and the additional information table, XCHNGE.

XECUTE has a driver loop that distributes the operations on each entity set in turn. The set generation/condensing routines are:

MUNION which builds a multiple union of two entity sets by copying all the attribute values of the first parent and those values of the second which do not occur in the first.

MINTER which builds a multiple intersection of two entity sets by copying only those attribute values of the first parent that exist in the second.

SINGLE which condenses an entity set by moving all attribute values of the entity set to a dummy entity set and then copying back only those values whose occurrence is the first or only in the entity set.

CRTESN which creates a cartesian set by copying multiple attribute values from each of its parent entity sets. It copies each value of the first parent the number of times dictated by the depth (number of values for a single occurrence attribute) of the second, and copies sets of values of the second parent the number of times dictated by the depth of the first.

The basic sub-routine with which entity sets are copied is COPYSET. It copies an entire set of attribute values from a specified parent entity set to the child, with special care in regards to single or multiple occurrence. Markers are set by the caller to COPYSET to track the set of attribute values to be copied and these markers are advanced at the end of copying. Level numbers are used to manage the copying of multiple occurrence items. A counter is incremented to track the number of sets of values copied. In the same line is SKIPSET, which skips a set of attribute values by incrementing the abovementioned markers to the specified parent without doing any copying. Hence we can control which sets of attribute values are to be copied to the child. COPYALL, on the other hand, just copies all values from an parent entity to a child.

Of aid to set generation are the sub-routines LOCATEKEYS and IUMATCH. The first sets up a key table to allow easy access to entity keys for the purpose of determining

intersections and unions. The second determines if two sets of attribute values are in fact one, according to the keys assigned.

The subset conditioning routines function recursively to traverse the parser-filled execution tree in order to perform the user specified operations on the attribute values of the real and virtual (composite) entity sets. The argument passing between the recursive calls is through structural duplicates of the linked lists containing the original copies of the values. The main recursive sub-routines are as follows:

OPERATE sequences the processing of the nodes by identifying each node's label and/or number of children, and subsequently distributing the manipulation of the child or children to the other routines, or to a recursive invocation of itself. A node of no child is either a variable, constant, multiple constant, or operator of no argument; the terminal node must therefore create a vertical or unitary list to return to the tree node above. A node of one child is an operator of one argument and hence must return either a unitary or vertical list, depending upon the operator type. A node of two children is an operator with two arguments and may have a multiple constant as a second child. The GO_DOWN routines execute the operations, defined by the labels of the nodes, on their

children, if any exist. The STR routine is a special string operation handler which is also recursive.

STR is a very powerful routine that handles nodes labeled 'str'. It does substring operations on the first child of the 'str' node with specifications from the other five children. A clearcut description of the 'str' function is in Jameson Lee's description of the front-end of the VIFI. Essentially, STR uses OPERATE on those of the second, third, fifth, and sixth children of the 'str' node that are relevant to the type specification. The fourth child specifies the type of string addressing, absolute or relative, that is desired.

The support routines to the recursive ones above are:

GO_DOWN0 returns a unitary list containing data returned from no argument operators.

GO_DOWN1 checks to see if the operator of one argument that it handles returns unitary lists or vertical lists. If a vertical list is to be returned, it goes through each item in the list to evaluate the return for that item.

RET_UNITARY decides whether a one argument operator is defined to always return a unitary list. It also evaluates such operators with the appropriate operand.

UNARY takes a one argument operator and its argument and computes the return.

MULTEQ2 takes the '=' operator and its two arguments, the second of which must be a multiple constant, and evaluates the results of the first argument values each compared with the multiple list.

GO_DOWN2 determines the type of return that is expected of a generic two argument operator and proceeds to fill out a list with that specification.

BINARY takes a two argument operator and its arguments and computes the return.

CREATE_LIST creates a duplicate list of the values of the specified attribute, to be used for processing in upper nodes.

CREATE_CONST creates a unitary list of the single constant passed to it. It also can create a multiple constant index list which is used to address a multiple constant if the terminal node on which it acts is of that type.

TMPLTE is a function that can be used to do template matching on a string, hence giving the user this additional power in value searching.

DECRSET is used to decrease the number of values in an attribute by moving an entity set to a dummy and copying back only those sets of attribute values that have a corresponding ':true' in the conditioning list.

The routine DISPOSE is useful for ensuring no storage overflow since it is invoked whenever unnecessary storage can be deallocated.

The XECUTE routine that is used to discriminate information to the level specified by the user is thus described.

6.7.0 FINAL PROCESSING OF DATA

Although the DOBY routine has as yet not been implemented, due to unresolved problems regarding output interface, it is envisioned that this routine will essentially use the conditioning sub-routines of the XECUTE routine and return the condition lists to the user. Hence it follows the trend and methodology of our processing. It will return those lists which were specifically mentioned by the user and tag them with the labels, which could be of type virtual information, that should be extracted by the tokenizer in the earlier stages of processing.

The full complement of software for the rear-end is thus described.

7.0.0 DESIGN CONSIDERATIONS AND OVERVIEW

7.1.0 MODULARITY, MODIFIABILITY, AND EFFICIENCY

In terms of modularity, I think we have achieved a pretty good record. Our design strategies concentrate on clean interfacing of routines, with clearcut specifications of interfacing parameters. The horizontal control structure, regulated by the activity coordinators, provide sensitive debugging capabilities not available in vertical programming. The manner of tokenizing and parsing database statements allow for changes in the format of retrieval requests without affecting the inner routines. The way data incoming from the entity level is relocated allows easy changing of interface to the lower level. In general, I have no dissatisfactions concerning these issues.

There is, however, a concern I have regarding the complexity in the manner in which certain manipulations of data are performed. Since a certain amount of efficiency is desired in processing, a lot of not-so-obvious shortcuts were taken to derive results. Hence, documentation is required on the level so extremely clear that it becomes a drudgery. Furthermore, certain already implemented functionalities will not be used simply because not enough clear documentation is available.

The efficiency of the INFOPLEX structure has a lot to be desired. This is but a practical matter of structural design verses kludgy efficiency. In a microprocessor implementation, however, i am sure certain structural concepts would give way to efficiency and yet maintain that its integrity factor.

7.2.0 ADAPTABILITY IN OVERALL INFOPLEX ENVIRONMENT

The current virtual information stage is in concept adaptable to the current INFOPLEX environment, but there are quite a few areas in which conflicts arise because of lack of coordination of effort in the updating of the system as a whole. Even though modularity is desired, certain capabilities cannot be realized unless all stages of the system support those capabilities. With parallel microprocessor processing, the speed consideration in the inherently slow hierarchical control structure would hopefully diminish. The Test Vehicle can only approximate what the actual system can do; it can only illustrate some of the possible ways to view the manner data ought to be managed.

7.3.0 POTENTIAL POWER OF VIRTUAL INFORMATION

I personally can see quite a lot of possibilities in this virtual information idea. Our current implementation is somewhat restrictive in regards to some of the desired features, such as full inferring of facts. It can be adapted to practi-

cally any situation, however. Unfortunately, since the re-implementation of programs in general usually is easier done than the updating of existing ones (I program enough to know this), I fear that our package is only a transient element in the progression of the INFOPLEX system. As it is, we have put into VIFI all that could be allowed in the time allotted, and this already seems far enough advanced in comparison to the other components of the system. Hopefully, sometime all the desired features can be put together and INFOPLEX can see the market.

8.0.0 CONCLUSION

The INFOPLEX project seems to be a promising one. The virtual information concept is quite realizable using various schemes to track the transformation of data. According to the tests we have done to generate entity level requests from user inputs that we would consider fairly unrestrictive in terms of power, the difference between 'virtual' and 'real' information is but a few operations to connect the two. We were able to construct fairly restrictive entity requests when the user entered retrieval commands sometimes more complicated than those found in programming languages, and we were able to resolve the commands to those handled by the lower level, for the sake of efficiency, and those handled at the VIFI, for the sake of computational power and flexibility. The parser's approach to converting user-input commands provides a natural extension to increase the capability of the language. As it is, the user can already define a lot of functions based on the built-in ones. We can easily implement such things as data types or data units by a minimum amount of hacking because we already have a system of user definitions. However, since this is only the prototype of the virtual information processor, we feel that what we have is more than adequate.

The only unfortunate thing is that we were not able to fully test out the processing stages of the VIFI, since we lacked sufficient test data and time to establish the necessary linkages to the other components of INFOPLEX. Hopefully, our VIFI will eventually be integrated into the Test Vehicle, and I am sure it will fit in well in the design.

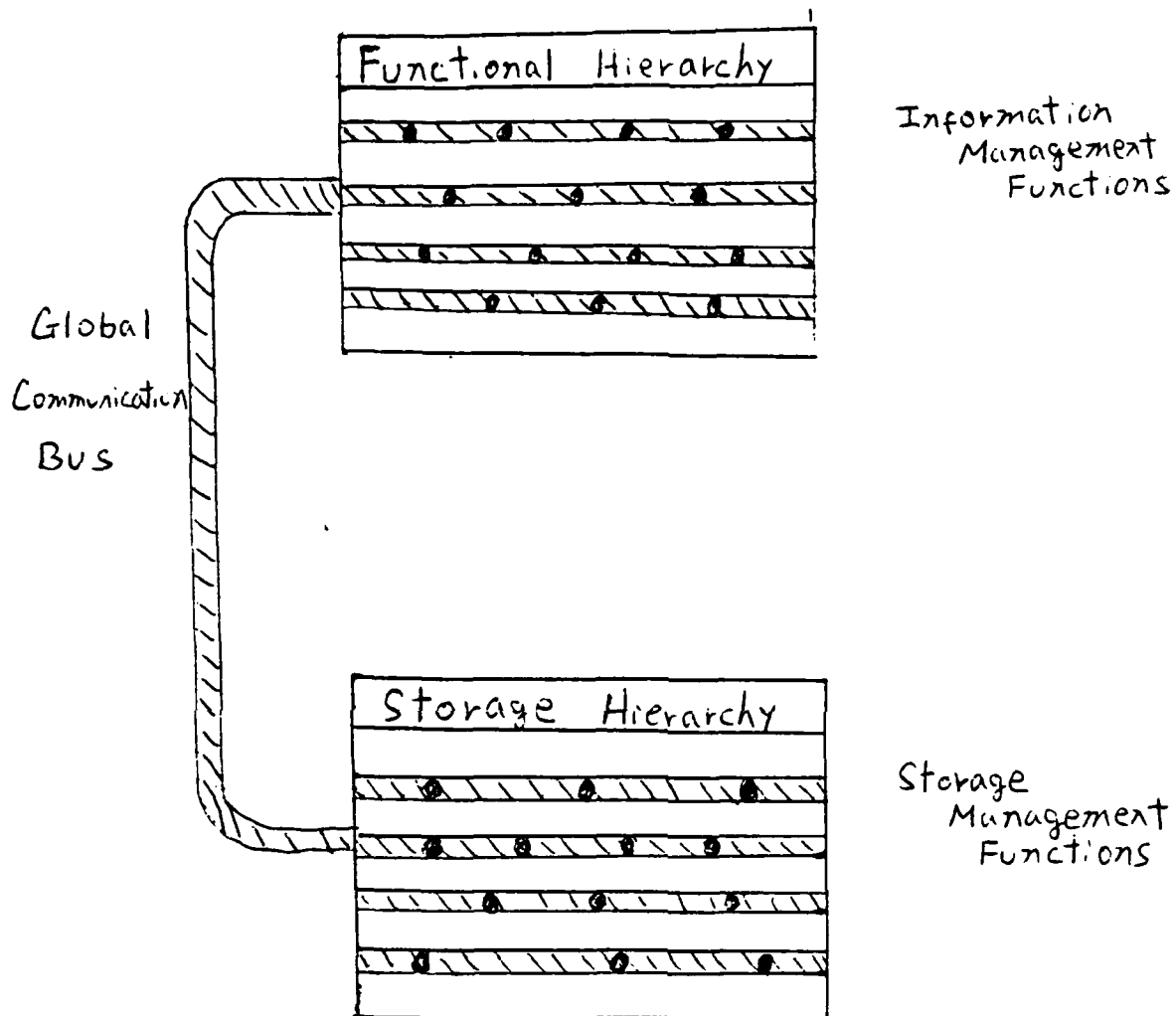
Bibliography

1. Harry R. Lewis, Christos H. Papadimitriou,
'Elements of the Theory of Computation'
Prentice-Hall Software Series
2. David Gries,
'Compiler Construction for Digital Computers'
John Wiley & Sons
3. Jeffrey Folinus, Stuart E. Madnick, Howard Schutzman,
'Virtual Information in Data Base Computers'
Center for Information Systems Research, M.I.T.
4. A Klug, D Tsichritzis,
Multiple View Support Within the Ansi/Sparc Framework
Center for Information Systems Research, M.I.T.
5. Meichun Hsu
'FSTV - The Software Test Vehicle for the Functional
Hierarchy of the INFOPLEX Data Base Computer'
Center for Information Systems Research, M.I.T.
6. Thomas A. Standish
'Data Structure Techniques'
Addison-Wesley Computer Science Series
7. Aho Ullman
'Principles of Compiler Design'
Addison-Wesley Computer Science Series
8. Tak To
'SHELL: A Simulation for the Software Test Vehicle
of the INFOPLEX Data Base Computer'
Center for Information Systems Research, M.I.T.
9. Chat-Yu Lam, Stuart E. Madnick
'INFOPLEX Data Base Computer Architecture'
Center for Information Systems Research, M.I.T.
10. Bruce Blumberg
'INFOSAM - A Sample Database Management System'
Center for Information Systems Research, M.I.T.

Appendix

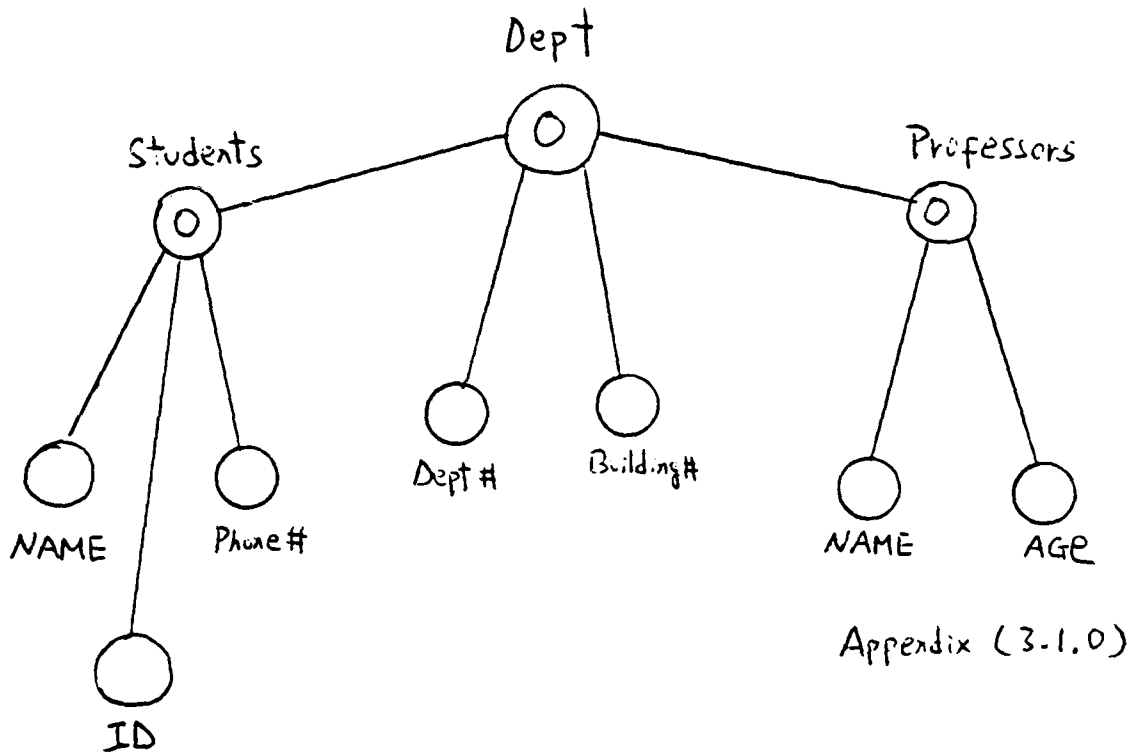
A

INFOPLEX



Appendix A (1.1.2)

Data Model
Data Statements



Appendix (3.1.0)

Database statements

Define
Adhoc
Listdef
Retrieve

Buffer Commands

Trans	Rxtrans
Endtrans	Terminate
Finput	Help
Fsave	Erasebuff
Dodel	Killexec

{Refer to Chapter 5 of Jameson Lee's thesis}

Appendix (3.2.0)

Virtual Definitions

Define heavy as weight > 100 ;
(defined)

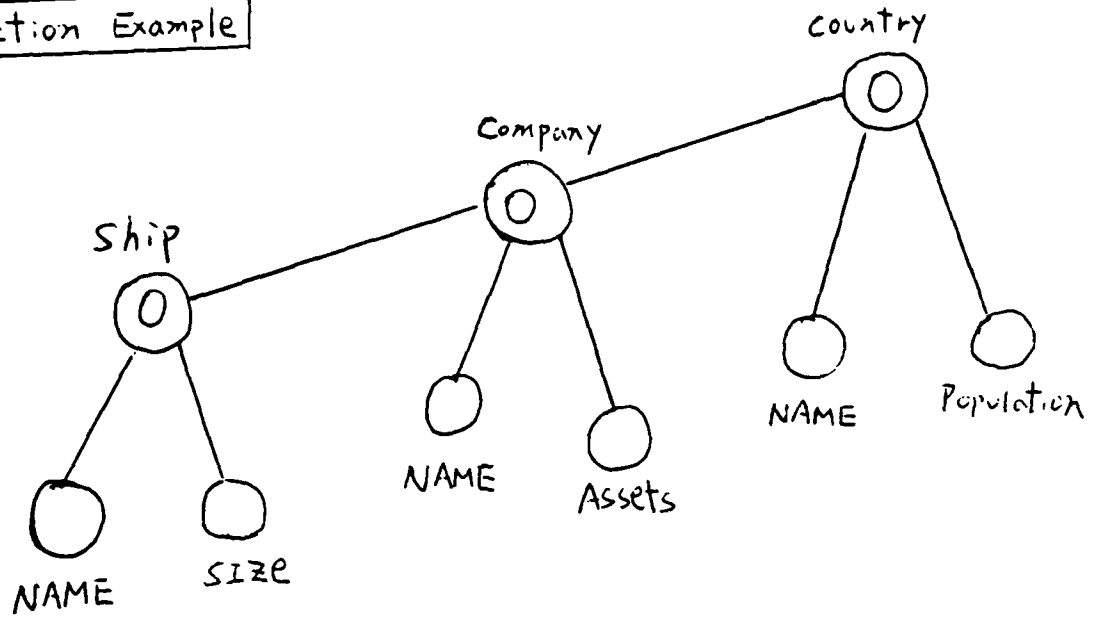
Define heavy Remove ; (removed)

Define heavy as weight > 200 ;
(redefined)

Define heavy as weight > 50 ;
(replaced)

Appendix (3.3.C)

Indirection Example



Appendix (3.6.0)

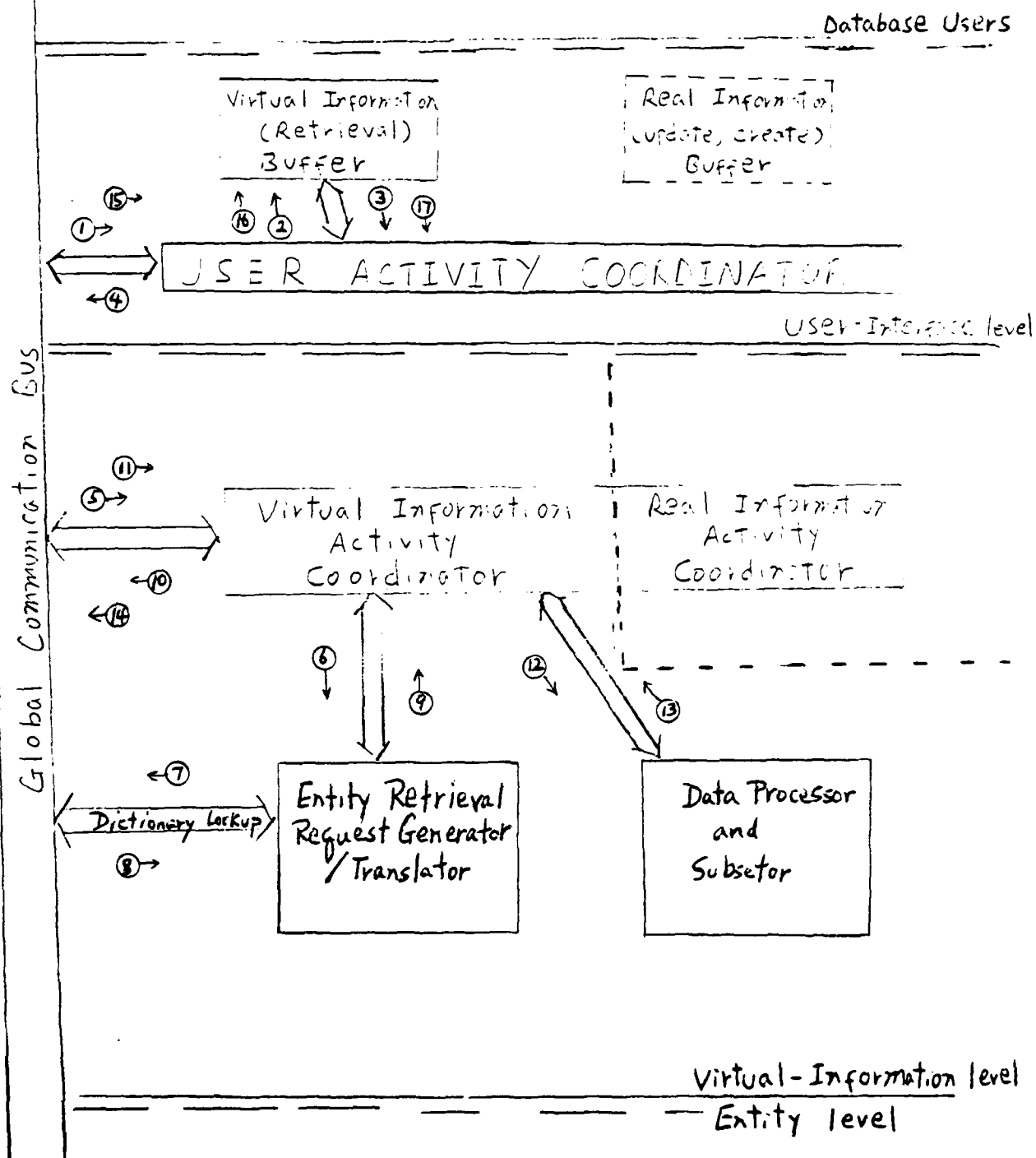
Set Operators

①	MU	multiple union
②	MI	multiple intersection
③	SU	single union
④	SI	single intersection
⑤	CS	cartesian product

{ Refer to Chapter 5 of Jameson Lee's Thesis }

Appendix (3.8.0)

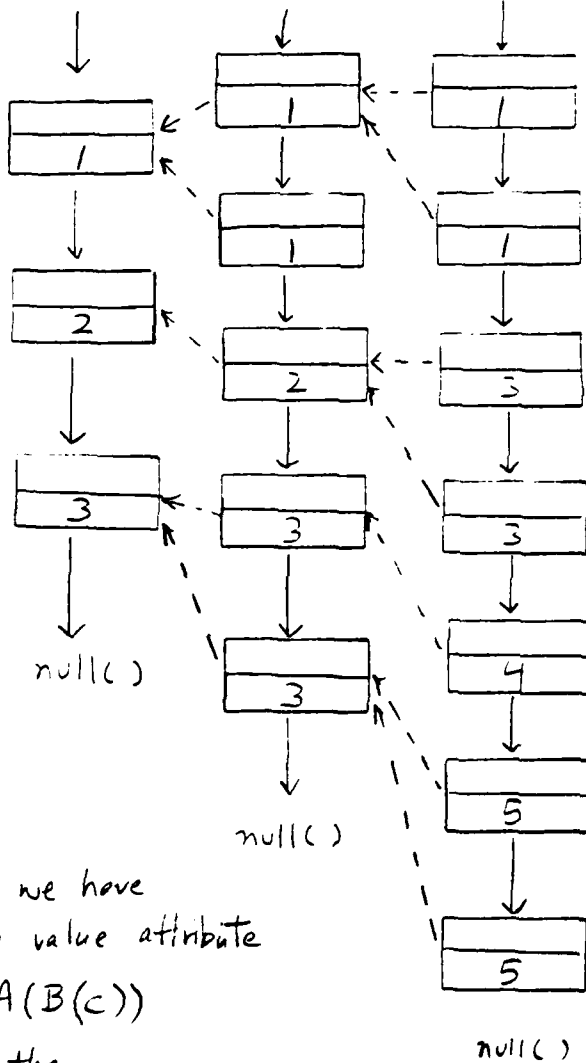
VIFI sub-level partition w/ Data flow



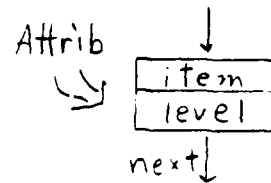
Appendix A (4.1.0)

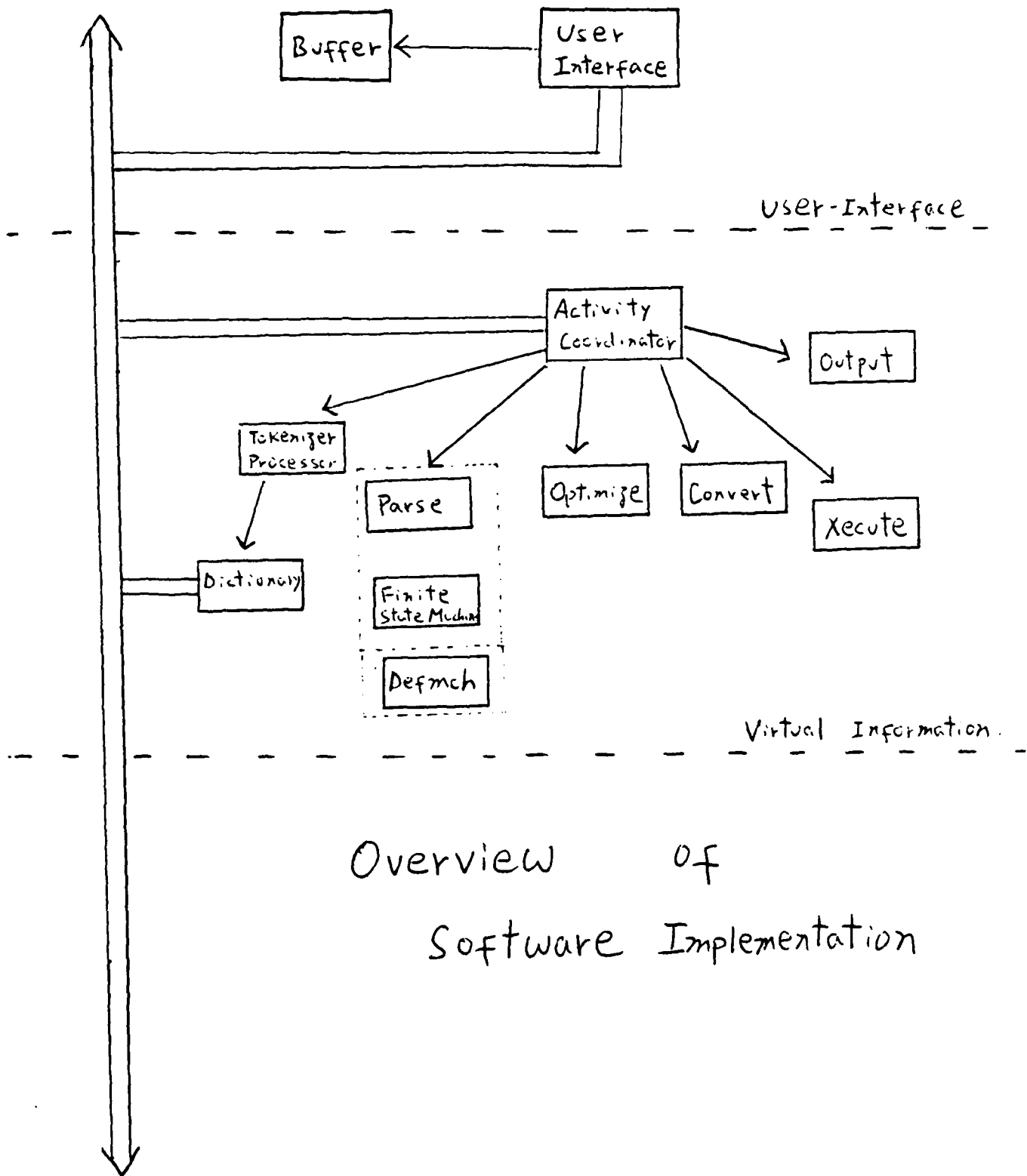
Attribute Values & Levels

index	Entity(?) Attr(n)			
1	2	3	4	
sing-occ	✓	X	X	
A-parent	0	1	2	
uses:	A	B	C	



Here we have
the value attribute
 $A(B(C))$
and the
entity attributes
 $A(B)$ and A





Overview of Software Implementation

Appendix (6.0.c)

Appendix

B

ALL00010
 ALL00020
 ALL00030
 ALL00040
 ALL00050
 ALL00060
 ALL00070
 ALL00080
 ALL00090
 ALL00100
 ALL00110
 ALL00120

*****ALL COPY*****
 MACRO INDEX SIZE
 MACH 2 5
 TOKEN 8 4
 XTREE 13 4
 ATTRIB 18 4
 ENTITY 23 16
 DICTION 40 13
 XCHNGE 54 1
 ARETE 56 32

```

DCL 1 MACH.
  2 STATE_MAP (200) FIXED.
  2 MATCH_CHAR (30) VAR.
  2 ACTION (500) CHAR (40) VAR.
  2 NEXT_STATE (500) FIXED;

DCL 1 TOKEN BASED.
  2 ITEM_CHAR (30) VAR.
  2 CLASS_CHAR (10) VAR.
  2 NEXT_PTR;

DCL 1 XTREE (1000).
  2 LABEL_CHAR (30) VAR.
  2 CHILD_FIXED.
  2 LINK_FIXED;

DCL 1 ATTRIB BASED.
  2 LEVEL_FIXED.
  2 ITEM_CHAR (50) VAR.
  2 NEXT_PTR;

DCL 1 ENTITY (0:12).
  2 NAME_CHAR (30) VAR.
  2 DEPTH_FIXED.
  2 VES_FN_CHAR (2) VAR.
  2 WHERE_FIXED.
  2 N_PARENT (2) FIXED.
  2 VES_PAR_PTR.
  2 ATTR (15).
    3 VES_KEY_BIT (1).
    3 CART_KEY_BIT (1).
    3 SING_OCC_BIT (1).
    3 A_PARENT_FIXED.
    3 USES_CHAR (30) VAR.
    3 LIST_PTR;

DCL 1 N_MAP (2) BASED.
  2 NUM (15) FIXED;

DCL 1 DICTION.
  2 CHECK.
    3 LIMIT_FIXED.
    3 LABEL (50) CHAR (50) VAR.
    3 TIMES (50) FIXED.
  2 MAIN.
    3 LIMIT_FIXED.
    3 FROM (100) CHAR (50) VAR.
    3 TO (100) CHAR (160) VAR.
  2 ADHOC.
    3 LIMIT_FIXED.
    3 FROM (50) CHAR (50) VAR.
    3 TO (50) CHAR (160) VAR;

DCL XCHANGE (20) CHAR (80) VAR;

```

```

ALLOO130
ALLOO140
ALLOO150
ALLOO160
ALLOO170
ALLOO180
ALLOO190
ALLOO200
ALLOO210
ALLOO220
ALLOO230
ALLOO240
ALLOO250
ALLOO260
ALLOO270
ALLOO280
ALLOO290
ALLOO300
ALLOO310
ALLOO320
ALLOO330
ALLOO340
ALLOO350
ALLOO360
ALLOO370
ALLOO380
ALLOO390
ALLOO400
ALLOO410
ALLOO420
ALLOO430
ALLOO440
ALLOO450
ALLOO460
ALLOO470
ALLOO480
ALLOO490
ALLOO500
ALLOO510
ALLOO520
ALLOO530
ALLOO540
ALLOO550
ALLOO560
ALLOO570
ALLOO580
ALLOO590
ALLOO600
ALLOO610
ALLOO620
ALLOO630
ALLOO640
ALLOO650
ALLOO660

```

```

COPY ARETE
DCL 1 RETE_ARG BASED (RETEP). /* WILL ALSO BE RETE_RTN */
2 CTL_INFO.
3 LEN FIXED BIN (15).
3 CBTP FIXED BIN (15) INIT (21).
3 PTR PTR.
2 NUM_ATTR FIXED.
2 NUM_COND FIXED. /* NUMBER OF CONDITIONS */
2 ENT.
3 NAME CHAR (30) VAR. /* ENTITY SET NAME */
3 DEPTH FIXED. /* FILLED IN WHEN RETURNED */
3 ATTR (NATTR REFER (RETE_ARG.NUM_ATTR)).
4 SING_OCC BIT (1). /* IF SINGLE OCCUR THEN SET */
4 A_PARENT FIXED. /* PARENT NUMBER */
4 A_USES CHAR (30) VAR. /* ATTRIBUTE NAME */
4 LIST_PTR. /* POINT TO LIST OF OCC OF THIS ATTR IF ANY */
2 COND (NCOND REFER (RETE_ARG.NUM_COND)).
3 ATTRREF FIXED. /* POINT TO ATTR IN ATTR ARRAY ABOVE */
3 NEG_BIT (1). /* NEGATION OF RELATOR */
3 REL_CHAR (1). /* '=', '<', '>' */
3 CDATA (10) CHAR (50) VAR. /* UP TO 10 'MULTI' TIMES */
2 RTN_CODE FIXED BIN.
2 NEXT_PTR PTR.
DCL RETEP_PTR.
DCL 1 RETE_RTN1 BASED. /* USED WHEN RETURNED */
2 CTL.
3 LEN FIXED BIN (31).
3 CBTP FIXED BIN (31) INIT (46).
3 PTR PTR.
2 LEVEL FIXED BIN. /* OCCUR NUMBER */
2 ITEM_CHAR (50).
2 NEXT_PTR PTR. /* POINT TO NEXT ON THE SAME ATTRIBUTE LIST */

```

```

ALL00670
ALL00680
ALL00690
ALL00700
ALL00710
ALL00720
ALL00730
ALL00740
ALL00750
ALL00760
ALL00770
ALL00780
ALL00790
ALL00800
ALL00810
ALL00820
ALL00830
ALL00840
ALL00850
ALL00860
ALL00870
ALL00880
ALL00890
ALL00900
ALL00910
ALL00920
ALL00930
ALL00940
ALL00950
ALL00960
ALL00970
ALL00980
ALL00990

```

PL/I OPTIMIZING COMPILER IDCTNRY: PROC (DICTION, COMMSTR) RETURNS (CHAR (160) VAR);

NUMBER LEV NT

```

10 0 DCTNRY: PROC (DICTION, COMMSTR) RETURNS (CHAR (160) VAR);
XINCLUDE DICTION;*****
100010 1 0 DCL 1 DICTION,
2 CHECK,
3 LIMIT FIXED,
3 LABEL (50) CHAR (50) VAR,
3 TIMES (50) FIXED,
2 MAIN,
3 LIMIT FIXED,
3 FROM (100) CHAR (50) VAR,
3 TO (100) CHAR (160) VAR,
2 ADHOC,
3 LIMIT FIXED,
3 FROM (50) CHAR (50) VAR,
3 TO (50) CHAR (160) VAR;
*****
200040 1 0 DCL (COMMSTR,FINDSTR) CHAR (160) VAR,
COMMAND CHAR (50) VAR;
*****
200070 1 0 COMMAND = GETS (COMMSTR,.,.);
200080 1 0 SELECT (COMMAND);
200090 1 1 WHEN ('FIND') DO;
200100 1 2 IF BIF IDENT (COMMSTR) THEN
IF INDEX ('(.),{,},%',SUBSTR (COMMSTR,1,1)) = 0 THEN
RETURN (COMMSTR || ',:B');
ELSE
RETURN (COMMSTR || ',:D');
200130 1 2 FINDSTR = FIND (DICTION,ADHOC);
200150 1 2 IF FINDSTR ^= 'NOT FOUND' THEN
RETURN (FINDSTR);
200180 1 2 FINDSTR = FIND (DICTION,MAIN);
200190 1 2 IF FINDSTR ^= 'NOT FOUND' THEN
RETURN (FINDSTR);
200210 1 2 RETURN (COMMSTR || ',:R');
200220 1 2 END;
200230 1 1 WHEN ('SAVE')
RETURN (SAVE (DICTION,MAIN,'MAIN'));
200250 1 1 WHEN ('ADHSAVE')
RETURN (SAVE (DICTION,ADHOC,'ADHOC'));
200270 1 1 WHEN ('REMOVE')
RETURN (REMOVE (DICTION,MAIN,'MAIN'));
200290 1 1 WHEN ('ADHREMOVE')
RETURN (REMOVE (DICTION,ADHOC,'ADHOC'));
200310 1 1 WHEN ('INITIALIZE') DO;
200320 1 2 DICTION.MAIN.LIMIT = 0;
200330 1 2 DICTION.MAIN.FROM (*) = '';
200340 1 2 DICTION.MAIN.TO (*) = '';
200350 1 2 RETURN ('DICTION_MAIN_INITIALIZED');

```

PL/I OPTIMIZING COMPILER IDCTNRY: PROC (DICTION, COMMSTR) RETURNS (CHAR (160) VAR):

NUMBER LEV NT

```

200360 1 2      END;
200370 1 1      WHEN ('ADHCNTRCLR') DO;
200380 1 2      DICTION.ADHOC.LIMIT = 0;
200390 1 2      DICTION.ADHOC.FROM (*) = '';
200400 1 2      DICTION.ADHOC.TO (*) = '';
200410 1 2      DICTION.CHECK.LIMIT = 0;
200420 1 2      DICTION.CHECK.LABEL (*) = '';
200430 1 2      DICTION.TIMES (*) = 0;
200440 1 2      RETURN ('DICTION_ADHOC_AND_COUNTER_CLEARED');
200450 1 2      END;
200460 1 1      OTHERWISE
200480 1 1      RETURN ('COMMAND_NOT_FOUND,' || COMMAND);
200490 1 1      END;
200500
DCT00360
DCT00370
DCT00380
DCT00390
DCT00400
DCT00410
DCT00420
DCT00430
DCT00440
DCT00450
DCT00460
DCT00470
DCT00480
DCT00490
DCT00500

```


PL/I OPTIMIZING COMPILER IDCTNRY: PROC (DICTION, COMMSTR) RETURNS (CHAR (160) VAR);

NUMBER LEV NT

```

200720 1 0 FIND: PROC (SUBDICTION) RETURNS (CHAR (160) VAR);
200740 2 0 DCL 1 SUBDICTION,
      2 LIMIT FIXED,
      2 FROM (.) CHAR (50) VAR,
      2 TO (.) CHAR (160) VAR;
200780 2 0 DCL (LOOKSTR, MATCHSTR, REPSTR) CHAR (160) VAR;
200800 2 0 LOOKSTR = COMMSTR;
200810 2 0 IF SUBDICTION.LIMIT > 0 THEN
      DO 1 = 1 TO SUBDICTION.LIMIT;
      MATCHSTR = SUBDICTION.FROM (1);
      IF GETS (LOOKSTR, 'N') = GETS (MATCHSTR, 'N') THEN DO;
      REPSTR = SUBDICTION.TO (1);
      IF OK TIMES THEN DO;
      DO WHILE (MATCHSTR ^= '');
      CALL REPLACE (REPSTR,
        'N', ' ', GETS (MATCHSTR, 'N'),
        GETS (LOOKSTR, 'N'));
      END;
      RETURN (REPSTR || ' ' || V');
      END;
200910 2 4
200920 2 3
200930 2 3
200940 2 2
      ELSE
      RETURN ('1*ERROR* TOO MANY FIND DEFINITIONS, '
        || COMMSTR);
200970 2 2
200980 2 1
200990 2 0
      END;
      RETURN ('NOT_FOUND');
201010 2 0
201020
201030
      END FIND;

```

DCT00720
 DCT00730
 DCT00740
 DCT00750
 DCT00760
 DCT00770
 DCT00780
 DCT00790
 DCT00800
 DCT00810
 DCT00820
 DCT00830
 DCT00840
 DCT00850
 DCT00860
 DCT00870
 DCT00880
 DCT00890
 DCT00900
 DCT00910
 DCT00920
 DCT00930
 DCT00940
 DCT00950
 DCT00960
 DCT00970
 DCT00980
 DCT00990
 DCT01000
 DCT01010
 DCT01020
 DCT01030

PL/I OPTIMIZING COMPILER IDCTNRY: PROC (DICTION, COMMSTR) RETURNS (CHAR (160) VAR);

NUMBER LEV NT

```

201040 1 0 OK_TIMES: PROC RETURNS (BIT (1));
201060 2 0 DCL (I,J) FIXED;
201080 2 0 IF DICTION.CHECK.LIMIT > 0 THEN
201100 2 1 DO I = 1 TO DICTION.CHECK.LIMIT;
201110 2 2   J = 1;
201120 2 2   I = 200;
201130 2 2   END;
201140 2 1   END;
201150 2 0 ELSE
201170 2 0   I = 0;
201180 2 1 IF I < 200 THEN DO;
201190 2 1   DICTION.CHECK.LIMIT = DICTION.CHECK.LIMIT + 1;
201200 2 1   J = DICTION.CHECK.LIMIT;
201210 2 1   DICTION.CHECK.LABEL (J) = COMMSTR;
201220 2 1   END;
201230 2 0 DICTION.CHECK_TIMES (J) = DICTION.CHECK_TIMES (J) + 1;
201250 2 0 RETURN (DICTION.CHECK_TIMES (J) <= 10);
201270 2 0 END OK_TIMES;
DCTO1040
DCTO1050
DCTO1060
DCTO1070
DCTO1080
DCTO1090
DCTO1100
DCTO1110
DCTO1120
DCTO1130
DCTO1140
DCTO1150
DCTO1160
DCTO1170
DCTO1180
DCTO1190
DCTO1200
DCTO1210
DCTO1220
DCTO1230
DCTO1240
DCTO1250
DCTO1260
DCTO1270

```

PL/I OPTIMIZING COMPILER 1DCTNRY: PROC (DICTION, COMMSTR) RETURNS (CHAR (160) VAR);

NUMBER LEV NT

```

201280 1 0 SAVE: PROC (SUBDICTION, DICTNAME) RETURNS (CHAR (160) VAR);
201300 2 0 DCL 1 SUBDICTION,
      2 LIMIT FIXED,
      2 FROM (*) CHAR (50) VAR,
      2 TO (*) CHAR (160) VAR;
201340 2 0 DCL DICTNAME CHAR (5) VAR,
      LOOKUP CHAR (50) VAR,
      (FIRSTSP, I) FIXED;
201380 2 0 LOOKUP = GETS (COMMSTR, ',');
201390 2 0 IF BIF_IDENT (LOOKUP) THEN
      RETURN ('*ERROR* ATTEMPT_TO_REDEFINE_BIF, '
      || LOOKUP);
201420 2 0 FIRSTSP = 0;
201430 2 0 IF SUBDICTION.LIMIT > 0 THEN
      DO I = 1 TO SUBDICTION.LIMIT;
      IF SUBDICTION.FROM (I) = LOOKUP THEN DO;
      SUBDICTION.TO (I) = COMMSTR;
      RETURN ('REPLACED_IN_' || DICTNAME || ', ' || LOOKUP);
      END;
      ELSE IF SUBDICTION.FROM (I) = '' & FIRSTSP = 0 THEN
      FIRSTSP = 1;
      END;
201510 2 1 END;
201520 2 0 IF FIRSTSP = 0 THEN DO;
201530 2 1 SUBDICTION.LIMIT = SUBDICTION.LIMIT + 1;
201540 2 1 FIRSTSP = SUBDICTION.LIMIT;
201550 2 1 END;
201560 2 0 SUBDICTION.FROM (FIRSTSP) = LOOKUP;
201570 2 0 SUBDICTION.TO (FIRSTSP) = COMMSTR;
201580 2 0 RETURN ('SAVED_IN_' || DICTNAME || ', ' || LOOKUP);
201600 2 0 END SAVE;

```

DCTO1280
 DCTO1290
 DCTO1300
 DCTO1310
 DCTO1320
 DCTO1330
 DCTO1340
 DCTO1350
 DCTO1360
 DCTO1370
 DCTO1380
 DCTO1390
 DCTO1400
 DCTO1410
 DCTO1420
 DCTO1430
 DCTO1440
 DCTO1450
 DCTO1460
 DCTO1470
 DCTO1480
 DCTO1490
 DCTO1500
 DCTO1510
 DCTO1520
 DCTO1530
 DCTO1540
 DCTO1550
 DCTO1560
 DCTO1570
 DCTO1580
 DCTO1590
 DCTO1600
 DCTO1610
 DCTO1620

PL/1 OPTIMIZING COMPILER 1DCTNRY: PROC (DICTION, COMMSTR) RETURNS (CHAR (160) VAR);

NUMBER LEV NT

```

201630 1 0 REMOVE. PROC (SUBDICTION, DICTNAME) RETURNS (CHAR (160) VAR);
201650 2 0 DCL 1 SUBDICTION,
      2 LIMIT FIXED,
      2 FROM (*) CHAR (50) VAR,
      2 TO (*) CHAR (160) VAR;
201690 2 0 DCL DICTNAME CHAR (5) VAR,
      1 FIXED;
201720 2 0 IF BIF_IDENT (COMMSTR) THEN
      RETURN ('*ERROR* ATTEMPT TO REMOVE_BIF,' || COMMSTR);
201740 2 0 IF SUBDICTION.LIMIT > 0 THEN
      DO I = 1 TO SUBDICTION.LIMIT;
      IF SUBDICTION.FROM (I) = COMMSTR THEN DO;
      SUBDICTION.FROM (I) = '';
      RETURN ('REMOVED_FROM_' || DICTNAME || ',' || COMMSTR);
      END;
      END;
201820 2 0 RETURN ('*ERROR* NOT_FOUND_IN_' || DICTNAME || ',' || COMMSTR);
201840 2 0 END REMOVE;
DCTO1630
DCTO1640
DCTO1650
DCTO1660
DCTO1670
DCTO1680
DCTO1690
DCTO1700
DCTO1710
DCTO1720
DCTO1730
DCTO1740
DCTO1750
DCTO1760
DCTO1770
DCTO1780
DCTO1790
DCTO1800
DCTO1810
DCTO1820
DCTO1830
DCTO1840
DCTO1850
DCTO1860

```

PL/J OPTIMIZING COMPILER 1DCTNRY : PROC (DICTIO,J, COMMSTR) RETURNS (CHAR (160) VAR);

NUMBER LEV NT

```

201870 1 0 REPLACE: PROC (RSTR, FROMSTR, TOSTR);
201890 2 0 DCL (RSTR, REPLSTR, FROMSTR, TOSTR) CHAR (160) VAR,
      I FIXED;
201920 2 0 REPLSTR = '';
201930 2 0 I = INDEX (RSTR, FROMSTR);
201940 2 0 DO WHILE (I ^= 0);
201950 2 1   REPLSTR = REPLSTR || SUBSTR (RSTR, I, I - 1) || TOSTR;
201960 2 1   RSTR = SUBSTR (RSTR, I + LENGTH (FROMSTR));
201970 2 1   I = INDEX (RSTR, FROMSTR);
201980 2 1   END;
201990 2 0 RSTR = REPLSTR || RSTR;
202010 2 0 END REPLACE;
DCTO1870
DCTO1890
DCTO1890
DCTO1900
DCTO1910
DCTO1920
DCTO1930
DCTO1940
DCTO1950
DCTO1960
DCTO1970
DCTO1980
DCTO1990
DCTO2000
DCTO2010
DCTO2020
DCTO2030

```

PL/I OPTIMIZING COMPILER 1DCTNRY: PROC (DICTION, COMMSTR) RETURNS (CHAR (160) VAR);

NUMBER LEV NT

```

202040 1 0 GETS: PROC (LIST, TERM_ITEM) RETURNS (CHAR (160) VAR);
202060 2 0 DCL LIST CHAR (*) VAR,
      TERM_ITEM CHAR (1),
      RTN_LIST CHAR (160) VAR,
      I FIXED;
202110 2 0 I = INDEX (LIST, TERM_ITEM);
202120 2 0 IF I = 0 THEN DO;
202130 2 1 RTN_LIST = LIST;
202140 2 1 LIST = ',';
202150 2 1 END;
202160 2 0 ELSE DO;
202170 2 1 RTN_LIST = SUBSTR (LIST, I, I - 1);
202180 2 1 LIST = SUBSTR (LIST, I + 1);
202190 2 1 END;
202200 2 0 RETURN (RTN_LIST);
202220 2 0 END GETS;

202250 1 0 END DCTNRY;
DCT02040
DCT02050
DCT02060
DCT02070
DCT02080
DCT02090
DCT02100
DCT02110
DCT02120
DCT02130
DCT02140
DCT02150
DCT02160
DCT02170
DCT02180
DCT02190
DCT02200
DCT02210
DCT02220
DCT02230
DCT02240
DCT02250
DCT02260
DCT02270
DCT02280
DCT02290
DCT02300

```

PL/I OPTIMIZING COMPILER 1DEFMCH: PROC (MACH);

```

NUMBER LEV NT
10 0 0 DEFMCH: PROC (MACH);
    %INCLUDE MACH;*****
100010 1 0 DCL 1 MACH,
    2 STATE_MAP (200) FIXED,
    2 MATCH (500) CHAR (30) VAR,
    2 ACTION (500) CHAR (40) VAR,
    2 NEXT_STATE (500) FIXED;
    *****
200040 1 0 DCL MACHIN FILE RECORD;
200050 1 0 DCL FREE_LOC FIXED INIT (1);
200060 1 0 DCL LINE_CHAR (80),
    (LINE1,TYPE) CHAR (80) VAR;

200090 1 0 ON ENDFILE (MACHIN) GOTO COMPLETE;

200110 1 0 OPEN FILE (MACHIN) INPUT;
200120 1 0 DO WHILE ('1'B);
200130 1 1 READ FILE (MACHIN) INTO (LINE);
200140 1 1 LINE1 = LINE;
200150 1 1 TYPE = GETS (LINE1, '\');
200160 1 1 IF TYPE = 'S' THEN
    MACH.STATE_MAP (FIXED (GETS (LINE1, '\'))) = FREE_LOC;
    ELSE IF TYPE = 'T' THEN DO;
200180 1 1 MACH.MATCH (FREE_LOC) = GETS (LINE1, '\');
200190 1 2 MACH.ACTION (FREE_LOC) = GETS (LINE1, '\');
200200 1 2 MACH.NEXT_STATE (FREE_LOC) = FIXED (GETS (LINE1, '\'));
200210 1 2 FREE_LOC = FREE_LOC + 1;
200220 1 2 END;
200230 1 1 END;
200240 1 1
200250

```

```

DEFO0010
DEFO0020
DEFO0030
MAC00010
MAC00020
MAC00030
MAC00040
MAC00050
DEFO0030
DEFO0040
DEFO0050
DEFO0060
DEFO0070
DEFO0080
DEFO0090
DEFO0100
DEFO0110
DEFO0120
DEFO0130
DEFO0140
DEFO0150
DEFO0160
DEFO0170
DEFO0180
DEFO0190
DEFO0200
DEFO0210
DEFO0220
DEFO0230
DEFO0240
DEFO0250

```

PL/I OPTIMIZING COMPILER 1DEFNCH: PROC (MACH);

NUMBER LEV NT

```

200260 1 0 GETS: PROC (LIST,TERM_ITEM) RETURNS (CHAR (80) VAR);
200280 2 0 DCL LIST CHAR (*) VAR,
      TERM_ITEM CHAR (1),
      RTN_LIST CHAR (80) VAR,
      I FIXED;
200330 2 0 I = INDEX (LIST,TERM_ITEM);
200340 2 0 IF I = 0 THEN DO;
200350 2 1 RTN_LIST = LIST;
200360 2 1 LIST = ',';
200370 2 1 END;
200380 2 0 ELSE DO;
200390 2 1 RTN_LIST = SUBSTR (LIST,I - 1);
200400 2 1 LIST = SUBSTR (LIST,I + 1);
200410 2 1 END;
200420 2 0 RETURN (RTN_LIST);
200440 2 0 END GETS;
200460 1 0 COMPLETE:
      PUT SKIP LIST ('MACHINE DEFINITION COMPLETE');
200480 1 0 CLOSE FILE (MACHIN);

```

```

DEFO0260
DEFO0270
DEFO0280
DEFO0290
DEFO0300
DEFO0310
DEFO0320
DEFO0330
DEFO0340
DEFO0350
DEFO0360
DEFO0370
DEFO0380
DEFO0390
DEFO0400
DEFO0410
DEFO0420
DEFO0430
DEFO0440
DEFO0450
DEFO0460
DEFO0470
DEFO0480
DEFO0490

```

1DEFMCH: PROC (MACH):

PL/I OPTIMIZING COMPILER

NUMBER LEV NT

200500 1 0 END DEFMCH:

DEFO0500

PL/I OPTIMIZING COMPILER 1PARSE: PROC (MACH,TOKENS_PTR,XTREE,XCHNGE,ENTITY,DEBUG);

NUMBER LEV NT

```

10 0 0 PARSE: PROC (MACH,TOKENS_PTR,XTREE,XCHNGE,ENTITY,DEBUG);
      %INCLUDE MACH;*****
100010 1 0 DCL 1 MACH,
      2 STATE_MAP (200) FIXED,
      2 MATCH_ (500) CHAR (30) VAR,
      2 ACTION (500) CHAR (40) VAR,
      2 NEXT_STATE (500) FIXED;
      *****
300010 1 0 DCL 1 TOKEN BASED,
      2 ITEM CHAR (30) VAR,
      2 CLASS CHAR (10) VAR,
      2 NEXT_PTR;
      *****
500010 1 0 DCL 1 XTREE (1000),
      2 LABEL CHAR (30) VAR,
      2 CHILD FIXED,
      2 LINK FIXED;
      *****
600010 1 0 DCL XCHNGE;*****
700010 1 0 DCL 1 ENTITY (0:12),
      2 NAME CHAR (30) VAR,
      2 DEPTH FIXED,
      2 VES_FN CHAR (2) VAR,
      2 WHERE FIXED,
      2 N_PARENT (2) FIXED,
      2 VES_PAR_PTR,
      2 ATTR (15),
      3 VES_KEY BIT (1),
      3 CART_KEY BIT (1),
      3 SING_OCC BIT (1),
      3 A_PARENT FIXED,
      3 USES CHAR (30) VAR,
      3 LIST_PTR;
700150 1 0 DCL 1 N_MAP (2) BASED,
      2 NUM (15) FIXED;
      *****
800090 1 0 DCL TOKENS_PTR PTR;
800100 1 0 DCL DEBUG_BIT (1);
800110 1 0 DCL (XTREE_LOC,XCH_LOC,CUR_ES) FIXED;
800120 1 0 DCL STACK (3,0:100) CHAR (30) VAR,
      TOS (3) FIXED;
800140 1 0 DCL VES (0:9) FIXED,

```

PL/I OPTIMIZING COMPILER IPARSE: PROC (MACH, TOKENS_PTR, XTREE, XCHANGE, ENTITY, DEBUG);

NUMBER LEV NT

```

      VES_LOC FIXED;
      800160 1 0 DCL (STATE_NUM,N) FIXED,
      800180 1 0 DCL (INPUT_PTR,FREE_PTR) PTR;
      800190 1 0 DCL ERROR_FILE OUTPUT;
      800200 1 0 OPEN FILE (ERROR);
      800210 1 0 XTREE (*).LABEL = '';
      800220 1 0 XTREE (*).CHILD = 0;
      800230 1 0 XTREE (*).LINK = 0;
      800240 1 0 XTREE_LOC = 1;
      800250 1 0 XCH_LOC = 0;
      800260 1 0 ENTITY (*).NAME = '';
      800270 1 0 ENTITY (*).DEPTH = 0;
      800280 1 0 ENTITY (*).VES_FN = '';
      800290 1 0 ENTITY (*).WHERE = 0;
      800300 1 0 ENTITY (*).N_PARENT (*) = 0;
      800310 1 0 ENTITY (*).VES_PAR = NULL ();
      800320 1 0 ENTITY (*).ATTR (*) .VES_KEY = 'O'B;
      800330 1 0 ENTITY (*).ATTR (*) .CART_KEY = 'O'B;
      800340 1 0 ENTITY (*).ATTR (*) .SING_OCC = '1'B;
      800350 1 0 ENTITY (*).ATTR (*) .A_PARENT = 0;
      800360 1 0 ENTITY (*).ATTR (*) .USES = '';
      800370 1 0 ENTITY (*).ATTR (*) .LIST = NULL ();
      800380 1 0 CUR_ES = 0;
      800390 1 0 STACK (*,O) = 'ØBDS';
      800400 1 0 TOS (*) = 0;
      800410 1 0 VES (*) = 0;
      800420 1 0 VES_LOC = -1;
      800430 1 0 INPUT_PTR = TOKENS_PTR;
      800440 1 0 STATE_NUM = 1;
PAR00150
PAR00160
PAR00170
PAR00180
PAR00190
PAR00200
PAR00210
PAR00220
PAR00230
PAR00240
PAR00250
PAR00260
PAR00270
PAR00280
PAR00290
PAR00300
PAR00310
PAR00320
PAR00330
PAR00340
PAR00350
PAR00360
PAR00370
PAR00380
PAR00390
PAR00400
PAR00410
PAR00420
PAR00430
PAR00440
PAR00450

```

PL/I OPTIMIZING COMPILER 1PARSE: PROC (MACH, TOKENS_PTR, XTREE, XCHNGE, ENTITY, DEBUG);

NUMBER LEV NT

```

800460 1 0 DO WHILE (STATE_NUM ^= 0);
800470 1 1 IF DEBUG THEN
      PUT SKIP (2) EDIT
      ('TRANSIT: STATE = ', STATE_NUM,
      'INPUT =$', INPUT_PTR -> TOKEN.ITEM, '$$',
      'CLASS =$', INPUT_PTR -> TOKEN.CLASS, '$$',
      'STK#1 =$', STACK (1,TOS (1)), '$$',
      'STK#2 =$', STACK (2,TOS (2)), '$$')
      (A,F(5),4(SKIP,A,A,A));
800550 1 1 DO N = MACH.STATE_MAP (STATE_NUM)
      TO MACH.STATE_MAP (STATE_NUM + 1) - 1;
800570 1 2 IF MATCHING (MACH.MATCH (N)) THEN DO;
800580 1 3 STATE_NUM = ACTING (MACH.ACTION (N), MACH.NEXT_STATE (N));
800590 1 3 N = 10000;
800600 1 3 END;
800610 1 2 END;
800620 1 1 IF N < 10000 THEN DO;
      /*TRANSITIONS NOT CLOSED*/
      PUT FILE (ERROR) SKIP EDIT
      ('CLOSURE ON STATE = ', STATE_NUM,
      'INPUT =$', INPUT_PTR -> TOKEN.ITEM, '$$',
      'CLASS =$', INPUT_PTR -> TOKEN.CLASS, '$$',
      'STK#1 =$', STACK (1,TOS (1)), '$$',
      'STK#2 =$', STACK (2,TOS (2)), '$$')
      (A,F(5),4(SKIP,A,A,A));
800710 1 2 IF DEBUG THEN
      PUT SKIP LIST ('PREMATURE TERMINATION');
800730 1 2 STATE_NUM = 0;
800740 1 2 END;
800750 1 1 END;
800760 1 0 CLOSE FILE (ERROR);
PAR00460
PAR00470
PAR00480
PAR00490
PAR00500
PAR00510
PAR00520
PAR00530
PAR00540
PAR00550
PAR00560
PAR00570
PAR00580
PAR00590
PAR00600
PAR00610
PAR00620
PAR00630
PAR00640
PAR00650
PAR00660
PAR00670
PAR00680
PAR00690
PAR00700
PAR00710
PAR00720
PAR00730
PAR00740
PAR00750
PAR00760
PAR00770
PAR00780

```

PL/I OPTIMIZING COMPILER 1PARSE: PROC (MACH,TOKENS_PTR,XTREE,XCHANGE,ENTITY,DEBUG);

NUMBER LEV NT

```

800790 1 0 MATCHING: PROC (MATCH_LIST) RETURNS (BIT (1));
800810 2 0 DCL MATCH_LIST CHAR (30) VAR,
      (REST_LIST,PROCESS_LIST) CHAR (30) VAR,
      I FIXED;
800850 2 0 IF MATCH_LIST = '' THEN
      RETURN ('1'B);
800870 2 0 REST_LIST = MATCH_LIST;
800880 2 0 DO WHILE (REST_LIST ^= '');
800890 2 1 PROCESS_LIST = GETS (REST_LIST, '');
800900 2 1 IF DEBUG THEN
      PUT SKIP EDIT
      ('MATCHING =$',PROCESS_LIST,'$$')
      (3 (A));
800940 2 1 IF FOUND (INPUT_PTR -> TOKEN.ITEM,
      INPUT_PTR -> TOKEN.CLASS) THEN
      IF FOUND (STACK (1,TOS (1)), '') THEN
      IF FOUND (STACK (2,TOS (2)), '') THEN
      RETURN ('1'B);
800990 2 1 END;
801000 2 0 RETURN ('0'B);
PAR00790
PAR00800
PAR00810
PAR00820
PAR00830
PAR00840
PAR00850
PAR00860
PAR00870
PAR00880
PAR00890
PAR00900
PAR00910
PAR00920
PAR00930
PAR00940
PAR00950
PAR00960
PAR00970
PAR00980
PAR00990
PAR01000
PAR01010

```

PL/I OPTIMIZING COMPILER 1PARSE: PROC (MACH,TOKENS_PTR,XTREE,XCHANGE,ENTITY,DEBUG);

```

NUMBER LEV NT
801020 2 0 FOUND: PROC (FIND_ITEM,FIND_CLASS) RETURNS (BIT (1));
801040 3 0 DCL (FIND_ITEM,MATCH_ITEM,MATCH_FIND) CHAR (30) VAR,
      FIND_CLASS CHAR (10) VAR;
801070 3 0 MATCH_ITEM = GETS (PROCESS_LIST, '');
801080 3 0 IF DEBUG THEN
      PUT SKIP EDIT
      ('FIND_MATCH = $',MATCH_ITEM,'$$',
      'FIND_ITEM = $',FIND_ITEM,'$$',
      'FIND_CLASS = $',FIND_CLASS,'$$')
      (A,A,A,2(SKIP,A,A,A));
801140 3 0 SELECT (MATCH_ITEM);
801150 3 1 WHEN ('',FIND_CLASS) RETURN ('1'B);
801160 3 1 WHEN ('@CONC') MATCH_ITEM = '';
801170 3 1 WHEN ('@CONC>') MATCH_ITEM = '+,-,*,/,1';
801180 3 1 WHEN ('@SUMOP') MATCH_ITEM = '+,-';
801190 3 1 WHEN ('@SUMOP>') MATCH_ITEM = '+,-,*,/,1';
801200 3 1 WHEN ('@MULTOP') MATCH_ITEM = '*,/';
801210 3 1 WHEN ('@MULTOP>') MATCH_ITEM = '*,/,**';
801220 3 1 WHEN ('@VIRT') MATCH_ITEM = 'VO,V1,V2,V3,V4,V5,V6,V7,V8,V9';
801230 3 1 WHEN ('@SETOP') MATCH_ITEM = 'MU,M1,SU,S1,CS';
801240 3 1 WHEN ('@REL') MATCH_ITEM = '>,<=';
801250 3 1 WHEN ('@CMP') MATCH_ITEM = 'AND,OR,XOR';
801260 3 1 OTHERWISE;
801270 3 1 END;
801280 3 0 MATCH_FIND = FIND_ITEM;
801290 3 0 MATCH_FIND = GETS (MATCH_FIND, '');
801300 3 0 DO WHILE (MATCH_ITEM ^= '');
801310 3 1 IF DEBUG THEN
      PUT SKIP EDIT ('SEP_ITEM = $',MATCH_ITEM,'$$')
      (3 (A));
801340 3 1 IF GETS (MATCH_ITEM, '') = MATCH_FIND THEN
      RETURN ('1'B);
801360 3 1 END;
801370 3 0 RETURN ('0'B);
801390 3 0 END FOUND;
801410 2 0 END MATCHING;
PAR01020
PAR01030
PAR01040
PAR01050
PAR01060
PAR01070
PAR01080
PAR01090
PAR01100
PAR01110
PAR01120
PAR01130
PAR01140
PAR01150
PAR01160
PAR01170
PAR01180
PAR01190
PAR01200
PAR01210
PAR01220
PAR01230
PAR01240
PAR01250
PAR01260
PAR01270
PAR01280
PAR01290
PAR01300
PAR01310
PAR01320
PAR01330
PAR01340
PAR01350
PAR01360
PAR01370
PAR01380
PAR01390
PAR01400
PAR01410
PAR01420

```

PL/I OPTIMIZING COMPILER IPARSE: PROC (MACH,TOKENS_PTR,XTREE,XCHANGE,ENTITY,DEBUG);

NUMBER LEV NT

```

801430 1 0 ACTING: PROC (ACTION_LIST,NXT_STATE) RETURNS (FIXED);
801450 2 0 DCL (ACTION_LIST,REST_LIST) CHAR (40) VAR,
      (PROCESS_LIST,ACTION_NAME,GARBAGE,RTN_STATE) CHAR (30) VAR,
      NXT_STATE FIXED;

801490 2 0 REST_LIST = ACTION_LIST;
801500 2 0 DO WHILE (REST_LIST ^= '');
801510 2 1 PROCESS_LIST = GETS (REST_LIST,1);
801520 2 1 IF DEBUG THEN
      PUT SKIP EDIT
        ('ACTING =$',PROCESS_LIST,'$$')
        (3 (A));
801560 2 1 ACTION_NAME = GETS (PROCESS_LIST,1);
801570 2 1 SELECT (ACTION_NAME);
801580 2 2 WHEN ('PUSH','P')
      CALL PUSH (FIXED (GETS (PROCESS_LIST,1)),
        CHANGE (PROCESS_LIST));
      WHEN ('POP')
      GARBAGE = POP (FIXED (PROCESS_LIST));
      WHEN ('DEL') DO;
      FREE_PTR = INPUT_PTR;
      INPUT_PTR = INPUT_PTR -> TOKEN.NEXT;
      FREE FREE_PTR -> TOKEN;
      END;
      WHEN ('GENENT')
      CALL GENENT;
      WHEN ('VIRTX')
      CALL VIRTX;
      WHEN ('VIRTA')
      CALL VIRTA;
      WHEN ('ATTWHR')
      ENTITY (CUR_ES) WHERE = FIXED (SUBSTR (POP (1),2));
      WHEN ('GENNODE','GD')
      CALL GENNODE;
      WHEN ('INDX')
      CALL EXCH (':IND');
      WHEN ('MULX')
      CALL EXCH (':MUL');
      WHEN ('ADDON')
      CALL ADDON;
      OTHERWISE
      PUT FILE (ERRDR) SKIP LIST
        ('BAD ACTION: ',ACTION_NAME);
      END;
801870 2 2 END;
801880 2 1 IF
801890 2 0 IF NXT_STATE < 0 THEN DO;
801900 2 1 RTN_STATE = POP (2);
801910 2 1 GARBAGE = GETS (RTN_STATE,1);

```

PL/I OPTIMIZING COMPILER IPARSE: PROC (MACH,TOKENS_PTR,XTREE,XCHNGE,ENTITY,DEBUG);

NUMBER LEV NT

801920 2 1 RETURN (FIXED (RTN_STATE));
 801930 2 1 END;
 801940 2 0 ELSE
 RETURN (NXT_STATE);
 801970 2 0 END ACTING;

PARO1920
 PARO1930
 PARO1940
 PARO1950
 PARO1960
 PARO1970
 PARO1980

PL/I OPTIMIZING COMPILER 1PARSE: PROC (MACH,TOKENS_PTR,XTREE,XCHNGE,ENTITY,DEBUG);

NUMBER LEV NT

```

801990 1 0 GETS: PROC (LIST,TERM_ITEM) RETURNS (CHAR (30) VAR);
802010 2 0 DCL LIST CHAR (*) VAR.
      TERM_ITEM CHAR (1),
      RTN_LIST CHAR (30) VAR,
      I FIXED;
802060 2 0 I = INDEX (LIST,TERM_ITEM);
802070 2 0 IF I = 0 THEN DO;
802080 2 1 RTN_LIST = LIST;
802090 2 1 LIST = ',';
802100 2 1 END;
802110 2 0 ELSE DO;
802120 2 1 RTN_LIST = SUBSTR (LIST,I - 1);
802130 2 1 LIST = SUBSTR (LIST,I + 1);
802140 2 1 END;
802150 2 0 RETURN (RTN_LIST);
802170 2 0 END GETS;
PAR01990
PAR02000
PAR02010
PAR02020
PAR02030
PAR02040
PAR02050
PAR02060
PAR02070
PAR02080
PAR02090
PAR02100
PAR02110
PAR02120
PAR02130
PAR02140
PAR02150
PAR02160
PAR02170
PAR02180

```

PL/I OPTIMIZING COMPILER 1PARSE: PROC (MACH,TOKENS_PTR,XTREE,XCHNGE,ENTITY,DEBUG);

NUMBER LEV NT

```

802190 1 0 PUSH: PROC (STACK_NUM,STACK_ITEM);
802210 2 0 DCL STACK_NUM FIXED,
      STACK_ITEM CHAR (30) VAR;
802240 2 0 TOS (STACK_NUM) = TOS (STACK_NUM) + 1;
802250 2 0 STACK (STACK_NUM,TOS (STACK_NUM)) = STACK_ITEM;
802270 2 0 END PUSH;

802290 1 0 POP: PROC (STACK_NUM) RETURNS (CHAR (30) VAR);
802310 2 0 DCL STACK_NUM FIXED;
802330 2 0 TOS (STACK_NUM) = TOS (STACK_NUM) - 1;
802340 2 0 RETURN (STACK (STACK_NUM,TOS (STACK_NUM) + 1));
802360 2 0 END POP;

802380 1 0 CHANGE: PROC (CH_ITEM) RETURNS (CHAR (30) VAR);
802400 2 0 DCL (CH_ITEM,INP_SOURCE) CHAR (30) VAR;
802420 2 0 IF INDEX (CH_ITEM,'e') = 0 THEN
      RETURN (CH_ITEM);
802440 2 0 INP_SOURCE = GETS (CH_ITEM,'e');
802450 2 0 SELECT (INP_SOURCE);
802460 2 1 WHEN ('1')
      CH_ITEM = INPUT_PTR -> TOKEN.ITEM || CH_ITEM;
      WHEN ('C')
      CH_ITEM = INPUT_PTR -> TOKEN.ITEM
      || CH_ITEM || INPUT_PTR -> TOKEN.CLASS;
      WHEN ('1','2','3')
      CH_ITEM = STACK (FIXED (INP_SOURCE),TOS (FIXED (INP_SOURCE)))
      || CH_ITEM;
      OTHERWISE;
802540 2 1
802550 2 1 END;
802560 2 0 RETURN (CH_ITEM);
802580 2 0 END CHANGE;

```

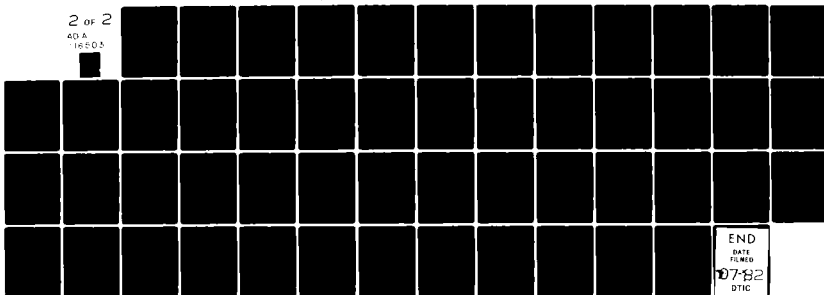
AD-A116 503

ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA F/G 9/2
VIRTUAL INFORMATION FACILITY OF THE INFOPLEX SOFTWARE TEST VEH1--ETC(U)
MAY 82 P LU
M010-8205-11 N00039-81-C-0663
NL

UNCLASSIFIED

2 OF 2

ADA
16503



PL/I OPTIMIZING COMPILER IPARSE: PROC (MACH,TOKENS_PTR,XTREE,XCHNGE,ENTITY,DEBUG);

NUMBER LEV NT

```

802600 1 0 GENENT: PROC;
802620 2 0 DCL (ENT_ITEM,ENT_NAME) CHAR (30) VAR,
      (NUM_KEYS,1) FIXED;

802650 2 0 CUR_ES = CUR_ES + 1;
802660 2 0 ENTITY (CUR_ES).VES_FN = 'S';
802670 2 0 ENT_ITEM = POP (2);
802680 2 0 IF ENT_ITEM = ')' THEN DO;
802690 2 1 ENTITY (CUR_ES).N_PARENT (2) = FIXED (SUBSTR (POP (1),2));
802700 2 1 NUM_KEYS = 0;
802710 2 1 DO WHILE (ENT_ITEM ^= '(');
802720 2 2 CALL PUSH (3,POP (1));
802730 2 2 NUM_KEYS = NUM_KEYS + 1;
802740 2 2 ENT_ITEM = POP (2);
802750 2 2 END;
802760 2 1 DO I = 1 TO NUM_KEYS;
802770 2 2 CALL GENATTR (POP (3));
802780 2 2 END;
802790 2 1 ENTITY (CUR_ES).VES_FN = POP (2);
802800 2 1 ENT_ITEM = POP (2);
802810 2 1 END;
802820 2 0 ENT_NAME = POP (1);
802830 2 0 IF SUBSTR (ENT_NAME,1,1) = ':' THEN
      ENTITY (CUR_ES).N_PARENT (1) = FIXED (SUBSTR (ENT_NAME,2));
      ELSE DO;
802850 2 0 ENTITY (CUR_ES).NAME = ENT_NAME;
802860 2 1 ENTITY (CUR_ES).VES_FN = 'R';
802870 2 1 END;
802880 2 0 CALL PUSH (1,':' || CHAR (CUR_ES));
802890 2 0 END GENENT;
802910 2 0
802920

```

PL/I OPTIMIZING COMPILER

NUMBER LEV NT

```
802930      1 0 VIRTX: PROC;
```

802950 2 0 DCL GARBAGE CHAR (30) VAR:

```

802970 2 0 STACK (1,TOS (1)) =
      , , , , , VES (FIXED (SUBSTR (STACK (1,TOS (1)),2)))
      C C C C C / *MUST BE "*"*/

```

803010 2 0 END VIRTX:

803030 1 0 VIRTA: PROC:

```

803050      2 0 VES_LOC = VES_LOC + 1:
803050      2 0 VES (VES_LOC) = FIXED (SUBSTR (POP (1),2)):
803050      2 0

```

803080 2 0 END VIRTA:

PAR02930
PAR02940
PAR02950
PAR02960
PAR02970
PAR02980
PAR02990
PAR03000
PAR03010
PAR03020

PAR03030
PAR03040
PAR03050
PAR03060
PAR03070
PAR03080
PAR03090

PL/I OPTIMIZING COMPILER IPARSE: PROC (MACH,TOKENS_PTR,XTREE,XCHNGE.ENTITY,DEBUG);

NUMBER LEV NT

```

803100 1 0 GENNODE: PROC;
803120 2 0 DCL (OP_ITEM,ARG_ITEM,NODE_LOCATION) CHAR (30) VAR,
      I FIXED;
803150 2 0 OP_ITEM = POP (2);
803160 2 0 NODE_LOCATION = ' '; !! CHAR (XTREE_LOC);
803170 2 0 CALL PUTX (GETS (OP_ITEM,':'),FIXED (OP_ITEM));
803180 2 0 IF FIXED (OP_ITEM) > 0 THEN /*O IF NO ARG OP*/
      DO I = 1 TO FIXED (OP_ITEM);
803200 2 1 ARG_ITEM = POP (1);
803210 2 1 IF SUBSTR (ARG_ITEM,1,1) = ':' THEN
      XTREE (FIXED (POP (3))).LINK = FIXED (SUBSTR (ARG_ITEM,2));
      ELSE DO;
803230 2 1 XTREE (FIXED (POP (3))).LINK = XTREE_LOC;
803240 2 2 CALL GENATTR (ARG_ITEM);
803250 2 2 CALL PUTX (ARG_ITEM,0);
803260 2 2 END;
803270 2 2 END;
803280 2 1 END;
803290 2 0 CALL PUSH (1,NODE_LOCATION);

803310 2 0 PUTX: PROC (PUTX_ITEM,CHILD_NUM);
803330 3 0 DCL PUTX_ITEM CHAR (30) VAR,
      (CHILD_NUM,1) FIXED;
803360 3 0 XTREE (XTREE_LOC).LABEL = PUTX_ITEM;
803370 3 0 XTREE (XTREE_LOC).CHILD = CHILD_NUM;
803380 3 0 IF CHILD_NUM > 0 THEN
      DO I = 0 TO CHILD_NUM - 1;
803400 3 1 CALL PUSH (3,CHAR (XTREE_LOC + 1)); /*WATCH FOR CONV ERRORS*/
803410 3 1 END;
803420 3 0 XTREE_LOC = XTREE_LOC + MAX (CHILD_NUM,1);
803440 3 0 END PUTX;
803460 2 0 END GENNODE;
PAR03100
PAR03110
PAR03120
PAR03130
PAR03140
PAR03150
PAR03160
PAR03170
PAR03180
PAR03190
PAR03200
PAR03210
PAR03220
PAR03230
PAR03240
PAR03250
PAR03260
PAR03270
PAR03280
PAR03290
PAR03300
PAR03310
PAR03320
PAR03330
PAR03340
PAR03350
PAR03360
PAR03370
PAR03380
PAR03390
PAR03400
PAR03410
PAR03420
PAR03430
PAR03440
PAR03450
PAR03460
PAR03470

```

PL/I OPTIMIZING COMPILER IPARSE: PROC (MACH, TOKENS_PTR, XTREE, XCHNGE, ENTITY, DEBUG);

NUMBER LEV NT

```

803480 1 0 GENATTR: PROC (ATTR_ITEM);
/*MODIFIES ATTR_ITEM*/
803510 2 0 DCL (ATTR_ITEM, GEN_ITEM, VES_CART_KEY, NM_ITEM) CHAR (30) VAR,
      NAME_ITEM CHAR (80) VAR,
      (PARENT_NUM, I) FIXED,
      (VIRTUAL, CARTESIAN) BIT (1);
803560 2 0 GEN_ITEM = ATTR_ITEM;
803570 2 0 NAME_ITEM = GETS (GEN_ITEM, '');
803580 2 0 IF GEN_ITEM = 'IND' THEN
      NAME_ITEM = XCHNGE (FIXED (NAME_ITEM));
803600 2 0 ELSE IF GEN_ITEM = 'R' THEN
      RETURN;
803620 2 0 PARENT_NUM = 0;
803630 2 0 VIRTUAL = 'O'B;
803640 2 0 CARTESIAN = 'O'B;
803650 2 0 DO WHILE (NAME_ITEM = '');
803660 2 1 VES_CART_KEY = GETS (NAME_ITEM, '');
803670 2 1 NM_ITEM = GETS (VES_CART_KEY, '');
803680 2 1 VIRTUAL = ((GETS (VES_CART_KEY, '') = 'V') | VIRTUAL);
803690 2 1 CARTESIAN = ((VES_CART_KEY = 'C') | CARTESIAN);
803700 2 1 DO I = PARENT_NUM + 1 TO 15
      WHILE (ENTITY (CUR_ES).ATTR (I).USES = '');
      IF NM_ITEM = ENTITY (CUR_ES).ATTR (I).USES
      & CARTESIAN = ENTITY (CUR_ES).ATTR (I).CART_KEY
      & PARENT_NUM = ENTITY (CUR_ES).ATTR (I).A_PARENT THEN DO;
803720 2 2 PARENT_NUM = I;
      I = 20;
      END;
803750 2 3 IF I = 16 THEN
803760 2 3 /*TOO MANY ATTRIBUTES FOR THIS ES*/;
803770 2 3 ELSE IF I < 16 THEN DO;
803780 2 2 ENTITY (CUR_ES).ATTR (I).USES = NM_ITEM;
803790 2 2 ENTITY (CUR_ES).ATTR (I).VES_KEY = VIRTUAL;
803810 2 1 ENTITY (CUR_ES).ATTR (I).CART_KEY = CARTESIAN;
803820 2 2 ENTITY (CUR_ES).ATTR (I).A_PARENT = PARENT_NUM;
803830 2 2 PARENT_NUM = I;
803840 2 2 END;
803850 2 2 END;
803860 2 2 ATTR_ITEM = CHAR (PARENT_NUM) || 'NTH';
803870 2 2 END;
803880 2 1 END;
803890 2 0 ATTR_ITEM = CHAR (PARENT_NUM) || 'NTH';
803900 2 0 END GENATTR;
PAR03480
PAR03490
PAR03500
PAR03510
PAR03520
PAR03530
PAR03540
PAR03550
PAR03560
PAR03570
PAR03580
PAR03590
PAR03600
PAR03610
PAR03620
PAR03630
PAR03640
PAR03650
PAR03660
PAR03670
PAR03680
PAR03690
PAR03700
PAR03710
PAR03720
PAR03730
PAR03740
PAR03750
PAR03760
PAR03770
PAR03780
PAR03790
PAR03800
PAR03810
PAR03820
PAR03830
PAR03840
PAR03850
PAR03860
PAR03870
PAR03880
PAR03890
PAR03900
PAR03910

```

PL/I OPTIMIZING COMPILER 1PARSE: PROC (MACH, TOKENS_PTR, XTREE, XCHANGE, ENTITY, DEBUG):

NUMBER LEV NT

803920 1 0 EXCH: PROC (TYP):

803940 2 0 DCL TYP CHAR (4):

803960 2 0 XCH_LOC = XCH_LOC + 1;

803970 2 0 XCHANGE (XCH_LOC) = GETS (STACK (1,TDS (1)), '.');

803980 2 0 STACK (1,TDS (1)) = CHAR (XCH_LOC) || TYP;

804000 2 0 END EXCH;

804020 1 0 ADDON: PROC:

804040 2 0 XCHANGE (XCH_LOC) = XCHANGE (XCH_LOC) || ' ' || STACK (1,TDS (1));

804060 2 0 END ADDON;

PARO3920
PARO3930
PARO3940
PARO3950
PARO3960
PARO3970
PARO3980
PARO3990
PARO4000
PARO4010

PARO4020
PARO4030
PARO4040
PARO4050
PARO4060
PARO4070

1PARSE: PROC (MACH.TOKENS_PTR,XTREE,XCHNGE,ENTITY,DEBUG);

PL/I OPTIMIZING COMPILER

NUMBER LEV NT

804080 1 0 END PARSE;

PARO4080

NUMBER LEV NT

```

10 0 SMPLFY: PROC (XTREE,ENTITY,XCHNGE,RETPTR):
    %INCLUDE XTREE;*****
100010 1 0 DCL 1 XTREE (1000),
    2 LABEL CHAR (30) VAR,
    2 CHILD FIXED,
    2 LINK FIXED;
    *****
300010 1 0 %INCLUDE XCHNGE;*****
    DCL XCHNGE (20) CHAR (80) VAR;
    *****
400010 1 0 %INCLUDE ENTITY;*****
    DCL 1 ENTITY (0:12),
    2 NAME CHAR (30) VAR,
    2 DEPTH FIXED,
    2 VES_FN CHAR (2) VAR,
    2 WHERE FIXED,
    2 N_PARENT (2) FIXED,
    2 VES_PAR PTR,
    2 ATTR (15),
    3 VES_KEY BIT (1),
    3 CART_KEY BIT (1),
    3 SING_OCC BIT (1),
    3 A_PARENT FIXED,
    3 USES CHAR (30) VAR,
    3 LIST PTR;
400150 1 0 DCL 1 N_MAP (2) BASED,
    2 NUM (15) FIXED;
    *****
600010 1 0 %INCLUDE ATTRIB;*****
    DCL 1 ATTRIB BASED,
    2 LEVEL FIXED,
    2 ITEM CHAR (50) VAR,
    2 NEXT PTR;
    *****
700020 1 0 %INCLUDE ARETE;*****
    DCL 1 RETE_ARG BASED (RETEP), /* WILL ALSO BE RETE_RTN */
    2 CTL_INFO,
    3 LEN_FIXED BIN (15),
    3 CBTP_FIXED BIN (15) INIT (21),
    3 PTR PTR,
    2 NUM_ATTR FIXED,
    2 NUM_COND FIXED, /* NUMBER OF CONDITIONS */
    2 ENT,
    3 NAME CHAR (30) VAR, /* ENTITY SET NAME */
    3 DEPTH FIXED, /* FILLED IN WHEN RETURNED */
    3 ATTR (NATTR REFER (RETE_ARG_NUM_ATTR)),
    4 SING_OCC BIT (1), /* IF SINGLE OCCUR THEN SET */
    4 A_PARENT FIXED, /* PARENT NUMBER */

```

PL/I OPTIMIZING COMPILER 1SMPLFY: PROC (XTREE,ENTITY,XCHNGE,RETPTR);

NUMBER LEV NT

```

4 USES CHAR(30) VAR, /* ATTRIBUTE NAME */
4 LIST PTR, /* POINT TO LIST OF OCC OF THIS ATTR IF ANY */
2 COND (NCOND REFER (RETE_ARG.NUM COND)),
3 ATTRREF FIXED, /* POINT TO ATTR IN ATTR ARRAY ABOVE */
3 NEG BIT(1), /* NEGATION OF RELATOR */
3 REL CHAR(1), /* '<', '>', '<=', '>=' */
2 CDATA (10) CHAR (50) VAR, /* UP TO 10 'MULTI' ITMES */
2 RTN CODE FIXED BIN,
2 NEXT_PTR PTR;
700240 1 0 DCL RETEP PTR;

700260 1 0 DCL 1 RETE_RTN1 BASED, /* USED WHEN RETURNED */
2 CTL,
3 LEN FIXED BIN(31),
3 CBTP FIXED BIN (31) INIT (46),
3 PTR PTR,
2 LEVEL FIXED BIN, /* OCCUR NUMBER */
2 ITEM CHAR (50),
2 NEXT_PTR PTR; /* POINT TO NEXT ON THE SAME ATTRIBUTE LIST */
*****
800090 1 0 DCL (RETPTR,TAIIR,P) PTR,
800110 1 0 DCL 1 ENTCOND (12,12),
2 ATTRREF FIXED,
2 NEG BIT (1),
2 REL CHAR (1) VAR,
2 CDATA (10) CHAR (30) VAR;

800170 1 0 ENTCOND (*,*) .ATTRREF = 0;
800180 1 0 ENTCOND (*,*) .NEG = 'O'B;
800190 1 0 ENTCOND (*,*) .REL = ' ';
800200 1 0 ENTCOND (*,*) .CDATA (*) = ' ';

```

PL/I OPTIMIZING COMPILER ISMPLFY: PROC (XTREE,ENTITY,XCHNGE,RETPTR):

NUMBER	LEV	NT		
800220	1	0	DO CUR_ES = 12 TO 1 BY -1;	SMP00220
800230	1	1	SELECT ENTITY (CUR_ES).VES_FN);	SMP00230
800240	1	2	WHEN ('S') DO;	SMP00240
800250	1	3	ALLOCATE N_MAP SET (ENTITY (CUR_ES).VES_PAR);	SMP00250
800260	1	3	CALL PRPGATE (ENTITY (CUR_ES).N_PARENT (1),	SMP00260
			'O'B,	SMP00270
			ENTITY (CUR_ES).VES_PAR -> N_MAP (1).NUM (*));	SMP00280
800290	1	3	END;	SMP00290
800300	1	2	WHEN ('MU','MI','SU','SI') DO;	SMP00300
800310	1	3	ALLOCATE N_MAP SET (ENTITY (CUR_ES).VES_PAR);	SMP00310
800320	1	3	CALL PRPGATE (ENTITY (CUR_ES).N_PARENT (1),	SMP00320
			'O'B,	SMP00330
			ENTITY (CUR_ES).VES_PAR -> N_MAP (1).NUM (*));	SMP00340
800350	1	3	CALL PRPGATE (ENTITY (CUR_ES).N_PARENT (2),	SMP00350
			'O'B,	SMP00360
			ENTITY (CUR_ES).VES_PAR -> N_MAP (2).NUM (*));	SMP00370
800380	1	3	END;	SMP00380
800390	1	2	WHEN ('CS') DO;	SMP00390
800400	1	3	ALLOCATE N_MAP SET (ENTITY (CUR_ES).VES_PAR);	SMP00400
800410	1	3	CALL PRPGATE (ENTITY (CUR_ES).N_PARENT (1),	SMP00410
			'I'B,	SMP00420
			ENTITY (CUR_ES).VES_PAR -> N_MAP (1).NUM (*));	SMP00430
800440	1	3	CALL PRPGATE (ENTITY (CUR_ES).N_PARENT (2),	SMP00440
			'O'B,	SMP00450
			ENTITY (CUR_ES).VES_PAR -> N_MAP (2).NUM (*));	SMP00460
800470	1	3	END;	SMP00470
800480	1	2	OTHERWISE;	SMP00480
800490	1	2	END;	SMP00490
800500	1	1	END;	SMP00500
				SMP00510

PL/I OPTIMIZING COMPILER 1SMPLFY: PROC (XTREE,ENTITY,XCHANGE,RETPTR);

NUMBER LEV NT

```

800520 1 0 RETPTR = NULL ();
800530 1 0 DO CUR_ES = 1 TO 12 WHILE (ENTITY (CUR_ES).VES_FN ^= '');
800540 1 1 NCOND = 0;
800550 1 1 IF ENTITY (CUR_ES).VES_FN = 'R' THEN DO;
800560 1 2 IF ENTITY (CUR_ES).WHERE > 0
      & SEARCH (ENTITY (CUR_ES).WHERE) <= -3 THEN
      ENTITY (CUR_ES).WHERE = 0;
      DO I = 1 TO 15 WHILE (ENTITY (CUR_ES).ATTR (I).USES ^= '');
      END;
      NATTR = I - 1;
      ALLOCATE RETE_ARG SET (P);
      P -> RETE_ARG.ENT_NAME = ENTITY (CUR_ES).NAME;
      P -> RETE_ARG.ENT_DEPTH = ENTITY (CUR_ES).DEPTH;
      DO I = 1 TO NATTR;
      P -> RETE_ARG.ENT.ATTR (I).SING_OCC =
        ENTITY (CUR_ES).ATTR (I).SING_OCC;
      P -> RETE_ARG.ENT.ATTR (I).A_PARENT =
        ENTITY (CUR_ES).ATTR (I).A_PARENT;
      P -> RETE_ARG.ENT.ATTR (I).USES =
        ENTITY (CUR_ES).ATTR (I).USES;
      END;
      DO I = 1 TO NCOND;
      P -> RETE_ARG.COND (I).ATTRREF = ENTCOND (CUR_ES,I).ATTRREF;
      P -> RETE_ARG.COND (I).NEG = ENTCOND (CUR_ES,I).NEG;
      P -> RETE_ARG.COND (I).REL = ENTCOND (CUR_ES,I).REL;
      P -> RETE_ARG.COND (I).CDATA (*) =
        ENTCOND (CUR_ES,I).CDATA (*);
      END;
      IF RETPTR = NULL () THEN
        RETPTR = P;
      ELSE DO;
        TAILR -> RETE_ARG.NEXT_PTR = P;
        TAILR -> RETE_ARG.CTL_INFO_PTR = P;
      END;
      TAILR = P;
      END;
      TAILR -> RETE_ARG.NEXT_PTR = NULL ();
      TAILR -> RETE_ARG.CTL_INFO_PTR = NULL ();
      END;

```

PL/I OPTIMIZING COMPILER 1SMPLFY: PROC (XTREE,ENTITY,XCHNGE,RETPTR);

NUMBER LEV NT

```

800920 1 0 PRPGATE: PROC (PAR_ENT,CART_TYP,MAP_NUM);
800940 2 0 DCL PAR_ENT FIXED;
800950 2 0 DCL CART_TYP BIT (1);
800960 2 0 DCL MAP_NUM (15) FIXED;
800970 2 0 DCL (I,J,PARENT_NUM) FIXED;
800980 2 0 DCL NAME_ITEM_CHAR (80) VAR,
      NM_ITEM_CHAR (30) VAR;

801010 2 0 MAP_NUM (*) = 0;
801020 2 0 DO I = 1 TO 15 WHILE (ENTITY (CUR_ES).ATTR (I).USES ^= '');
801030 2 1 IF ENTITY (CUR_ES).ATTR (I).CART_KEY = CART_TYP THEN DO;
801040 2 2 J = I;
801050 2 2 NAME_ITEM = ENTITY (CUR_ES).ATTR (J).USES;
801060 2 2 DO WHILE (ENTITY (CUR_ES).ATTR (J).A_PARENT ^= 0);
801070 2 3 J = ENTITY (CUR_ES).ATTR (J).A_PARENT;
801080 2 3 NAME_ITEM = ENTITY (CUR_ES).ATTR (J).USES || '';
      || NAME_ITEM;
801100 2 3 END;
801110 2 2 PARENT_NUM = 0;
801120 2 2 DO WHILE (NAME_ITEM ^= '');
801130 2 3 NM_ITEM = GETS (NAME_ITEM, '');
801140 2 3 DO J = PARENT_NUM + 1 TO 15
      WHILE (ENTITY (PAR_ENT).ATTR (J).USES ^= '');
801160 2 4 IF NM_ITEM = ENTITY (PAR_ENT).ATTR (J).USES
      & ENTITY (PAR_ENT).ATTR (J).CART_KEY = 'O'B
      & PARENT_NUM = ENTITY (PAR_ENT).ATTR (J).A_PARENT THEN
      DO;
        PARENT_NUM = J;
        J = 20;
      END;
      END;
      IF J = 16 THEN
        /*100 MAY ATTRIBUTES FOR THIS ES*/;
      ELSE IF J < 16 THEN DO;
        ENTITY (PAR_ENT).ATTR (J).USES = NM_ITEM;
        ENTITY (PAR_ENT).ATTR (J).VES_KEY = 'O'B;
        ENTITY (PAR_ENT).ATTR (J).CART_KEY = 'O'B;
        ENTITY (PAR_ENT).ATTR (J).A_PARENT = PARENT_NUM;
        PARENT_NUM = J;
      END;
      END;
      MAP_NUM (I) = PARENT_NUM;
      END;
      END;
801360 2 1 END;
801380 2 0 END PRPGATE;

```

SMP00920
 SMP00930
 SMP00940
 SMP00950
 SMP00960
 SMP00970
 SMP00980
 SMP00990
 SMP01000
 SMP01010
 SMP01020
 SMP01030
 SMP01040
 SMP01050
 SMP01060
 SMP01070
 SMP01080
 SMP01090
 SMP01100
 SMP01110
 SMP01120
 SMP01130
 SMP01140
 SMP01150
 SMP01160
 SMP01170
 SMP01180
 SMP01190
 SMP01200
 SMP01210
 SMP01220
 SMP01230
 SMP01240
 SMP01250
 SMP01260
 SMP01270
 SMP01280
 SMP01290
 SMP01300
 SMP01310
 SMP01320
 SMP01330
 SMP01340
 SMP01350
 SMP01360
 SMP01370
 SMP01380
 SMP01390

PL/I OPTIMIZING COMPILER 1SMPLFY: PROC (XTREE,ENTITY,XCHNGE,RETPTR);

NUMBER LEV NT

```

801400 1 0 SEARCH: PROC (LINK_PT) RETURNS (FIXED) RECURSIVE;
801420 2 0 DCL (LINK_PT, TAG1, TAG2) FIXED,
      (TERM_TYP, TERM_NM) CHAR (30) VAR;
801450 2 0 IF XTREE (LINK_PT).CHILD = 0 THEN DO;
801460 2 1 TERM_TYP = XTREE (LINK_PT).LABEL;
801470 2 1 TERM_NM = GETS (TERM_TYP, '');
801480 2 1 IF TERM_TYP = 'NTH' THEN
      RETURN (-2);
801500 2 1 ELSE IF INDEX ('A.S.MUL', TERM_TYP) ^= 0 THEN
      RETURN (-1);
801520 2 1 END;
801530 2 0 ELSE IF XTREE (LINK_PT).CHILD = 1
      & XTREE (LINK_PT).LABEL = ''
      & SEARCH (XTREE (LINK_PT).LINK) = -3 THEN DO;
801560 2 1 ENTCOND (CUR_ES, NCND).NEG =
      ^ENTCOND (CUR_ES, NCND).NEG;
      RETURN (-3);
801580 2 1 END;
801590 2 1 END;
801600 2 0 ELSE IF XTREE (LINK_PT).CHILD = 2
      & INDEX ('>.<.', XTREE (LINK_PT).LABEL) ^= 0 THEN DO;
801620 2 1 TAG1 = SEARCH (XTREE (LINK_PT).LINK);
801630 2 1 TAG2 = SEARCH (XTREE (LINK_PT + 1).LINK);
801640 2 1 IF TAG1 = -2 & TAG2 = -1 THEN DO;
801650 2 2 CALL SETREL (LINK_PT,
      XTREE (LINK_PT).LINK,
      XTREE (LINK_PT + 1).LINK);
      RETURN (-3);
801680 2 2 END;
801690 2 2 END;
801700 2 1 ELSE IF TAG1 = -1 & TAG2 = -2 THEN DO;
801710 2 2 CALL SETREL (LINK_PT,
      XTREE (LINK_PT + 1).LINK,
      XTREE (LINK_PT).LINK);
      RETURN (-3);
801740 2 2 END;
801750 2 2 END;
801760 2 1 ELSE IF TAG1 <= -3 THEN
      IF TAG2 <= -3 THEN
      RETURN (-4);
      ELSE IF TAG2 = 0 THEN
      RETURN (XTREE (LINK_PT + 1).LINK);
      ELSE
      RETURN (TAG2);
801790 2 1 ELSE IF TAG2 <= -3 THEN
      IF TAG1 = 0 THEN
      RETURN (XTREE (LINK_PT).LINK);
      ELSE
      RETURN (TAG1);
801810 2 1 IF TAG1 > 0 THEN

```

PL/I OPTIMIZING COMPILER ISMPFY: PROC (XTREE,ENTITY,XCHNGE,RETPTR);

NUMBER LEV NT

```

      XTREE (LINK_PT).LINK = TAG1;
      IF TAG2 > 0 THEN
        XTREE (LINK_PT + 1).LINK = TAG2;
      END;
      RETURN (0);
      801900 2 1
      801920 2 1
      801930 2 0
      801950 2 0 END SEARCH;

```

SMP01890
 SMP01900
 SMP01910
 SMP01920
 SMP01930
 SMP01940
 SMP01950
 SMP01960
 SMP01970

PL/I OPTIMIZING COMPILER 1SMPLFY: PROC (XTREE,ENTITY,XCHANGE,REIPTR);

NUMBER LEV NT

```

801980 1 0 GETS: PROC (LIST,TERM_ITEM) RETURNS (CHAR (30) VAR);
802000 2 0 DCL LIST CHAR (*) VAR,
      TERM_ITEM CHAR (1),
      RTN_LIST CHAR (30) VAR,
      I FIXED;
802050 2 0 I = INDEX (LIST,TERM_ITEM);
802060 2 0 IF I = 0 THEN DO;
802070 2 1 RTN_LIST = LIST;
802080 2 1 LIST = ',';
802090 2 1 END;
802100 2 0 ELSE DO;
802110 2 1 RTN_LIST = SUBSTR (LIST,I - 1);
802120 2 1 LIST = SUBSTR (LIST,I + 1);
802130 2 1 END;
802140 2 0 RETURN (RTN_LIST);
802160 2 0 END GETS;
SMPO1980
SMPO1990
SMPO2000
SMPO2010
SMPO2020
SMPO2030
SMPO2040
SMPO2050
SMPO2060
SMPO2070
SMPO2080
SMPO2090
SMPO2100
SMPO2110
SMPO2120
SMPO2130
SMPO2140
SMPO2150
SMPO2160
SMPO2170

```

PL/I OPTIMIZING COMPILER 1SMPFY: PROC (XTREE,ENTITY,XCHNGE,REIPTR);

NUMBER LEV NT

```

802180 1 0 SETREL: PROC (LINK_PT,ATTR_LINK,CONST_LINK);
802200 2 0 DCL (LINK_PT,ATTR_LINK,CONST_LINK,I) FIXED,
      CONSTTYP CHAR (30) VAR,
      CONSTANT CHAR (80) VAR;

802240 2 0 NCOND = NCOND + 1;
802250 2 0 ENTCOND (CUR_ES,NCOND).REL = XTREE (LINK_PT).LABEL;
802260 2 0 ENTCOND (CUR_ES,NCOND).ATTREF
      = FIXED (GETS (XTREE (ATTR_LINK).LABEL,','));
802280 2 0 CONSTTYP = XTREE (CONST_LINK).LABEL;
802290 2 0 CONSTANT = GETS (CONSTTYP,','));
802300 2 0 IF CONSTTYP = 'MUL' THEN
      CONSTANT = XCHNGE (FIXED (CONSTANT));
802320 2 0 DO I = 1 TO 10 WHILE (CONSTANT ^= '');
802330 2 1 ENTCOND (CUR_ES,NCOND).CDATA (I) = GETS (CONSTANT,','));
802340 2 1 END;

802360 2 0 END SETREL;

```

SMP02180
 SMP02190
 SMP02200
 SMP02210
 SMP02220
 SMP02230
 SMP02240
 SMP02250
 SMP02260
 SMP02270
 SMP02280
 SMP02290
 SMP02300
 SMP02310
 SMP02320
 SMP02330
 SMP02340
 SMP02350
 SMP02360
 SMP02370

1SMPLFY: PROC (XTREE,ENTITY,XCHNGE,RETPTR);

PL/I OPTIMIZING COMPILER

NUMBER LEV NT

802380 1 0 END SMPLFY;

SMPO2380

NUMBER LEV NT

```

10      0  CONVERT: PROC (ENTITY, RETPTR);
          %INCLUDE ENTITY;*****
1000010  1  0  DCL 1 ENTITY (0:12),
          2 NAME CHAR (30) VAR,
          2 DEPTH FIXED,
          2 VES_FN CHAR (2) VAR,
          2 WHERE FIXED,
          2 N_PARENT (2) FIXED,
          2 VES_PAR PTR,
          2 ATTR (15),
          3 VES_KEY BIT (1),
          3 CART_KEY BIT (1),
          3 SING_OCC BIT (1),
          3 A_PARENT FIXED,
          3 USES CHAR (30) VAR,
          3 LIST PTR;
100150  1  0  DCL 1 N_MAP (2) BASED,
          2 NUM (15) FIXED;
          *****
          %INCLUDE ATTRIB;*****
100300010  1  0  DCL 1 ATTRIB BASED,
          2 LEVEL FIXED,
          2 ITEM CHAR (50) VAR,
          2 NEXT PTR;
          *****
100500020  1  0  DCL 1 RETE_ARG BASED (RETEP), /* WILL ALSO BE RETE_RTN */
          2 CTL_INFO,
          3 LEN_FIXED BIN (15),
          3 CBTP_FIXED BIN (15) INIT (21),
          3 PTR_PTR,
          2 NUM_ATTR FIXED,
          2 NUM_COND FIXED, /* NUMBER OF CONDITIONS */
          2 ENT,
          3 NAME_CHAR (30) VAR, /* ENTITY SET NAME */
          3 DEPTH_FIXED, /* FILLED IN WHEN RETURNED */
          3 ATTR (NATTR REFER (RETE_ARG.NUM_ATTR)),
          4 SING_OCC BIT (1), /* IF SINGLE OCCUR THEN SET */
          4 A_PARENT_FIXED, /* PARENT NUMBER */
          4 USES_CHAR (30) VAR, /* ATTRIBUTE NAME */
          4 LIST_PTR, /* POINT TO LIST OF OCC OF THIS ATTR IF ANY */
          2 COND (NCOND REFER (RETE_ARG.NUM_COND)),
          3 ATTRREF_FIXED, /* POINT TO ATTR IN ATTR ARRAY ABOVE */
          3 NEG_BIT (1), /* NEGATION OF RELATOR */
          3 REL_CHAR (1), /* '<', '>', '<=', '>=' */
          3 CDATA (10) CHAR (50) VAR, /* UP TO 10 'MULTI' ITMES */
          2 RTN_CODE_FIXED BIN,
          2 NEXT_PTR_PTR;
          *****
          CNV000010
          CNV000020
          CNV000030
          ENT000010
          ENT000020
          ENT000030
          ENT000040
          ENT000050
          ENT000060
          ENT000070
          ENT000080
          ENT000090
          ENT000100
          ENT000110
          ENT000120
          ENT000130
          ENT000140
          ENT000150
          ENT000160
          CNV000030
          CNV000040
          AT1000010
          AT1000020
          AT1000030
          AT1000040
          CNV000040
          CNV000050
          F2A000020
          F2A000030
          F2A00040
          F2A00050
          F2A00060
          F2A00070
          F2A00080
          F2A00090
          F2A00100
          F2A00110
          F2A00120
          F2A00130
          F2A00140
          F2A00150
          F2A00160
          F2A00170
          F2A00180
          F2A00190
          F2A00200
          F2A00210
          F2A00220
          F2A00230

```

PL/I OPTIMIZING COMPILER 1CNVERT: PROC (ENTITY,RETPTR);

NUMBER LEV NT

```

500240 1 0 DCL RETEP PTR;
500260 1 0 DCL 1 RETE RTN1 BASED, /* USED WHEN RETURNED */
      2 CTL,
      3 LEN FIXED BIN(31),
      3 CBTP FIXED BIN (31) INIT (46),
      3 PTR PTR,
      2 LEVEL FIXED BIN, /* OCCUR NUMBER */
      2 ITEM CHAR (50),
      2 NEXT_PTR PTR; /* POINT TO NEXT ON THE SAME ATTRIBUTE LIST */
*****
600070 1 0 DCL (RETPTR, TAILR, P, TAILP) PTR,
      (CUR_ES, I) FIXED;

600100 1 0 TAILR = RETPTR;
600110 1 0 DO CUR_ES = 1 TO 12 WHILE (ENTITY (CUR_ES).VES_FN ^= '');
600120 1 1 IF ENTITY (CUR_ES).VES_FN = 'R' THEN DO;
600130 1 2 ENTITY (CUR_ES).DEPTH = TAILR -> RETE_ARG.ENT.DEPTH;
600140 1 2 DO I = 1 TO 15 WHILE (ENTITY (CUR_ES).ATTR (I).USES ^= '');
600150 1 3 ENTITY (CUR_ES).ATTR (I).SING_OCC =
      TAILR -> RETE_ARG.ENT.ATTR (I).SING_OCC;
600170 1 3 Q = TAILR -> RETE_ARG.ENT.ATTR (I).LIST;
600180 1 3 DO WHILE (Q ^= NULL ());
600190 1 4 ALLOCATE ATTRIB SET (P);
600200 1 4 P -> ATTRIB.ITEM = Q -> RETE_RTNI.ITEM;
600210 1 4 P -> ATTRIB.LEVEL = Q -> RETE_RTNI.LEVEL;
600220 1 4 IF ENTITY (CUR_ES).ATTR (I).LIST = NULL ( ) THEN
      ENTITY (CUR_ES).ATTR (I).LIST = P;
      ELSE
        TAILP -> ATTRIB.NEXT = P;
600240 1 4 TAILP = P;
600260 1 4 Q = Q -> RETE_RTNI.NEXT_PTR;
600270 1 4 END;
600280 1 4 TAILP -> ATTRIB.NEXT = NULL ( );
600290 1 3 END;
600300 1 3 TAILR = TAILR -> RETE_ARG.NEXT_PTR;
600310 1 2 END;
600320 1 2 END;
600330 1 1 END;

600350 1 0 END CNVERT;

```

F2A00240
 F2A00250
 F2A00260
 F2A00270
 F2A00280
 F2A00290
 F2A00300
 F2A00310
 F2A00320
 F2A00330
 CNV00050
 CNV00060
 CNV00070
 CNV00080
 CNV00090
 CNV00100
 CNV00110
 CNV00120
 CNV00130
 CNV00140
 CNV00150
 CNV00160
 CNV00170
 CNV00180
 CNV00190
 CNV00200
 CNV00210
 CNV00220
 CNV00230
 CNV00240
 CNV00250
 CNV00260
 CNV00270
 CNV00280
 CNV00290
 CNV00300
 CNV00310
 CNV00320
 CNV00330
 CNV00340
 CNV00350

0 XECUTE: PROC (XTREE,ENTITY,XCHNGE);

```

XEC00010
XEC00020
XEC00030
XEC00040
XEC00050
XEC00060
XEC00070
XEC00080
XEC00090
XEC00100
XEC00110
XEC00120
XEC00130
XEC00140
XEC00150
XEC00160
XEC00170
XEC00180
XTR00010
XTR00020
XTR00030
XTR00040
XEC00180
XEC00190
ENT00010
ENT00020
ENT00030
ENT00040
ENT00050
ENT00060
ENT00070
ENT00080
ENT00090
ENT00100
ENT00110
ENT00120
ENT00130
ENT00140
ENT00150
ENT00160
XEC00190
XEC00200
IND00010
XEC00200
XEC00200
XEC00210
ATTC0010
ATTC0020
ATTC0030
ATTC0040
XECUTE: PROC (XTREE,ENTITY,XCHNGE);

0

10

/******
/*
/*EACH NODE IN THIS TREE TRAVERSAL PROGRAM HAS A UNIT_VERT TAG
/*AND THESE TAGS ARE PROPAGATED UP THE TREE FROM CHILDREN TO
/*PARENTS. PROPAGATION OF THE TAGS FOLLOWS THE RULE THAT UNITARY
/*ITEMS CAN ONLY RESULT FROM THE GENERIC OPERATION BETWEEN TWO
/*UNITARY ITEMS OR FROM A BUILTIN FUNCTION WHICH RETURNS A UNITARY
/*ITEM; IN ALL OTHER CASES THE RETURNED ITEMS ARE OF TYPE VERTICAL.
/*THIS SCHEME ALLOWS GENERIC OPERATORS TO DESIGNATE THE EXPECTED
/*TYPING OF THEIR RETURN ITEMS.
/*
/******
/******
%INCLUDE XTREE;*****
DCL 1 XTREE (1000).
    2 LABEL CHAR (30) VAR.
    2 CHILD FIXED.
    2 LINK FIXED;
*****
%INCLUDE ENTITY;*****
DCL 1 ENTITY (0:12),
    2 NAME CHAR (30) VAR.
    2 DEPTH FIXED,
    2 VES_FN CHAR (2) VAR,
    2 WHERE FIXED,
    2 N_PARENT (2) FIXED,
    2 VES_PAR PTR,
    2 ATTR (15),
    3 VES_KEY BIT (1),
    3 CART_KEY BIT (1),
    3 SING_OCC BIT (1),
    3 A_PARENT FIXED,
    3 USES_CHAR (30) VAR,
    3 LIST_PTR;
DCL 1 N_MAP (2) BASED,
    2 NUM (15) FIXED;
*****
%INCLUDE XCHNGE;*****
DCL XCHNGE (20) CHAR (80) VAR;
*****
%INCLUDE ATTRIB;*****
DCL 1 ATTRIB BASED,
    2 LEVEL FIXED,
    2 ITEM CHAR (50) VAR,
    2 NEXT_PTR;

```

PL/I OPTIMIZING COMPILER EXECUTE: PROC (XTREE,ENTITY,XCHNGE);

NUMBER LEV NT

```

*****
400230 1 0 DCL UNIT_VERT BIT (1),
      (CUR_ES,M,COPY_COUNT,KEYSET (15),NEWVL (0:2,15)) FIXED,
      (SLCTLIST,ATAIL (0:2,15)) PTR;

400270 1 0 DO CUR_ES = 1 TO 12 WHILE (ENTITY (CUR_ES).VES_FN ^= '');
400280 1 1 SELECT (ENTITY (CUR_ES).VES_FN);
400290 1 2 WHEN ('S')
      CALL COPYALL (ENTITY (CUR_ES).N_PARENT (1));
400310 1 2 WHEN ('MU')
      CALL MUNION;
400330 1 2 WHEN ('MI')
      CALL MINTER;
400350 1 2 WHEN ('SU') DO;
400360 1 3 CALL MUNION;
400370 1 3 CALL SINGLE;
400380 1 3 END;
400390 1 2 WHEN ('SI') DO;
400400 1 3 CALL MINTER;
400410 1 3 CALL SINGLE;
400420 1 3 END;
400430 1 2 WHEN ('CS')
      CALL CRTESN;
400450 1 2 OTHERWISE;
400460 1 2 END;
400470 1 1 IF ENTITY (CUR_ES).WHERE > 0 THEN
      IF OPERATE (SLCTLIST,ENTITY (CUR_ES).WHERE) THEN
          CALL DECRSET;
      ELSE IF SLCTLIST -> ATTRIB.ITEM = ':FALSE' THEN DO;
          DO M = 1 TO 15 WHILE (ENTITY (CUR_ES).ATTR (M).USES ^= '');
              CALL DISPOSE (ENTITY (CUR_ES).ATTR (M).LIST);
          END;
          ENTITY (CUR_ES).DEPTH = 0;
      END;
      END;
400500 1 1
400510 1 2
400520 1 3
400530 1 3
400540 1 2
400550 1 2
400560 1 1
      END;

```

```

XEC00210
XEC00220
XEC00230
XEC00240
XEC00250
XEC00260
XEC00270
XEC00280
XEC00290
XEC00300
XEC00310
XEC00320
XEC00330
XEC00340
XEC00350
XEC00360
XEC00370
XEC00380
XEC00390
XEC00400
XEC00410
XEC00420
XEC00430
XEC00440
XEC00450
XEC00460
XEC00470
XEC00480
XEC00490
XEC00500
XEC00510
XEC00520
XEC00530
XEC00540
XEC00550
XEC00560
XEC00570

```

PL/1 OPTIMIZING COMPILER 1XECUTE: PROC (XTREE, ENTITY, XCHANGE);

NUMBER LEV NT

```

400580    1    0    OPERATE: PROC (LIST1, LINK_PT) RETURNS (BIT (1)) RECURSIVE;
400610    2    0    DCL (LIST1, LIST2) PTR,
/*MODIFIES LIST1*/
         LINK_PT FIXED,
         (TERM_TYP, TERM_NM) CHAR (30) VAR,
         (UNIT_VERT1, UNIT_VERT2) BIT (1);
400660    2    0    IF XTREE (LINK_PT).CHILD = 0 THEN DO;
400670    2    1    TERM_TYP = XTREE (LINK_PT).LABEL;
400680    2    1    TERM_NM = GETS (TERM_TYP, ':');
400690    2    1    SELECT (TERM_TYP);
400700    2    2    WHEN ('NTH')
              UNIT_VERT1 = CREATE_LIST (LIST1, FIXED (TERM_NM));
400720    2    2    WHEN ('A', 'S')
              UNIT_VERT1 = CREATE_CONST (LIST1, TERM_NM);
400740    2    2    WHEN ('MUL')
              CALL CREATE_CONST (LIST1, ':', '|| TERM_NM);
400760    2    2    OTHERWISE
              UNIT_VERT1 = GO_DOWNO (LIST1, TERM_NM);
              END;
400780    2    2    END;
400790    2    1    END;
400800    2    0    ELSE IF XTREE (LINK_PT).CHILD = 1 THEN DO;
400810    2    1    UNIT_VERT1 = OPERATE (LIST1, XTREE (LINK_PT).LINK);
400820    2    1    UNIT_VERT1 = GO_DOWN1 (LIST1, UNIT_VERT1, XTREE (LINK_PT).LABEL);
400830    2    1    END;
400840    2    0    ELSE IF XTREE (LINK_PT).CHILD = 2 THEN DO;
400850    2    1    UNIT_VERT1 = OPERATE (LIST1, XTREE (LINK_PT).LINK);
400860    2    1    UNIT_VERT2 = OPERATE (LIST2, XTREE (LINK_PT + 1).LINK);
400870    2    1    IF SUBSTR (LIST2 -> ATTRIB.ITEM, 1, 1) = ':' THEN
              UNIT_VERT1 =
              MULTEQ2 (LIST1,
              XCHANGE (FIXED (SUBSTR (LIST2 -> ATTRIB.ITEM, 2))));
400910    2    1    ELSE
              UNIT_VERT1 = GO_DOWN2 (LIST1, LIST2, UNIT_VERT1, UNIT_VERT2,
              XTREE (LINK_PT).LABEL);
400940    2    1    END;
400950    2    0    ELSE IF XTREE (LINK_PT).LABEL = 'STR' THEN
              UNIT_VERT1 = STR (LIST1, LINK_PT);
400970    2    0    ELSE
              /*BAD CHILDREN*/;
400990    2    0    RETURN (UNIT_VERT1);
401010    2    0    END OPERATE;
XEC00580
XEC00590
XEC00600
XEC00610
XEC00620
XEC00630
XEC00640
XEC00650
XEC00660
XEC00670
XEC00680
XEC00690
XEC00700
XEC00710
XEC00720
XEC00730
XEC00740
XEC00750
XEC00760
XEC00770
XEC00780
XEC00790
XEC00800
XEC00810
XEC00820
XEC00830
XEC00840
XEC00850
XEC00860
XEC00870
XEC00880
XEC00890
XEC00900
XEC00910
XEC00920
XEC00930
XEC00940
XEC00950
XEC00960
XEC00970
XEC00980
XEC00990
XEC01000
XEC01010
XEC01020

```

PL/I OPTIMIZING COMPILER 1XECUTE: PROC (XTREE.ENTITY,XCHNGE);

```

NUMBER LEV NT
401030 1 0 STR: PROC (LIST, LINK_PT) RETURNS (BIT (1)) RECURSIVE;
401050 2 0 DCL (LIST, LISTP, P (4)) PTR,
      (LINK_PT, START_PT, END_PT) FIXED,
      (UNIT_VERT, ENDING_RELATIVE, FIX_START, FIX_END) BIT (1) INIT ('O'B'),
      UV (4) BIT (1) INIT ((4)'O'B'),
      STRING CHAR (30) VAR;
401110 2 0 UNIT_VERT = OPERATE (LIST, XTREE (LINK_PT).LINK);
401120 2 0 UV (1) = OPERATE (P (1), XTREE (LINK_PT + 1).LINK);
401130 2 0 IF XTREE (LINK_PT + 2).LINK = -1 THEN
      FIX_START = '1'B;
401150 2 0 ELSE
      UV (2) = OPERATE (P (2), XTREE (LINK_PT + 2).LINK);
401170 2 0 ENDING = (XTREE (LINK_PT + 3).LINK = -1);
401180 2 0 IF ^ENDING THEN DO;
401190 2 1 RELATIVE = (XTREE (LINK_PT + 3).LINK = -3);
401200 2 1 UV (3) = OPERATE (P (3), XTREE (LINK_PT + 4).LINK);
401210 2 1 IF XTREE (LINK_PT + 5).LINK = -1 THEN
      FIX_END = '1'B;
401230 2 1 ELSE
      UV (4) = OPERATE (P (4), XTREE (LINK_PT + 5).LINK);
401250 2 1 END;
401260 2 0 LISTP = LIST;
401270 2 0 DO WHILE (LISTP ^= NULL ());
401280 2 1 STRING = LISTP -> ATTRIB.ITEM;
401290 2 1 IF FIX_START THEN
      START_PT = FIND_PT (1, 0, 'O'B');
401310 2 1 ELSE
      START_PT = FIND_PT (1, 1, '1'B);
401330 2 1 START_PT = MIN (MAX (START_PT, 1), LENGTH (STRING) + 1);
401340 2 1 IF ENDING THEN
      LISTP -> ATTRIB.ITEM = SUBSTR (STRING, START_PT);
401360 2 1 ELSE DO;
401370 2 2 IF FIX_END THEN
      END_PT = FIND_PT (3, 0, 'O'B');
401390 2 2 ELSE IF RELATIVE THEN
      END_PT = FIND_PT (3, START_PT, 'O'B');
401410 2 2 ELSE
      END_PT = FIND_PT (3, 1, 'O'B');
401430 2 2 END_PT = MIN (END_PT, LENGTH (STRING));
401440 2 2 LISTP -> ATTRIB.ITEM =
      SUBSTR (STRING,
      START_PT,
      MAX (END_PT - START_PT + 1, 0));
401480 2 2 END;
401490 2 1 DO I = 1 TO 4;
401500 2 2 IF UV (I) THEN
      P (I) = P (I) -> ATTRIB.NEXT;
XECO1030
XECO1040
XECO1050
XECO1060
XECO1070
XECO1080
XECO1090
XECO1100
XECO1110
XECO1120
XECO1130
XECO1140
XECO1150
XECO1160
XECO1170
XECO1180
XECO1190
XECO1200
XECO1210
XECO1220
XECO1230
XECO1240
XECO1250
XECO1260
XECO1270
XECO1280
XECO1290
XECO1300
XECO1310
XECO1320
XECO1330
XECO1340
XECO1350
XECO1360
XECO1370
XECO1380
XECO1390
XECO1400
XECO1410
XECO1420
XECO1430
XECO1440
XECO1450
XECO1460
XECO1470
XECO1480
XECO1490
XECO1500
XECO1510

```

PL/I OPTIMIZING COMPILER 1XECUTE: PROC (XTREE,ENTITY,XCHANGE);

NUMBER LEV NT

```

401520 2 2      END;
401530 2 1      LISTP = LISTP -> ATTRIB.NEXT;
401540 2 1      END;
401550 2 0      DO I = 1 TO 4;
401560 2 1      CALL DISPOSE (P (I));
401570 2 1      END;
401580 2 0      RETURN (UNIT_VERT);

```

XECO1520
XECO1530
XECO1540
XECO1550
XECO1560
XECO1570
XECO1580
XECO1590

PL/I OPTIMIZING COMPILER 1XECUTE: PROC (XTREE.ENTITY,XCHNGE);

NUMBER LEV NT

```

401600 2 0 FIND_PT: PROC (PNUM,FND_LOC,OFFSET) RETURNS (FIXED);
401620 3 0 DCL (PNUM,FND_LOC,NUM_OCC,I) FIXED,
      OFFSET BIT (1),
      TEMPSTR CHAR (30) VAR;
401660 3 0 ON CONVERSION GOTO MESS;
401670 3 0 NUM_OCC = FIXED (P (PNUM) -> ATTRIB.ITEM);
401680 3 0 IF FND_LOC < 1 THEN
      RETURN (NUM_OCC);
401700 3 0 TEMPSTR = P (PNUM + 1) -> ATTRIB.ITEM || ',';
401710 3 0 DO I = FND_LOC TO LENGTH (STRING) WHILE (NUM_OCC > 0);
401720 3 1 IF IMPLTE (SUBSTR (STRING,I),TEMPSTR) THEN
      NUM_OCC = NUM_OCC - 1;
401740 3 1 END;
401750 3 0 IF OFFSET THEN
      I = I + LENGTH (TEMPSTR);
401770 3 0 RETURN (I - 2);
401790 3 0 MESS:
      IF OFFSET THEN
        NUM_OCC = LENGTH (STRING) + 1;
      ELSE
        NUM_OCC = 0;
401820 3 0 PUT FILE (ERROR) SKIP EDIT
      ('CONV ERROR FOR STR_OP',
      'STRING ARGUMENT =$',STRING,'$$',
      'NUM_OCC ARGUMENT =$',P (PNUM) -> ATTRIB.ITEM,'$$',
      'FORCED NUM_OCC =$',CHAR (NUM_OCC),'$$')
      (A,3 (SKIP,A,A));
401900 3 0 RETURN (NUM_OCC);
401920 3 0 END FIND_PT;
401940 2 0 END STR;
XECO1600
XECO1610
XECO1620
XECO1630
XECO1640
XECO1650
XECO1660
XECO1670
XECO1680
XECO1690
XECO1700
XECO1710
XECO1720
XECO1730
XECO1740
XECO1750
XECO1760
XECO1770
XECO1780
XECO1790
XECO1800
XECO1810
XECO1820
XECO1830
XECO1840
XECO1850
XECO1860
XECO1870
XECO1880
XECO1890
XECO1900
XECO1910
XECO1920
XECO1930
XECO1940
XECO1950

```

PL/I OPTIMIZING COMPILER EXECUTE: PROC (XTREE.ENTITY.XCHANGE);

NUMBER LEV NT

```

401960 1 0 GO_DOWNNO: PROC (LIST.OPNM) RETURNS (BIT (1));
      /*MODIFIES LIST*/
401990 2 0 DCL LIST PTR.
      OPNM CHAR (30) VAR;
402010 2 0 DCL DATE BUILTIN;
402030 2 0 ALLOCATE ATTRIB SET (LIST);
402040 2 0 LIST -> ATTRIB.NEXT = NULL ();
402050 2 0 SELECT (OPNM);
402060 2 1 WHEN ('DATE')
      LIST -> ATTRIB.ITEM = DATE;
402080 2 1 OTHERWISE;
402090 2 1 END;
402100 2 0 RETURN ('O'B);
402120 2 0 END GO_DOWNNO;
XECO1960
XECO1970
XECO1980
XECO1990
XECO2000
XECO2010
XECO2020
XECO2030
XECO2040
XECO2050
XECO2060
XECO2070
XECO2080
XECO2090
XECO2100
XECO2110
XECO2120
XECO2130

```

PL/I OPTIMIZING COMPILER 1XECUTE PROC (XTREE,ENTITY,XCHANGE);

NUMBER LEV NT

```

402140 1 0 GO_DOWN1: PROC (LIST,UNIT_VERT,REL) RETURNS (BIT (1));
/*MODIFIES LIST,UNIT_VERT*/
402170 2 0 DCL (LIST,P) PTR,
UNIT_VERT BIT (1),
REL CHAR (30) VAR;
402210 2 0 IF ^RET UNITARY (LIST,REL,UNIT_VERT) THEN
IF UNIT_VERT THEN
LIST -> ATTRIB. ITEM = UNARY (LIST -> ATTRIB. ITEM,REL);
ELSE DO;
P = LIST;
DO WHILE (P ^= NULL ());
P -> ATTRIB. ITEM = UNARY (P -> ATTRIB. ITEM,REL);
P = P -> ATTRIB. NEXT;
END;
END;
402300 2 1 RETURN (UNIT_VERT);
402310 2 0
402330 2 0 END GO_DOWN1;
XEC02140
XEC02150
XEC02160
XEC02170
XEC02180
XEC02190
XEC02200
XEC02210
XEC02220
XEC02230
XEC02240
XEC02250
XEC02260
XEC02270
XEC02280
XEC02290
XEC02300
XEC02310
XEC02320
XEC02330
XEC02340

```

PL/I OPTIMIZING COMPILER 1XECUTE PROC (XTREE,ENTITY,XCHANGE);

NUMBER LEV NT

```

402350 1 0 RET_UNITARY: PROC (LIST,REL,UNIT_VERT) RETURNS (BIT (1));
402360
402370 /*MODIFIES? LIST,UNIT_VERT*/
402380 2 0 DCL (LIST,P) PTR,
402390 REL CHAR (30) VAR,
402400 UNIT_VERT BIT (1),
402410 RESULT FIXED;
402420
402430 2 0 ON CONVERSION GOTO MESS;
402440 2 0 ON OVERFLOW GOTO MESS;
402450 2 0 IF INDEX ('DEPTH,MAX,MIN,SUM',REL) ^= 0 THEN DO;
402460 2 1 IF REL = 'DEPTH' THEN DO;
402470 2 2 LIST -> ATTRIB.ITEM = ENTITY (CUR_ES).DEPTH;
402480 2 2 END;
402490 2 1 ELSE DO;
402500 2 2 RESULT = FIXED (LIST -> ATTRIB.ITEM);
402510 2 2 P = LIST -> ATTRIB.NEXT;
402520 2 2 DO WHILE (P ^= NULL ());
402530 2 3 SELECT (REL);
402540 2 4 WHEN ('MAX')
402550 IF RESULT < FIXED (P -> ATTRIB.ITEM) THEN
402560 RESULT = FIXED (P -> ATTRIB.ITEM);
402570 2 4 WHEN ('MIN')
402580 IF RESULT > FIXED (P -> ATTRIB.ITEM) THEN
402590 RESULT = FIXED (P -> ATTRIB.ITEM);
402600 2 4 WHEN ('SUM')
402610 RESULT = RESULT + FIXED (P -> ATTRIB.ITEM);
402620 OTHERWISE;
402630 2 4 END;
402640 2 3 P = P -> ATTRIB.NEXT;
402650 2 3 END;
402660 2 2 LIST -> ATTRIB.ITEM = FIXED (RESULT);
402670 2 2 END;
402680 2 1 CALL DISPOSE (LIST -> ATTRIB.NEXT);
402690 2 1 UNIT_VERT = 'O'B;
402700 2 1 RETURN ('1'B);
402710 2 1 END;
402720 2 0 ELSE
402730 RETURN ('O'B);
402740
402750 2 0 MESS:
402760 PUT FILE (ERROR) SKIP EDIT
402770 ('CONV/OVFLOW/BAD_ARG ERROR FOR RET_UNITARY_OP',
402780 'ITEM ARGUMENT =',P -> ATTRIB.ITEM,'$',
402790 'UNITARY UNIARY OPERATOR =',REL,'$',
402800 'FORCED RESULT =',EMPTY$$)
402810 (A,2 (SKIP,A,A).SKIP,A);
402820 2 0 LIST -> ATTRIB.ITEM = '.EMPTY';
402830 2 0 LIST -> ATTRIB.NEXT = NULL (1);
402840
XEC02350
XEC02360
XEC02370
XEC02380
XEC02390
XEC02400
XEC02410
XEC02420
XEC02430
XEC02440
XEC02450
XEC02460
XEC02470
XEC02480
XEC02490
XEC02500
XEC02510
XEC02520
XEC02530
XEC02540
XEC02550
XEC02560
XEC02570
XEC02580
XEC02590
XEC02600
XEC02610
XEC02620
XEC02630
XEC02640
XEC02650
XEC02660
XEC02670
XEC02680
XEC02690
XEC02700
XEC02710
XEC02720
XEC02730
XEC02740
XEC02750
XEC02760
XEC02770
XEC02780
XEC02790
XEC02800
XEC02810
XEC02820
XEC02830

```

PL/I OPTIMIZING COMPILER 1XECUTE: PROC (XTREE,ENTITY,XCHNGE);

NUMBER LEV NT

402840 2 0 UNIT_VERT = 'O'B;
 402850 2 0 RETURN ('1'B);
 402870 2 0 END RET_UNITARY;

XECO2840
 XECO2850
 XECO2860
 XECO2870
 XECO2880

PL/I OPTIMIZING COMPILER IEXECUTE: PROC (XTREE.ENTITY,XCHNGE);

NUMBER LEV NT

```

402890    1    0    UNARY: PROC (ITEM,REL) RETURNS (CHAR (50) VAR);
402910    2    0    DCL ITEM CHAR (50) VAR;
          REL CHAR (30) VAR;
402940    2    0    ON CONVERSION GOTO MESS;
402950    2    0    ON OVERFLOW GOTO MESS;
402960    2    0    SELECT (REL);
402970    2    1    WHEN ('^')
          IF ITEM = ':FALSE' THEN
          RETURN (':TRUE');
          ELSE IF ITEM = ':TRUE' THEN
          RETURN (':FALSE');
          WHEN ('ZER')
          IF FIXED (ITEM) = 0 THEN
          RETURN (CHAR (1));
          ELSE
          RETURN (CHAR (0));
          WHEN ('POS')
          IF FIXED (ITEM) > 0 THEN
          RETURN (CHAR (1));
          ELSE
          RETURN (CHAR (0));
          WHEN ('-P')
          RETURN (CHAR (-1 * FIXED (ITEM)));
          WHEN ('SGN')
          RETURN (CHAR (SIGN (FIXED (ITEM))));
          WHEN ('ABS')
          RETURN (CHAR (ABS (FIXED (ITEM))));
          OTHERWISE;
403180    2    1    END;
403190    2    1    END;
403210    2    0    MESS:
          PUT FILE (ERROR) SKIP EDIT
          ('CONV/OVERFLOW/BAD ARG ERROR FOR UNARY_OP',
          'ITEM ARGUMENT =$',ITEM,'$$',
          'MIXED UNARY OPERATOR =$',REL,'$$',
          'FORCED RESULT =$:EMPTY$$')
          (A,2 (SKIP.A,A),SKIP.A);
403280    2    0    RETURN (':EMPTY');
403300    2    0    END UNARY;
XEC02890
XEC02900
XEC02910
XEC02920
XEC02930
XEC02940
XEC02950
XEC02960
XEC02970
XEC02980
XEC02990
XEC03000
XEC03010
XEC03020
XEC03030
XEC03040
XEC03050
XEC03060
XEC03070
XEC03080
XEC03090
XEC03100
XEC03110
XEC03120
XEC03130
XEC03140
XEC03150
XEC03160
XEC03170
XEC03180
XEC03190
XEC03200
XEC03210
XEC03220
XEC03230
XEC03240
XEC03250
XEC03260
XEC03270
XEC03280
XEC03290
XEC03300
XEC03310

```

PL/I OPTIMIZING COMPILER 1XECUTE: PROC (XTREE,ENTITY,XCHNGE);

NUMBER LEV NT

```

403320 1 0 MULTEQ2: PROC (LIST,MULT_ITEMS);
/*MODIFIES LIST*/
403350 2 0 DCL (LIST,P) PTR,
(MULT_ITEMS,MULT_ITEMS1) CHAR (80) VAR,
STOP BIT (1);
403390 2 0 P = LIST;
403400 2 0 DO WHILE (P ^= NULL ());
403410 2 1 MULT_ITEMS1 = MULT_ITEMS;
403420 2 1 STOP = 'O'B;
403430 2 1 DO WHILE (^STOP & MULT_ITEMS1 ^= '');
403440 2 2 STOP = IMPLIE (P -> ATTRIB.ITEM.GETS (MULT_ITEMS1,''));
403450 2 2 END;
403460 2 1 IF STOP THEN
P -> ATTRIB.ITEM = 'TRUE';
ELSE
P -> ATTRIB.ITEM = 'FALSE';
403500 2 1 P = P -> ATTRIB.NEXT;
403510 2 1 END;
403530 2 0 END MULTEQ2;
XEC03320
XEC03330
XEC03340
XEC03350
XEC03360
XEC03370
XEC03380
XEC03390
XEC03400
XEC03410
XEC03420
XEC03430
XEC03440
XEC03450
XEC03460
XEC03470
XEC03480
XEC03490
XEC03500
XEC03510
XEC03520
XEC03530
XEC03540

```

PL/I OPTIMIZING COMPILER 1XECUTE: PROC (XTREE.ENTITY,XCHNGE);

NUMBER LEV NT

```

403550 1 0 GO_DOWN2: PROC (LIST1,LIST2,UNIT_VERT1,UNIT_VERT2,REL)
      RETURNS (BIT (1));
      /*MODIFIES LIST1 OR LIST2*/
403590 2 0 DCL (LIST1,LIST2,P (2)) PTR,
      (UNIT_VERT1,UNIT_VERT2,UNIT_VERT,ONE_VERT) BIT (1),
      REL_CHAR (30) VAR,
      STORE_NUM FIXED;
403640 2 0 UNIT_VERT = UNIT_VERT1 ; UNIT_VERT2;
403650 2 0 IF ^UNIT_VERT THEN
      LIST1 -> ATTRIB.ITEM = BINARY (LIST1 -> ATTRIB.ITEM,
      LIST2 -> ATTRIB.ITEM,REL);
      ELSE DO;
403680 2 0 STORE_NUM = 1;
403690 2 1 ONE_VERT = ^ (UNIT_VERT1 & UNIT_VERT2);
403700 2 1 IF ^UNIT_VERT1 THEN
403710 2 1 STORE_NUM = 2;
403730 2 1 P (1) = LIST1;
403740 2 1 P (2) = LIST2;
403750 2 1 DO WHILE (P (STORE_NUM) ^= NULL ());
403760 2 2 P (STORE_NUM) -> ATTRIB.ITEM =
      BINARY (P (1) -> ATTRIB.ITEM,
      P (2) -> ATTRIB.ITEM,REL);
403790 2 2 IF ONE_VERT THEN
      P (STORE_NUM) = P (STORE_NUM) -> ATTRIB.NEXT;
      ELSE DO;
403810 2 2 P (1) = P (1) -> ATTRIB.NEXT;
403820 2 3 P (2) = P (2) -> ATTRIB.NEXT;
403830 2 3 END;
403840 2 3 END;
403850 2 2 END;
403860 2 1 END;
403870 2 0 LIST1 = P (STORE_NUM);
403880 2 0 LIST2 = NULL (1);
403890 2 0 CALL DISPOSE (P (3 - STORE_NUM));
403900 2 0 RETURN (UNIT_VERT);
403920 2 0 END GO_DOWN2;
XEC03550
XEC03560
XEC03570
XEC03580
XEC03590
XEC03600
XEC03610
XEC03620
XEC03630
XEC03640
XEC03650
XEC03660
XEC03670
XEC03680
XEC03690
XEC03700
XEC03710
XEC03720
XEC03730
XEC03740
XEC03750
XEC03760
XEC03770
XEC03780
XEC03790
XEC03800
XEC03810
XEC03820
XEC03830
XEC03840
XEC03850
XEC03860
XEC03870
XEC03880
XEC03890
XEC03900
XEC03910
XEC03920
XEC03930

```

PL/1 OPTIMIZING COMPILER 1XECUTE: PROC (XTREE,ENTITY,XCHANGE);

NUMBER LEV NT

403940	1	0	BINARY: PROC (ITEM1,ITEM2,REL) RETURNS (CHAR (50) VAR);
403960	2	0	DCL (ITEM1,ITEM2) CHAR (50) VAR, REL CHAR (30) VAR;
403990	2	0	ON CONVERSION GOTO MESS;
404000	2	0	ON OVERFLOW GOTO MESS;
404010	2	0	ON ZERODIVIDE GOTO MESS;

XEC03940
XEC03950
XEC03960
XEC03970
XEC03980
XEC03990
XEC04000
XEC04010

PL/I OPTIMIZING COMPILER 1XECUTE: PROC (XTREE,ENTITY,XCHNGE);

NUMBER LEV NT

```

404020 2 0 SELECT (REL);
404030 2 1 WHEN ('=')
      IF ITEM1 = ITEM2
      || TMPLTE (ITEM1,ITEM2)
      || TMPLTE (ITEM2,ITEM1) THEN
      RETURN (':TRUE');
      ELSE
      RETURN (':FALSE');
404100 2 1 WHEN ('>')
      IF ITEM1 > ITEM2 THEN
      RETURN (':TRUE');
      ELSE
      RETURN (':FALSE');
404130 2 1 WHEN ('<')
      IF ITEM1 < ITEM2 THEN
      RETURN (':TRUE');
      ELSE
      RETURN (':FALSE');
404200 2 1 WHEN ('AND')
      IF ITEM1 = ':TRUE' & ITEM2 = ':TRUE' THEN
      RETURN (':TRUE');
      ELSE
      RETURN (':FALSE');
404230 2 1 WHEN ('OR')
      IF ITEM1 = ':TRUE' || ITEM2 = ':TRUE' THEN
      RETURN (':TRUE');
      ELSE
      RETURN (':FALSE');
404280 2 1 WHEN ('XOR')
      IF (ITEM1 = ':TRUE' & ITEM2 = ':FALSE')
      || (ITEM1 = ':FALSE' & ITEM2 = ':TRUE') THEN
      RETURN (':TRUE');
      ELSE
      RETURN (':FALSE');
404340 2 1 WHEN ('+')
      RETURN (CHAR (FIXED (ITEM1) + FIXED (ITEM2)));
404360 2 1 WHEN ('-')
      RETURN (CHAR (FIXED (ITEM1) - FIXED (ITEM2)));
404400 2 1 WHEN ('*')
      RETURN (CHAR (FIXED (ITEM1) * FIXED (ITEM2)));
404420 2 1 WHEN ('/')
      RETURN (CHAR (FIXED (FIXED (ITEM1) / FIXED (ITEM2))));
404440 2 1 WHEN ('1')
      RETURN (CHAR (FIXED (ITEM1) ** FIXED (ITEM2)));
404460 2 1 WHEN ('1')
      RETURN (ITEM1 || ITEM2);
404480 2 1 OTHERWISE DO;
404490 2 2 PUT FILE (ERROR) SKIP EDIT
      ('BINARY OPERATOR NOT SUPPORTED =$',REL,'$$').

```

PL/I OPTIMIZING COMPILER
1XECUTE: PROC (XTREE,ENTITY,XCHNGE):

NUMBER LEV NT

		'ITEM1 ARGUMENT =' , ITEM1, '\$\$',	XEC04510
		'ITEM2 ARGUMENT =' , ITEM2, '\$\$',	XEC04520
		'FORCED RESULT =' , \$:EMPTY\$')	XEC04530
		(3 (SKIP.A,A), SKIP.A);	XEC04540
404550	2 2	RETURN (';EMPTY');	XEC04550
404560	2 2	END;	XEC04560
404570	2 1	END;	XEC04570

PL/I OPTIMIZING COMPILER 1XECUTE: PROC (XTREE,ENTITY,XCHANGE);

NUMBER LEV NT

```

404580 2 0 MESS:
      PUT FILE (ERROR) SKIP EDIT
      ('CONV/OVFLOW/BAD_ARG ERROR FOR BINARY_OP',
       'ITEM1 ARGUMENT ='$,ITEM1,$$,
       'ITEM2 ARGUMENT ='$,ITEM2,$$,
       'MIXED BINARY OPERATOR ='$,REL,$$,
       'FORCED RESULT ='$,EMPTY$$)
      (A.3 (SKIP.A.A.A).SKIP.A);
404660 2 0 RETURN (':EMPTY');
404680 2 0 END BINARY;

```

XEC04580
 XEC04590
 XEC04600
 XEC04610
 XEC04620
 XEC04630
 XEC04640
 XEC04650
 XEC04660
 XEC04670
 XEC04680
 XEC04690

PL/I OPTIMIZING COMPILER 1XECUTE: PROC (XTREE,ENTITY,XCHANGE);

```

NUMBER  LEV  NT
404700   1   0  CREATE_LIST: PROC (LIST,USE_NUM) RETURNS (BIT (1));
/*MODIFIES LIST*/
404730   2   0  DCL (LIST,P,Q,TAILO) PTR,
              (USE_NUM,N) FIXED;
404760   2   0  IF ^ENTITY (CUR_ES).ATTR (USE_NUM).SING_OCC THEN DO;
404770   2   1  PUT FILE (ERROR) SKIP EDIT
              ('ATTEMPT TO OPERATE ON NON-SING OCC ITEM',
              'ITEM ARGUMENT =$',ENTITY (CUR_ES).ATTR (USE_NUM).USES,'$$',
              'FORCED RESULT =$:EMPTY$$')
              (A,SKIP,A,A,SKIP,A);
404820   2   1  RETURN (CREATE_CONST (LIST,':EMPTY'));
404830   2   1  END;
404840   2   0  LIST = NULL();
404850   2   0  N = 0;
404860   2   0  P = ENTITY (CUR_ES).ATTR (USE_NUM).LIST;
404870   2   0  DO WHILE (P ^= NULL ());
404880   2   1  N = N + 1;
404890   2   1  ALLOCATE ATTRIB SET (Q);
404900   2   1  Q -> ATTRIB.ITEM = P -> ATTRIB.ITEM;
404910   2   1  IF LIST = NULL ( ) THEN
              LIST = Q;
404930   2   1  ELSE
              TAILQ -> ATTRIB.NEXT = Q;
404950   2   1  TAILQ = Q;
404960   2   1  P = P -> ATTRIB.NEXT;
404970   2   1  END;
404980   2   0  TAILQ -> ATTRIB.NEXT = NULL ( );
404990   2   0  RETURN ('1'B);
405010   2   0  END CREATE_LIST;
XEC04700
XEC04710
XEC04720
XEC04730
XEC04740
XEC04750
XEC04760
XEC04770
XEC04780
XEC04790
XEC04800
XEC04810
XEC04820
XEC04830
XEC04840
XEC04850
XEC04860
XEC04870
XEC04880
XEC04890
XEC04900
XEC04910
XEC04920
XEC04930
XEC04940
XEC04950
XEC04960
XEC04970
XEC04980
XEC04990
XEC05000
XEC05010
XEC05020

```

PL/I OPTIMIZING COMPILER EXECUTE PROC (XFREE,ENTITY,XCHANGE);

NUMBER LEV NT

```

405030 1 0 CREATE_CONST: PROC (LIST,CONSTNM) RETURNS (BIT (1));
/*MODIFIES LIST*/
405060 2 0 DCL LIST PTR,
      CONSTNM CHAR (30) VAR;
405090 2 0 ALLOCATE ATTRIB SET (LIST);
405100 2 0 LIST -> ATTRIB.ITEM = CONSTNM;
405110 2 0 LIST -> ATTRIB.NEXT = NULL ();
405120 2 0 RETURN ('O'B);
405140 2 0 END CREATE_CONST;

```

XEC05030
 XEC05040
 XEC05050
 XEC05060
 XEC05070
 XEC05080
 XEC05090
 XEC05100
 XEC05110
 XEC05120
 XEC05130
 XEC05140
 XEC05150

PL/I OPTIMIZING COMPILER 1XECUTE: PROC (XTREE,ENTITY,XCHNGE);

NUMBER LEV NT

```

405160 1 0 TMLPTE: PROC (ITEM,TEMP_ITEM) RETURNS (BIT (1)) RECURSIVE;
405180 2 0 DCL (ITEM,TEMP_ITEM) CHAR (*) VAR.
      I FIXED;
405210 2 0 IF TEMP_ITEM = '' THEN
      IF ITEM = '' THEN
      RETURN ('1'B);
405240 2 0 ELSE
      RETURN ('0'B);
405260 2 0 ELSE IF SUBSTR (TEMP_ITEM,1,1) = '1' THEN
      IF ITEM = '' & LENGTH (TEMP_ITEM) = 1 THEN
      RETURN ('1'B);
405290 2 0 ELSE DO;
405300 2 1 DO I = 1 TO LENGTH (ITEM) + 1;
405310 2 2 IF TMLPTE (SUBSTR (ITEM,I),SUBSTR (TEMP_ITEM,2)) THEN
      RETURN ('1'B);
      END;
405330 2 2 RETURN ('0'B);
405340 2 1 RETURN ('0'B);
405350 2 1 END;
405360 2 0 ELSE IF ITEM = '' THEN
      RETURN ('0'B);
405380 2 0 ELSE IF SUBSTR (TEMP_ITEM,1,1) = '-' THEN
      RETURN (TMLPTE (SUBSTR (ITEM,2),SUBSTR (TEMP_ITEM,2)));
405400 2 0 ELSE
      RETURN (SUBSTR (ITEM,1,1) = SUBSTR (TEMP_ITEM,1,1)
      & TMLPTE (SUBSTR (ITEM,2),SUBSTR (TEMP_ITEM,2)));
405440 2 0 END TMLPTE;

```

PL/I OPTIMIZING COMPILER IEXECUTE: PROC (XTREE,ENTITY,XCHANGE);

NUMBER LEV NT

```

405460    1    0    DISPOSE: PROC (LIST);
405480    2    0    DCL (LIST,PNOW,PLAST) PTR;

405500    2    0    PNOW = LIST;
405510    2    0    DO WHILE (PNOW ^= NULL);
405520    2    1    PLAST = PNOW;
405530    2    1    PNOW = PNOW -> ATTRIB.NEXT;
405540    2    1    FREE PLAST -> ATTRIB;
405550    2    1    END;
405560    2    0    LIST = NULL ();

405580    2    0    END DISPOSE;

```

```

XEC05460
XEC05470
XEC05480
XEC05490
XEC05500
XEC05510
XEC05520
XEC05530
XEC05540
XEC05550
XEC05560
XEC05570
XEC05580
XEC05590

```

PL/I OPTIMIZING COMPILER 1XECUTE PROC (XTREE.ENTITY.XCHNGE);

NUMBER LEV NT

```

405600 1 0 MUNION: PROC;
405620 2 0 DCL (I,J) FIXED,
      STOP BIT (1);

405650 2 0 COPY_COUNT = 0;
405660 2 0 CALL LOCATEKEYS;
405670 2 0 DO I = 1, 2;
405680 2 1 DO J = 1 TO 15;
405690 2 2 ATAIL (I,J) = ENTITY (ENTITY (CUR_ES).N_PARENT (I))
      .ATTR (J).LIST;
405710 2 2 END;
405720 2 1 END;
405730 2 0 ATAIL (0,*) = NULL ();
405740 2 0 NEWLVL (*,*) = 0;
405750 2 0 DO WHILE (ATAIL (1,1) ^= NULL ());
405760 2 1 CALL COPYSET (ENTITY (CUR_ES).N_PARENT (1),
      1,
      ENTITY (CUR_ES).VES_PAR -> N_MAP (1).NUM (*));
405790 2 1 END;
405800 2 0 DO WHILE (ATAIL (2,1) ^= NULL ());
405810 2 1 DO I = 1 TO 15;
405820 2 2 ATAIL (1,I) = ENTITY (ENTITY (CUR_ES).N_PARENT (1))
      .ATTR (I).LIST;
405840 2 2 END;
405850 2 1 NEWLVL (1,*) = 0;
405860 2 1 STOP = 'O'B;
405870 2 1 DO WHILE (^STOP & ATAIL (1,1) ^= NULL ());
405880 2 2 STOP = IUMATCH (ENTITY (CUR_ES).VES_PAR -> N_MAP);
405890 2 2 CALL SKIPSET (ENTITY (CUR_ES).N_PARENT (1),1);
405900 2 2 END;
405910 2 1 IF STOP THEN
      CALL SKIPSET (ENTITY (CUR_ES).N_PARENT (2),2);
      ELSE
      CALL COPYSET (ENTITY (CUR_ES).N_PARENT (2),
      2,
      ENTITY (CUR_ES).VES_PAR -> N_MAP (2).NUM (*));
405930 2 1 END;

405970 2 1 END;
405980 2 0 ENTITY (CUR_ES).DEPTH = COPY_COUNT;

406000 2 0 END MUNION;

```

PL/I OPTIMIZING COMPILER 1XECUTE: PROC (XTREE.ENTITY,XCHNGE);

```

NUMBER LEV NT
406030 1 0 MINTER: PROC;
406050 2 0 DCL I FIXED,
      STOP BIT (1);
406080 2 0 COPY COUNT = 0;
406090 2 0 CALL LOCATEKEYS;
406100 2 0 DO I = 1 TO 15;
406110 2 1 ATAIL (1,1) = ENTITY (ENTITY (CUR_ES).N_PARENT (1)).ATTR (1).LIST;
406120 2 1 END;
406130 2 0 ATAIL (0,*) = NULL ();
406140 2 0 NEWLVL (*,*) = 0;
406150 2 0 DO WHILE (ATAIL (1,1) ^= NULL ());
406160 2 1 DO I = 1 TO 15;
406170 2 2 ATAIL (2,I) = ENTITY (ENTITY (CUR_ES).N_PARENT (2))
      .ATTR (I).LIST;
406190 2 2 END;
406200 2 1 NEWLVL (2,*) = 0;
406210 2 1 STOP = '0'B;
406220 2 1 DO WHILE (^STOP & ATAIL (1,1) ^= NULL ());
406230 2 2 STOP = IUMATCH (ENTITY (CUR_ES).VES_PAR -> N_MAP);
406240 2 2 CALL SKIPSET (ENTITY (CUR_ES).N_PARENT (2),2);
406250 2 2 END;
406260 2 1 IF STOP THEN
      CALL COPYSET (ENTITY (CUR_ES).N_PARENT (1),
1,
      ENTITY (CUR_ES).VES_PAR -> N_MAP (1).NUM (*));
406300 2 1 ELSE
      CALL SKIPSET (ENTITY (CUR_ES).N_PARENT (1),1);
406320 2 1 END;
406330 2 0 ENTITY (CUR_ES).DEPTH = COPY_COUNT;
406350 2 0 END MINTER;
XEC06030
XEC06040
XEC06050
XEC06060
XEC06070
XEC06080
XEC06090
XEC06100
XEC06110
XEC06120
XEC06130
XEC06140
XEC06150
XEC06160
XEC06170
XEC06180
XEC06190
XEC06200
XEC06210
XEC06220
XEC06230
XEC06240
XEC06250
XEC06260
XEC06270
XEC06280
XEC06290
XEC06300
XEC06310
XEC06320
XEC06330
XEC06340
XEC06350
XEC06360
XEC06370

```

PL/I OPTIMIZING COMPILER 1XECUTE: PROC (XTREE,ENTITY,XCHANGE);

NUMBER LEV NT

```

406380 1 0 SINGLE: PROC;
406400 2 0 DCL 1 ENTMAP (2);
406420 2 0 DCL (I,J) FIXED;
      STOP BIT (1);
406450 2 0 COPY_COUNT = 0;
406460 2 0 ENTITY (O) = ENTITY (CUR_ES);
406470 2 0 ENTITY (CUR_ES).ATTR (*)LIST = NULL ();
406480 2 0 DO I = 1 TO 15;
406490 2 1 ENTMAP (*).NUM (I) = I;
406500 2 1 ATAIL (1,I) = ENTITY (O).ATTR (I).LIST;
406510 2 1 END;
406520 2 0 ATAIL (O,*) = NULL ();
406530 2 0 NEWLVL (*,*) = 0;
406540 2 0 DO WHILE (ATAIL (1,1) ^= NULL ());
406550 2 1 IF ATAIL (1,1) -> ATTRIB.ITEM ^= ':EMPTYFILL' THEN DO;
406560 2 2 ATAIL (2,*) = ATAIL (1,*);
406570 2 2 NEWLVL (2,*) = NEWLVL (1,*);
406580 2 2 CALL SKIPSET (O,2);
406590 2 2 STOP = 'O'B;
406600 2 2 DO WHILE (^STOP & ATAIL (2,1) ^= NULL ());
406610 2 3 IF ATAIL (2,1) -> ATTRIB.ITEM ^= ':EMPTYFILL' THEN
      STOP = IUMATCH (ENTMAP);
      CALL SKIPSET (O,2);
      END;
406630 2 3 END;
406640 2 3 IF STOP THEN
406650 2 2 ATAIL (1,1) -> ATTRIB.ITEM = ':EMPTYFILL';
      END;
406670 2 2 CALL SKIPSET (O,1);
406680 2 1 END;
406690 2 1 DO I = 1 TO 15;
406700 2 0 ATAIL (1,I) = ENTITY (O).ATTR (I).LIST;
406710 2 1 END;
406720 2 0 NEWLVL (1,*) = 0;
406730 2 0 DO WHILE (ATAIL (1,1) ^= NULL ());
406740 2 1 IF ATAIL (1,1) -> ATTRIB.ITEM = ':EMPTYFILL' THEN
      CALL SKIPSET (O,1);
      ELSE
406770 2 1 CALL COPYSET (O,
      1,
      ENTMAP (1).NUM (*));
406810 2 1 END;
406820 2 0 DO I = 1 TO 15 WHILE (ENTITY (O).ATTR (I).LIST ^= NULL ());
406830 2 1 CALL DISPOSE (ENTITY (O).ATTR (I).LIST);
406840 2 1 END;
406850 2 0 ENTITY (CUR_ES).DEPTH = COPY_COUNT;

```

```

XEC06380
XEC06390
XEC06400
XEC06410
XEC06420
XEC06430
XEC06440
XEC06450
XEC06460
XEC06470
XEC06480
XEC06490
XEC06500
XEC06510
XEC06520
XEC06530
XEC06540
XEC06550
XEC06560
XEC06570
XEC06580
XEC06590
XEC06600
XEC06610
XEC06620
XEC06630
XEC06640
XEC06650
XEC06660
XEC06670
XEC06680
XEC06690
XEC06700
XEC06710
XEC06720
XEC06730
XEC06740
XEC06750
XEC06760
XEC06770
XEC06780
XEC06790
XEC06800
XEC06810
XEC06820
XEC06830
XEC06840
XEC06850
XEC06860

```

PL/1 OPTIMIZING COMPILER

1EXECUTE: PROC (XTREE,ENTITY,XCHANGE):

NUMBER LEV NT

406870 2 0 END SINGLE;

XEC06870
XEC06880
XEC06890

PAGE 184

PL/I OPTIMIZING COMPILER 1XECUTE: PROC (XTREE.ENTITY.XCHANGE);

```

NUMBER  LEV  NT
406900   1   0  CRTESN. PROC;
406920   2   0  DCL (I,N,REPEATVAL) FIXED;
406940   2   0  COPY_COUNT = 0;
406950   2   0  DO I = 1 TO 15;
406960   2   1  ATAIL (2,1) = ENTITY (ENTITY (CUR_ES).N_PARENT (1)).ATTR (1).LIST;
406970   2   1  END;
406980   2   0  ATAIL (0,*) = NULL;
406990   2   0  NEWLVL (*,*) = 0;
407000   2   0  REPEATVAL = ENTITY (ENTITY (CUR_ES).N_PARENT (2)).DEPTH;
407010   2   0  DO WHILE (ATAIL (2,1) ^= NULL ());
407020   2   1  DO N = 1 TO REPEATVAL;
407030   2   2  ATAIL (1,*) = ATAIL (2,*);
407040   2   2  NEWLVL (1,*) = 0;
407050   2   2  CALL COPYSET (ENTITY (CUR_ES).N_PARENT (1),
1.
1.
ENTITY (CUR_ES).VES_PAR -> N_MAP (1).NUM (*));
407080   2   2  END;
407090   2   1  CALL SKIPSET (ENTITY (CUR_ES).N_PARENT (1),2);
407100   2   1  END;
407110   2   0  ENTITY (CUR_ES).DEPTH = COPY_COUNT;
407120   2   0  DO I = 1 TO 15;
407130   2   1  ATAIL (1,1) = ENTITY (ENTITY (CUR_ES).N_PARENT (2)).ATTR (1).LIST;
407140   2   1  END;
407150   2   0  ATAIL (0,*) = NULL (1);
407160   2   0  NEWLVL (*,*) = 0;
407170   2   0  REPEATVAL = ENTITY (ENTITY (CUR_ES).N_PARENT (1)).DEPTH;
407180   2   0  DO N = 1 TO REPEATVAL;
407190   2   1  ATAIL (2,*) = ATAIL (1,*);
407200   2   1  NEWLVL (2,*) = 0;
407210   2   1  DO WHILE (ATAIL (2,1) ^= NULL ());
407220   2   2  CALL COPYSET (ENTITY (CUR_ES).N_PARENT (2),
2.
2.
ENTITY (CUR_ES).VES_PAR -> N_MAP (2).NUM (*));
407250   2   2  END;
407260   2   1  END;
407280   2   0  END CRTESN;

```

XEC06900
 XEC06910
 XEC06920
 XEC06930
 XEC06940
 XEC06950
 XEC06960
 XEC06970
 XEC06980
 XEC06990
 XEC07000
 XEC07010
 XEC07020
 XEC07030
 XEC07040
 XEC07050
 XEC07060
 XEC07070
 XEC07080
 XEC07090
 XEC07100
 XEC07110
 XEC07120
 XEC07130
 XEC07140
 XEC07150
 XEC07160
 XEC07170
 XEC07180
 XEC07190
 XEC07200
 XEC07210
 XEC07220
 XEC07230
 XEC07240
 XEC07250
 XEC07260
 XEC07270
 XEC07280
 XEC07290
 XEC07300

PL/I OPTIMIZING COMPILER 1XECUTE: PROC (XTREE, ENTITY, XCHNGE);

NUMBER LEV NT

```

407310    1    0    SKIPSET: PROC (SK_ENT, ENT_NUM);
407330    2    0    DCL (SK_ENT, ENT_NUM, I, LVL_LIM) FIXED;
407350    2    0    DO I = 1 TO 15 WHILE (ENTITY (SK_ENT).ATTR (I).USES ^= '');
407360    2    1    IF ENTITY (SK_ENT).ATTR (I).SING_OCC THEN
407370    2    1    LVL_LIM = NEWLVL (ENT_NUM, I) + 1;
407380    2    1    ELSE
407390    2    1    LVL_LIM = NEWLVL (ENT_NUM, ENTITY (SK_ENT).ATTR (I).A_PARENT);
407400    2    1    DO WHILE (ATAIL (ENT_NUM, I) ^= NULL ())
407410    2    2    & ATAIL (ENT_NUM, I) -> ATTRIB.LEVEL <= LVL_LIM;
407420    2    2    NEWLVL (ENT_NUM, I) = NEWLVL (ENT_NUM, I) + 1;
407430    2    2    ATAIL (ENT_NUM, I) = ATAIL (ENT_NUM, I) -> ATTRIB.NEXT;
407440    2    2    END;
407450    2    1    END;
407470    2    0    END SKIPSET;

```

```

XEC07310
XEC07320
XEC07330
XEC07340
XEC07350
XEC07360
XEC07370
XEC07380
XEC07390
XEC07400
XEC07410
XEC07420
XEC07430
XEC07440
XEC07450
XEC07460
XEC07470
XEC07480

```

PL/I OPTIMIZING COMPILER EXECUTE: PROC (XTREE, ENTITY, XCHNGE);

```

NUMBER LEV NT
407490 1 0 COPYSET: PROC (PAR_ENT, PAR_NUM, MAP_NUM);
407510 2 0 DCL (PAR_ENT, PAR_NUM, I, LVL_ST, LVL_LIM, OLDLVL (0:2, 15)) FIXED;
407520 2 0 DCL MAP_NUM (15) FIXED;
407530 2 0 DCL P PTR;
407550 2 0 OLDLVL (*, *) = NEWLVL (*, *);
407560 2 0 DO I = 1 TO 15 WHILE (ENTITY (CUR_ES).ATTR (I).USES ^= '');
407570 2 1 IF MAP_NUM (I) > 0
      & ENTITY (PAR_ENT).ATTR (MAP_NUM (I)).SING_OCC THEN DO;
      LVL_ST = OLDLVL (PAR_NUM, MAP_NUM (I));
      LVL_LIM = NEWLVL (PAR_NUM, MAP_NUM (I)) + 1;
      END;
      ELSE DO;
      LVL_ST = OLDLVL (PAR_NUM,
        ENTITY (PAR_ENT).ATTR (MAP_NUM (I)).A_PARENT);
      LVL_LIM = NEWLVL (PAR_NUM,
        ENTITY (PAR_ENT).ATTR (MAP_NUM (I)).A_PARENT);
407650 2 2
407670 2 2
407680 2 1 DO WHILE (ATAIL (PAR_NUM, MAP_NUM (I)) ^= NULL ())
      & ATAIL (PAR_NUM, MAP_NUM (I)) -> ATTRIB.LEVEL <= LVL_LIM);
407700 2 2 NEWLVL (O, I) = NEWLVL (O, I) + 1;
407710 2 2 NEWLVL (PAR_NUM, MAP_NUM (I)) = NEWLVL (PAR_NUM, MAP_NUM (I)) + 1;
407720 2 2 ALLOCATE ATTRIB SET (P);
407730 2 2 P -> ATTRIB.ITEM = ATAIL (PAR_NUM, MAP_NUM (I)) -> ATTRIB.ITEM;
407740 2 2 P -> ATTRIB.LEVEL = OLDLVL (O, I)
      + ATAIL (PAR_NUM, MAP_NUM (I)) -> ATTRIB.LEVEL;
407770 2 2 IF ENTITY (CUR_ES).ATTR (I).LIST = NULL () THEN
      ENTITY (CUR_ES).ATTR (I).LIST = P;
      ELSE
      ATAIL (O, I) -> ATTRIB.NEXT = P;
407810 2 2 ATAIL (O, I) = P;
407820 2 2 ATAIL (PAR_NUM, MAP_NUM (I)) = ATAIL (PAR_NUM, MAP_NUM (I))
      -> ATTRIB.NEXT;
      END;
407840 2 2
407850 2 1 END;
407860 2 0 COPY_COUNT = COPY_COUNT + 1;
407880 2 0 END COPYSET;

```

PL/I OPTIMIZING COMPILER 1XECUTE: PROC (XTREE.ENTITY.XCHANGE);

NUMBER LEV NT

```

407900 1 0 COPYALL: PROC (FR_ENT);
407920 2 0 DCL (FR_ENT.1,J) FIXED;
407940 2 0 ENTITY (CUR_ES).DEPTH = ENTITY (FR_ENT).DEPTH;
407950 2 0 ENTITY (CUR_ES).ATTR (*) .VES_KEY = ENTITY (FR_ENT).ATTR (*) .VES_KEY;
407960 2 0 ENTITY (CUR_ES).ATTR (*) .CART_KEY = ENTITY (FR_ENT).ATTR (*) .CART_KEY;
407970 2 0 ENTITY (CUR_ES).ATTR (*) .SING_OCC = ENTITY (FR_ENT).ATTR (*) .SING_OCC;
407980 2 0 ENTITY (CUR_ES).ATTR (*) .A_PARENT = ENTITY (FR_ENT).ATTR (*) .A_PARENT;
407990 2 0 ENTITY (CUR_ES).ATTR (*) .USES = ENTITY (FR_ENT).ATTR (*) .USES;
408000 2 0 DO I = 1 TO 15 WHILE (ENTITY (FR_ENT).ATTR (I).USES ^= '');
408010 2 1 O = ENTITY (FR_ENT).ATTR (I).LIST;
408020 2 1 DO WHILE (Q ^= NULL ());
408030 2 2 ALLOCATE ATTRIB SET (P);
408040 2 2 P -> ATTRIB.ITEM = Q -> ATTRIB.ITEM;
408050 2 2 P -> ATTRIB.LEVEL = Q -> ATTRIB.LEVEL;
408060 2 2 IF ENTITY (CUR_ES).ATTR (I).LIST = NULL ( ) THEN
      ENTITY (CUR_ES).ATTR (I).LIST = P;
408080 2 2 ELSE
      TAILP -> ATTRIB.NEXT = P;
408100 2 2 TAILP = P;
408110 2 2 Q = Q -> ATTRIB.NEXT;
408120 2 2 END;
408130 2 1 TAILP -> ATTRIB.NEXT = NULL ( );
408140 2 1 END;
408160 2 0 END COPYALL;
408170

```

PL/I OPTIMIZING COMPILER 1XECUTE: PROC (XTREE,ENTITY,XCHANGE);

NUMBER LEV NT

```

408180 1 0 IUMATCH: PROC (ENTMAP) RETURNS (BIT (1));
408200 2 0 DCL 1 ENTMAP (2),
          2 NUM (15) FIXED;
408230 2 0 DO I = 1 TO 15 WHILE ((KEYSET (I)) ^= 0);
408240 2 1 IF ATAIL (1,ENTMAP (1)).NUM (KEYSET (I))) -> ATTRIB.ITEM
          ^= ATAIL (2,ENTMAP (2)).NUM (KEYSET (I))) -> ATTRIB.ITEM THEN
          RETURN ('O'B);
408270 2 1 END;
408280 2 0 RETURN ('1'B);
408300 2 0 END IUMATCH;
XEC08180
XEC08190
XEC08200
XEC08210
XEC08220
XEC08230
XEC08240
XEC08250
XEC08260
XEC08270
XEC08280
XEC08290
XEC08300
XEC08310

```

PL/I OPTIMIZING COMPILER IEXECUTE: PROC (XTREE,ENTITY,XCHNGE);

NUMBER LEV NT

```

408320 1 0 LOCATEKEYS: PROC;
408340 2 0 DCL (I,J) FIXED;
408360 2 0 KEYSET (*) = 0;
408370 2 0 J = 0;
408380 2 0 DO I = 1 TO 15 WHILE (ENTITY (CUR_ES).ATTR (I).USES ^= '');
408390 2 1 IF ENTITY (CUR_ES).ATTR (I).VES_KEY THEN DO;
408400 2 2 J = J + 1;
408410 2 2 KEYSET (J) = I;
408420 2 2 END;
408430 2 1 END;
408450 2 0 END LOCATEKEYS;
408460

```

PL/I OPTIMIZING COMPILER 1XECUTE: PROC (XTREE.ENTITY,XCHANGE);

```

NUMBER  LEV  NT
408470   1   0  DECRSET: PROC;
408490   2   0  DCL ENTMAP_NUM (15) FIXED;
408500   2   0  DCL I FIXED;
408510   2   0  DCL P PTR;
408530   2   0  COPY_COUNT = 0;
408540   2   0  ENTITY (O) = ENTITY (CUR_ES);
408550   2   0  ENTITY (CUR_ES).ATTR (*)_LIST = NULL ();
408560   2   0  DO I = 1 TO 15;
408570   2   1    ENTMAP_NUM (I) = I;
408580   2   1    ATAIL (1,I) = ENTITY (O).ATTR (I).LIST;
408590   2   1    END;
408600   2   0  ATAIL (O,*) = NULL ();
408610   2   0  NEWLVL (*,*) = 0;
408620   2   0  P = SCLLIST;
408630   2   0  DO WHILE (P ^= NULL ());
408640   2   1    IF P -> ATTRIB.ITEM = 'TRUE' THEN
        CALL COPYSET (O,
            1,
            ENTMAP_NUM (*));
408680   2   1    ELSE
        CALL SKIPSET (O,1);
408700   2   1    P = P -> ATTRIB.NEXT;
408710   2   1    END;
408720   2   0  DO I = 1 TO 15 WHILE (ENTITY (O).ATTR (I).LIST ^= NULL ());
408730   2   1    CALL DISPOSE (ENTITY (O).ATTR (I).LIST);
408740   2   1    END;
408750   2   0  ENTITY (CUR_ES).DEPTH = COPY_COUNT;
408770   2   0  END DECRSET;
XEC08470
XEC08480
XEC08490
XEC08500
XEC08510
XEC08520
XEC08530
XEC08540
XEC08550
XEC08560
XEC08570
XEC08580
XEC08590
XEC08600
XEC08610
XEC08620
XEC08630
XEC08640
XEC08650
XEC08660
XEC08670
XEC08680
XEC08690
XEC08700
XEC08710
XEC08720
XEC08730
XEC08740
XEC08750
XEC08760
XEC08770
XEC08780

```

PL/I OPTIMIZING COMPILER 1XECUTE: PROC (XTREE.ENTITY.XCHANGE);

NUMBER LEV NT

```

408790 1 0 GETS: PROC (LIST,TERM_ITEM) RETURNS (BIT (1));
408810 2 0 DCL LIST CHAR (*) VAR;
      TERM_ITEM CHAR (1);
      RTN_LIST CHAR (30) VAR;
      I FIXED;
408860 2 0 I = INDEX (LIST,TERM_ITEM);
408870 2 0 IF I = 0 THEN DO;
408880 2 1 RTN_LIST = LIST;
408890 2 1 LIST = ',';
408900 2 1 END;
408910 2 0 ELSE DO;
408920 2 1 RTN_LIST = SUBSTR (LIST,I - 1);
408930 2 1 LIST = SUBSTR (LIST,I + 1);
408940 2 1 END;
408950 2 0 RETURN (RTN_LIST);
408970 2 0 END GETS;
XEC08790
XEC08800
XEC08810
XEC08820
XEC08830
XEC08840
XEC08850
XEC08860
XEC08870
XEC08880
XEC08890
XEC08900
XEC08910
XEC08920
XEC08930
XEC08940
XEC08950
XEC08960
XEC08970
XEC08980

```

1XECUTE: PROC (XTREE.ENTITY,XCHANGE):

PL/I OPTIMIZING COMPILER

NUMBER LEV NT

408990 1 0 END XECUTE;

XEC08990

DATE
ILME