January 1974

Research on Automatic Program Generation

Jesus A. Ramirez
University of Pennsylvania

N. Adam Rin
University of Pennsylvania

Maxine Brown
University of Pennsylvania

Noah S. Prywes
University of Pennsylvania

Research on Automatic Program Generation

Abstract

Automatic Program Generation Research has been conducted under Contract N00014-67-A-0216-0014, since 1971. The objective of the research has been to provide software generation directly from user specifications.

Initially, the research concentrated on a specific application, of generating file conversion programs. A first report on this subject was authored by Diane Pirog Smith, in December 1971, titled, "An Approach to Data Description and Conversion". Subsequently, a software system for automating this function was implemented by Jesus A. Ramirez and described by him in a report titled, "Automatic Generation of data conversion-programs Using A Data Description Language (DIL)". Currently, the objective of the research has been broadened to develop a user language and a software system for automatic generation of business oriented programs.

This technical report contains a collection of three papers intended to summarize the results of past research and the direction of current research. The first paper, by Ramirez, Rin and Prywes is a summmary of Dr. Ramirez's report and dissertation cited above. The second paper, titled, "An Overview of a System for Automatic Generation of File Conversion Programs " by. N. Adam Rin and Maxine Brown is intended to provide a more user oriented view based on their experience in utilization of the system developed by Ramirez.

There have been many research activities and a large number of papers in this area. The third paper, "Automatic Generation of Software Systems: A Survey," by N. Prywes serves to relate the research underway at the University of Pennsylvania, to the many recent and current activities in this field. It also aims to clearly define short and long term objectives and methodologies.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-74-05.

UNIVERSITY OF PENNSYLVANIA
The Moore School of Electrical Engineering
Department of Computer and Information Science

TECHNICAL REPORT

RESEARCH ON AUTOMATIC PROGRAM GENERATION

Project Supervisor
Noah S. Prywes

January 1974

Prepared for the
Office of Naval Research
Information Systems
Arlington, Va. 22217

under

Contract N00014-67-A-0216-0014
Project No. NR 049-153

Moore School Report # 74-05

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* University of Pennsylvania The Moore School of Electrical Engineering Dept. of Computer Information Sciences, Phila., Pa.	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED
	2b. GROUP

3. REPORT TITLE

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Technical Report

5. AUTHOR(S) *(First name, middle initial, last name)*
Jesus A. Ramirez, N. Adam Rin, Maxine Brown and Noah S. Prywes

6. REPORT DATE January 1974	7a. TOTAL NO OF PAGES 103	7b. NO. OF REFS
8a. CONTRACT OR GRANT NO. N00014-67-A-0216-0014	9a. ORIGINATOR'S REPORT NUMBER(S) Moore School Report 74-05	
b. PROJECT NO.		
c. NR 049-153	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*	
d.		

10 DISTRIBUTION STATEMENT
Reproduction in whole or in part is permitted for any purpose of the United States Government

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Office of Naval Research Information Systems Arlington, Virginia 22217

13. ABSTRACT
Automatic Program Generation Research has been conducted under contract N00014-67-A-0216-0014, since 1971. The objective of the research has been to provide software generation directly from user specifications.

Initially, the research concentrated on a specific application, of generating file conversion programs. A first report on this subject was authored by Diane Pirog Smith, in December 1971, titled, "An Approach to Data Description and Conversion." Subsequently a software system for automating this function was implemented by Jesus A. Ramirez and described by him in a report titled, "Automatic Generation of Data Conversion-Programs Using A Data Description Language (DDL)". Currently, the objective of the research has been broadened to develop a user language and a software system for automatic generation of business oriented programs.

This technical report contains a collection of three papers intended to summarize the results of past research and the direction of current research. The first paper, by Ramirez, Rin and Prywes is a summary of Dr. Ramirez's report and dissertation cited above. The second paper, titled, "An Overview Of A System For Automatic Generation of File Conversion Programs," by N. Adam Rin and Maxine Brown is intended to provide a more user oriented view based on their experience in utilization of the system developed by Ramirez.

(Continued on next page)

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Compilers, Generators, Problem Oriented Languages, Syntax Analysis, Lexical Analysis, Data Description Languages, Data Manipulation Language, Conversion Programs, Automatic Programming, I/O Utility, Automatic Program Generation						

13. continued

There have been many research activities and a large number of papers in this area. The third paper, "Automatic Generation of Software Systems - A Survey," by N. Prywes serves to relate the research underway at the University of Pennsylvania, to the many recent and current activities in this field. It also aims to clearly define short and long term objectives and methodologies.

PREFACE

Automatic Program Generation Research has been conducted under contract N00014-67-A-0216-0014, since 1971. The objective of the research has been to provide software generation directly from user specifications.

Initially, the research concentrated on a specific application, of generating file conversion programs. A first report on this subject was authored by Diane Pirog Smith, in December 1971, titled, "An Approach to Data Description and Conversion." Subsequently a software system for automating this function was implemented by Jesus A. Ramirez and described by him in a report, titled, "Automatic Generation of Data Conversion-Programs Using A Data Description Language (DDL)." Currently, the objective of the research has been broadened to develop a user language and a software system for automatic generation of business oriented programs.

This tecnnical report contains a collection of three papers intended to summarize the results of past research and the direction of current research. The first paper, by Ramirez, Rin and Prywes is a summary of Dr. Ramirez's report and dissertation cited above. The second paper, titled, "An Overview Of A System For Automatic Generation Of File Conversion Programs," by N. Adam Rin and Maxine Brown is intended to provide a more user oriented view based on their experience in utilization of the system developed by Ramirez.

There have been many research activities and a large number of papers in this area. The third paper, "Automatic Generation of Software Systems - A Survey," by N. Prywes serves to relate the research underway at the University of Pennsylvania, to the many recent and current activities in this field. It also aims to clearly define short and long term objectives and methodologies.

TABLE OF CONTENTS

TABLE OF CONTENTS

UNIVERSITY OF PENNSYLVANIA
The Moore School of Electrical Engineering
Department of Computer and Information Science

TECHNICAL REPORT

RESEARCH ON AUTOMATIC PROGRAM GENERATION

Project Supervisor
Noah S. Prywes

January 1974

Prepared for the
Office of Naval Research
Information Systems
Arlington, Va. 22217

under

Contract N00014-67-A-0216-0014
Project No. NR 049-153

Moore School Report # 74-05

"AUTOMATIC GENERATION OF DATA CONVERSION PROGRAMS USING A DATA DESCRIPTION LANGUAGE"

by:

J.A. Ramirez, N.A. Rin and N.S. Prywes

The Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, Pennsylvania 19174

ABSTRACT:

Costs and development times involved in computer software have been exceedingly large. Programming and testing constitute the major component of total software cost and account for most of the slippages that have been widely experienced. It has also been found exceedingly difficult for management to control the progress of programming and testing of software. Therefore the research program described here aims initially at reducing and controlling these components of the costs, particularly in the Business Application Programming Area.

Historically it has been necessary to employ the computer itself to reduce programming costs through use of high level programming languages, such as COBOL or PL/I, or use of Data Base Management Systems. The increasing costs indicate the need to employ even a higher level of automation than employed so far, and automate the programming in high level languages; namely automatically produce ad hoc programs in languages such as COBOL or PL/I.

The paper describes a methodology to automatically generate programs and the application of the methodology in a Processor for generating file conversion programs. The Processor, accepts as input descriptions of source and target files in a Data Description Language (DDL) and data manipulation specifications in a Data Manipulation Language (DML). It produces as an output a conversion program in PL/I capable of converting the source file and producing the target file.

1. INTRODUCTION:

The ultimate goal of the research reported in this paper is the automatic generation of programs to perform data processing. The design objectives have been as follows:

1. Input to the automatic programming system should be largely (but not completely) derived directly from the functional specifications for the program to be developed. (A number of languages for functional specifications are in use [1]).
2. Control, input/output and data definition code is to be generated automatically.
3. Data manipulation not specified in the input to the system, can be added in a modular manner without requiring concern with input/output, or control or data definition code.
4. A methodology for debugging, test and proofing of program components that is extensible to the entire end product program.

The paper describes a step toward this goal, a Processor which has been developed to generate programs for the limited application of data conversion. The Processor accepts as an input functional descriptions of source and target files in a Data Description Language (DDL). To specify validation criteria, data security, summaries and reports, the Processor accepts inputs in a Data Manipulation Language (DML). It produces as an output a conversion program in PL/1 capable of converting the source file and producing the target file. The advantage of using the Processor over programming directly in PL/1 are, briefly, the much simpler data definition in DDL, especially for variable length fields and records, and the automatic generation of JCL statements and ad-hoc PL/1 code for data declaration, input/output control logic and for calls on the DML procedures when appropriate.

The Processor generates the top levels of the program, performing the program design and producing documentation and cross referencing.

The emphasis in this paper is on the methodology used in constructing the DDL/DML Processor. The methodology is directed to implementing processors for automatic generation of programs for specific areas of data processing applications. It is also directed toward maximum ease in modification of the functional specification input language, as such languages are considered experimental until considerable experience has been gained in their use. This direction of the work, reported in the paper, is reflected in the discussion of languages used and information flow in the DDL/DML Processor in Sections 2 and 3 below, respectively.

The DDL/DML Processor automatically produces conversion programs in PL/1; i.e., PL/1 is used as an intermediate object/source language in the compilation process of DDL statements into machine code. The main reason for the use of PL/1 code is easier debugging and documenting of the compiler and the respective applications.

The Processor itself has also been programmed in PL/1, to facilitate ease of programming and to allow for better documentation. The processor is programmed for IBM 360/370 computers. It has been used in several applications to determine its computer usage, reliability and dependability.

A report on the Processor design and a User Guide to its applications and use are available as part of the Ph.D. Dissertation of the first Author [2].

The application of DDL/DML and the Processor consist of the following: (1) A Language for communication between humans about data structures: For example, for an analyst to describe a data base to an applications programmer. (2) Restructuring and conversion of files: As a utility, the Processor enables the user to convert files. (3) Interfacing files with different programs and programming languages: The Processor, can convert files into a structure which can be processed by another program. (4) Validating and editing of a file: By defining a file in DDL, one can validate it according to user-provided criteria written in DML. (5) Report generation: By defining the source file to be a data base and defining the target file to be a report, the DDL/DML Processor facilitates report generation. (6) Generally, single file to single file processing with intermediate manipulation of data.

2. SUMMARY OF LANGUAGE PROCESSING FACILITIES OF THE DDL/DML PROCESSOR

The processor has external as well as internal languages. The external user-oriented language used for input is DDL/DML. The internal languages are used to specify the syntax of DDL and some of the semantics. Based on these latter specifications various portions of the processor are automatically generated. The main aspects of both types of languages are summarized below.

DDL has extensive facilities for describing data files, covering both logical and physical structures of the data. There are facilities to describe such concepts as repeating fields, mandatory or optional fields, fixed or variable length fields. There are built-in functions to produce summaries, giving information on length of fields or the number of repeating fields. Code for editing and reformatting is generated automatically. The syntax of DDL is similar to the data descriptive facilities of COBOL, PL/1 or the DDL designed by the CODASYL Data Base Task Group [3]. The DDL used in the Processor is based on a subset of the DDL design by Smith [4]. In the course of the implementation and use it was found advantageous to make modifications to make DDL easier to implement and use. A more detailed description of the facilities of DDL accompanied by many illustrative examples of its use can be found in the User Guide Appendix of [2].

DML is used for specifying validation criteria for the data, for specifying complex conversions not built-in the Processor, for producing summaries, and for producing reports. DML is, in fact, a subset of PL/1, and therefore, has general manipulative capability. The Processor may be also regarded as an adjunct to a PL/1 Compiler, adding, in fact, the facilities of DDL to PL/1.

The syntax of DDL is easy to change. The specification of the syntax of DDL itself and its internal encoding is used as an input to a portion of the Processor, from which the Syntactic Analysis Program (SAP) for DDL is generated. The language used to specify the syntax of DDL is an Extended BNF (EBNF).

To convey some of the semantics, the specifying language was extended further by allowing the specific calling for subroutines. The Extended BNF With Subroutine Calls is referred to in the following as EBNF/WSC. It was used for specifying processing and encoding of the DDL Statements into internal tables, and thus also to reduce the work in programming the code generation.

The specification of DDL is used as an input to a program called Syntactic Analysis Program Generator (SAPG). This program then generates SAP, which is used to syntactically analyze and encode the input DDL/DML statements. This aspect of the processor was considered most important, as it was realized that the DDL being used, as well as other functional specification languages, are experimental and extensive changes may be necessary before even a near optimal language is obtained. This facility was indeed used to change the language. For instance, a change from the DDL designed by Smith [4] to a modified DDL was accomplished in less than a week. Another advantage was the ease in incorporating input error analysis, as experience was gained with use of the DDL Processor. It was also found that automatic generation of the SAP, via SAPG, greatly speeded and reduced the work required to implement the processor.

Transition Matrices were used in Lexical Analysis of the DDL as well as the EBNF/WSC statements following the ideas in [5], and [6].

It is, however, noteworthy that techniques for automatic generation of the Processor's code generation routines were not used. This was because the DDL processor is specially designed to generate conversion programs and the research to date [7] on automatic generation of code for general purpose language compilers is not applicable. Therefore, the code generation of the DDL/DML processor was hand coded in PL/1. Thereby, better efficiency was also achieved in the generation of the conversion programs. All the hand codes, as well as automatically generated parts of the system, including the SAPG, were written in PL/1 for reasons of facilitating programming, maintainability and limited machine independence.

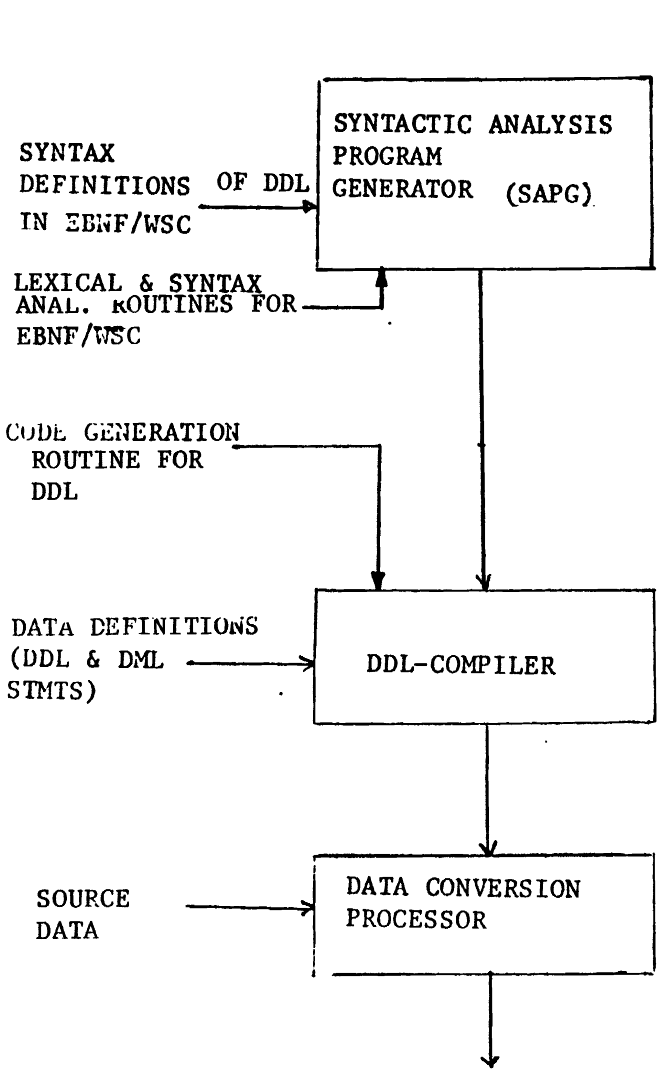
3. INFORMATION FLOW IN THE DDL/DML PROCESSOR

3.1 INTRODUCTION

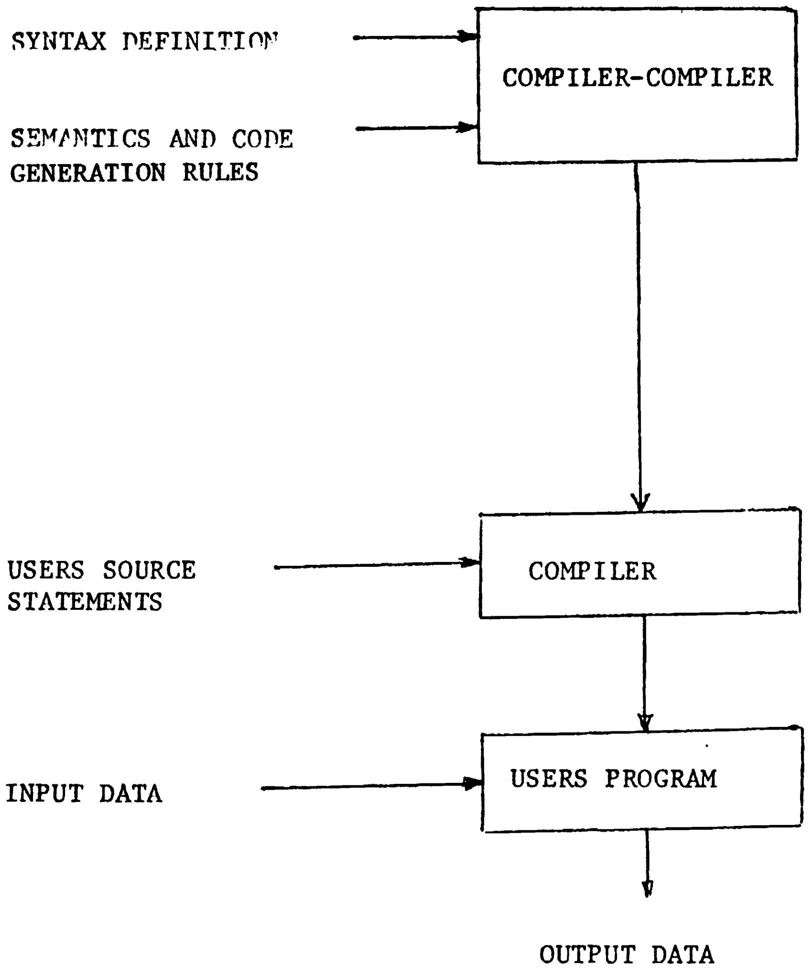
The information flow in the DDL/DML Processor is illustrated in Figures 1 through 6. Figures 1 and 2 give a progressively somewhat more detailed overview of the entire processor. The remaining four Figures, give a further detailed information on the three major components, the Syntax Analysis Program Generator, the DDL Compiler and the generated Conversion Program, respectively.

3.2 OVERVIEW OF THE DDL/DML PROCESSOR

The DDL/DML Processor is actually a set of three processors. These are shown in Figure 1a as follows: The first is the Syntactic Analysis Program Generator (SAPG). It accepts as an input the EBNF/WSC syntax specifications of the DDL and produces the Syntax Analysis Program (SAP) for the DDL. This product is then joined with inputs consisting of routines providing additional semantics, code generation logic, etc. to make the second processor, the DDL Compiler. The second processor accepts the DDL/DML statements and its product is the third processor,


SYNTAX DEFINITIONS OF DDL IN EBNF/WSC
SYNTACTIC ANALYSIS PROGRAM GENERATOR (SAPG)
LEXICAL & SYNTAX ANAL. ROUTINES FOR EBNF/WSC
CODE GENERATION ROUTINE FOR DDL
DATA DEFINITIONS (DDL & DML STMTS)
DDL-COMPILER
SOURCE DATA
DATA CONVERSION PROCESSOR
TARGET DATA

1a THE DDL/DML PROCESSOR



1b COMPILER-COMPILER SYSTEM

FIGURE 1
COMPARISON OF DDL/DML PROCESSOR AND COMPILER-COMPILER
SYSTEMS DESIGNS

the Data Conversion Program, which is used to convert the source data into the target data.

An analogy with existing computer programming language systems is illustrated by comparing Figure 1a with Figure 1b. The SAPG (in 1a) is analogous to a compiler-compiler's syntax analysis generation module (in 1b) which is used to produce syntax analysis programs for the specific language. The DDL Compiler (in 1a) is analogous to a programming language processor, (in 1b) such as COBOL, FORTRAN or PL/1 compilers. The Data Conversion Processor (in 1a) is analogous to an executeable user program (in 1b). The relationship between the use of the three processors in the DDL system is readily seen from the analogy.

The SAPG is a program used to create the syntax analysis program which in turn, becomes part of the DDL Compiler. The "generator" is a very valuable tool in the development of the DDL Compiler. Furthermore, it could be a "stand-alone" valuable tool in writing any language syntax analysis program.

The DDL Compiler and the Data Conversion Processor are the only components of the DDL Processor that most users will need. To produce a Data Conversion Processor, a user writes a data definition, a series of statements in DDL, for each of the source and target files. These statements are basically descriptions of the formats of the source and target files and of transformations from the former to the latter. If data manipulation is needed, a series of routines are written in DML. These statements are used as inputs to the DDL Compiler. The compiler in turn produces a data conversion program. Just as it is not necessary to compile a conventional program each time it is used, it is not necessary, also, to create a new data conversion program, each time it is used. The same object module could be re-used (either as PL/1 source or in object code form).

A somewhat more detailed view of the three major components of the DDL Processor is given in Figure 2. In Figure 2-6, the three processors are identified by being surrounded by broken lines. A rectangle with the missing top represents an input, and with a missing bottom an output. Outputs which are programs, are shown in trapezoids. When the output is a program, double lines are used to show where it is later used. Figure 2 shows an overview of the information flow with some additional detail. In particular, syntax and lexical analysis programs makes up the DDL Compiler.

The discussion in the following three sub-sections (and in Figures 3 through 6) provides a more detailed view of the components, in the order of top to bottom of Figure 2.

3.3 THE SYNTAX ANALYSIS PROGRAM GENERATOR (SAPG)

The top left of Figure 2 is expanded in Figure 3. The SAPG is a compiler in its own. It consists of Lexical Analysis (a) Syntax Analysis (b) and Code Generation (c) modules, all hand coded in PL/1. They are compiled into IBM/370 machine code using the PL/1 compiler, and form the SAPG.

The input to SAPG is the DDL syntax specification, in EBNF/WSC (d). The output (f) is a Syntax Analysis Program (SAP) - in PL/1, which

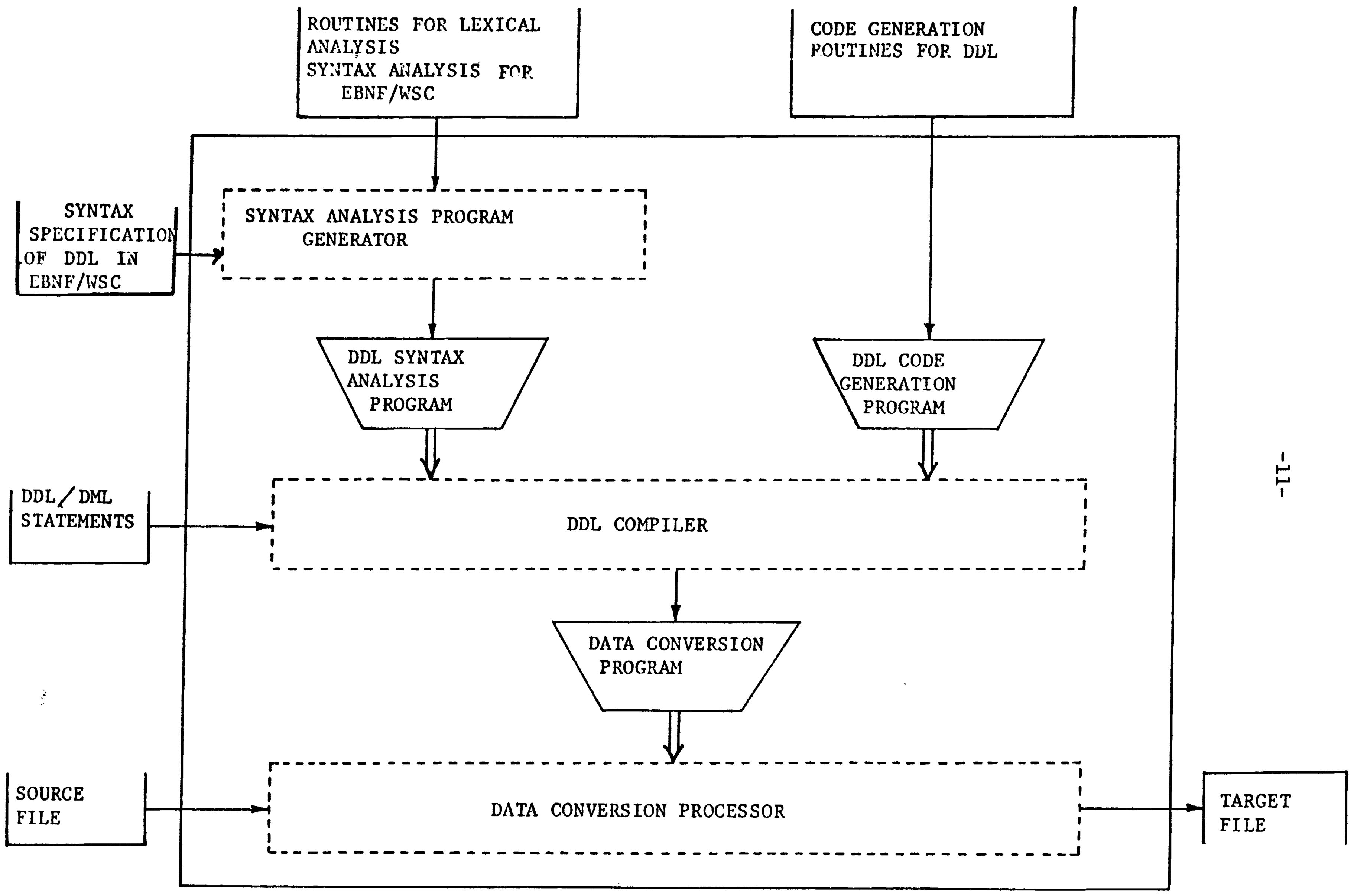

ROUTINES FOR LEXICAL ANALYSIS SYNTAX ANALYSIS FOR EBNF/WSC
CODE GENERATION ROUTINES FOR DDL
SYNTAX SPECIFICATION OF DDL IN EBNF/WSC
SYNTAX ANALYSIS PROGRAM GENERATOR
DDL SYNTAX ANALYSIS PROGRAM
DDL CODE GENERATION PROGRAM
DDL/DML STATEMENTS
DDL COMPILER
DATA CONVERSION PROGRAM
SOURCE FILE
DATA CONVERSION PROCESSOR
TARGET FILE

FIGURE 2
MAJOR COMPONENTS OF THE
DDL-PROCESSOR

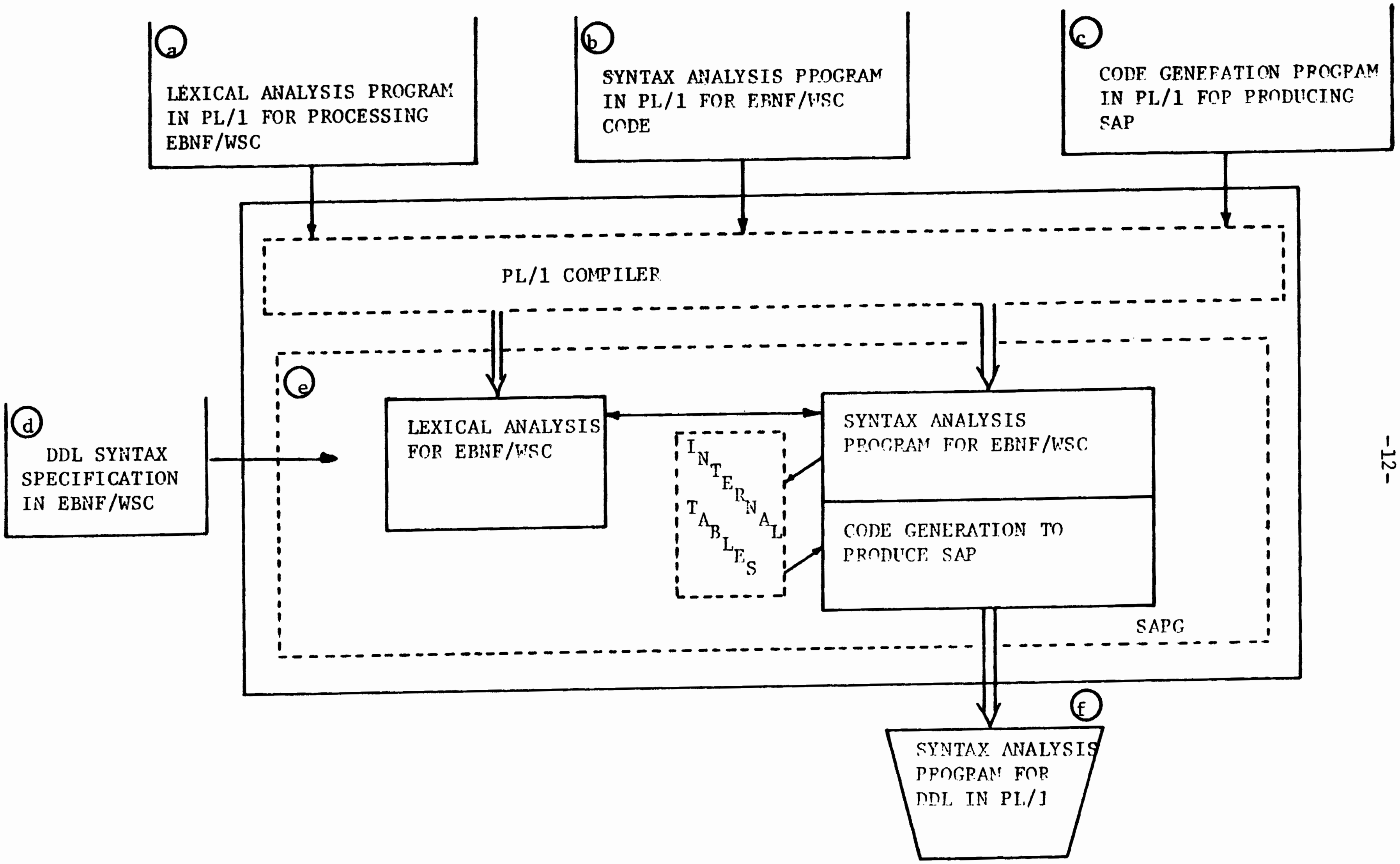

a
LEXICAL ANALYSIS PROGRAM IN PL/1 FOR PROCESSING EBNF/WSC
b
SYNTAX ANALYSIS PROGRAM IN PL/1 FOR EBNF/WSC CODE
c
CODE GENERATION PROGRAM IN PL/1 FOR PRODUCING SAP
PL/1 COMPILER
d
DDL SYNTAX SPECIFICATION IN EBNF/WSC
e
LEXICAL ANALYSIS FOR EBNF/WSC
INTERNAL TABLES
SYNTAX ANALYSIS PROGRAM FOR EBNF/WSC
CODE GENERATION TO PRODUCE SAP
SAPG
f
SYNTAX ANALYSIS PROGRAM FOR DDL IN PL/1

FIGURE 3
SYNTAX ANALYSIS PROGRAM GENERATOR
(SAPG)

can perform the syntax analysis on statements written in DDL.

The SAPG has all the components of a compiler. The lexical analysis consists of processing the EBNF/WSC statements to form tokens. The syntax analysis further processes the EBNF/WSC strings of tokens to perform various checks, generate error comments and organize the information in accordance with the internal tables.

Finally this information is analyzed and the SAP PL/1 code is generated for processing subsequently (in Figure 4) the DDL Statements.

3.4 THE DDL COMPILER

Figure 4 shows the components that are used to make up the DDL compiler. They are as follows:

(1) The Lexical Analysis Subroutine for DDL - hand coded in PL/1.

(2) The Syntax Analysis Program (SAP) - produced by the SAPG in PL/1.

(3) Supporting subroutines - written in PL/1, perform services required by the SAP. Services include recognition of syntactic elements, diagnostics of syntax errors, creation of Internal Tables (symbol and data tables) and, if specified, a cross-reference table of the identifiers used in the DDL program. These subroutines can be called in the EBNF/WSC statements.

(4) Code Generation Program - written in PL/1, forming the basis of the code generation phase of the DDL compiler. In it is contained the logic required to interpret the information stored in the internal tables and to generate PL/1 statements which will perform the data conversion indicated by the analysis of DDL statements.

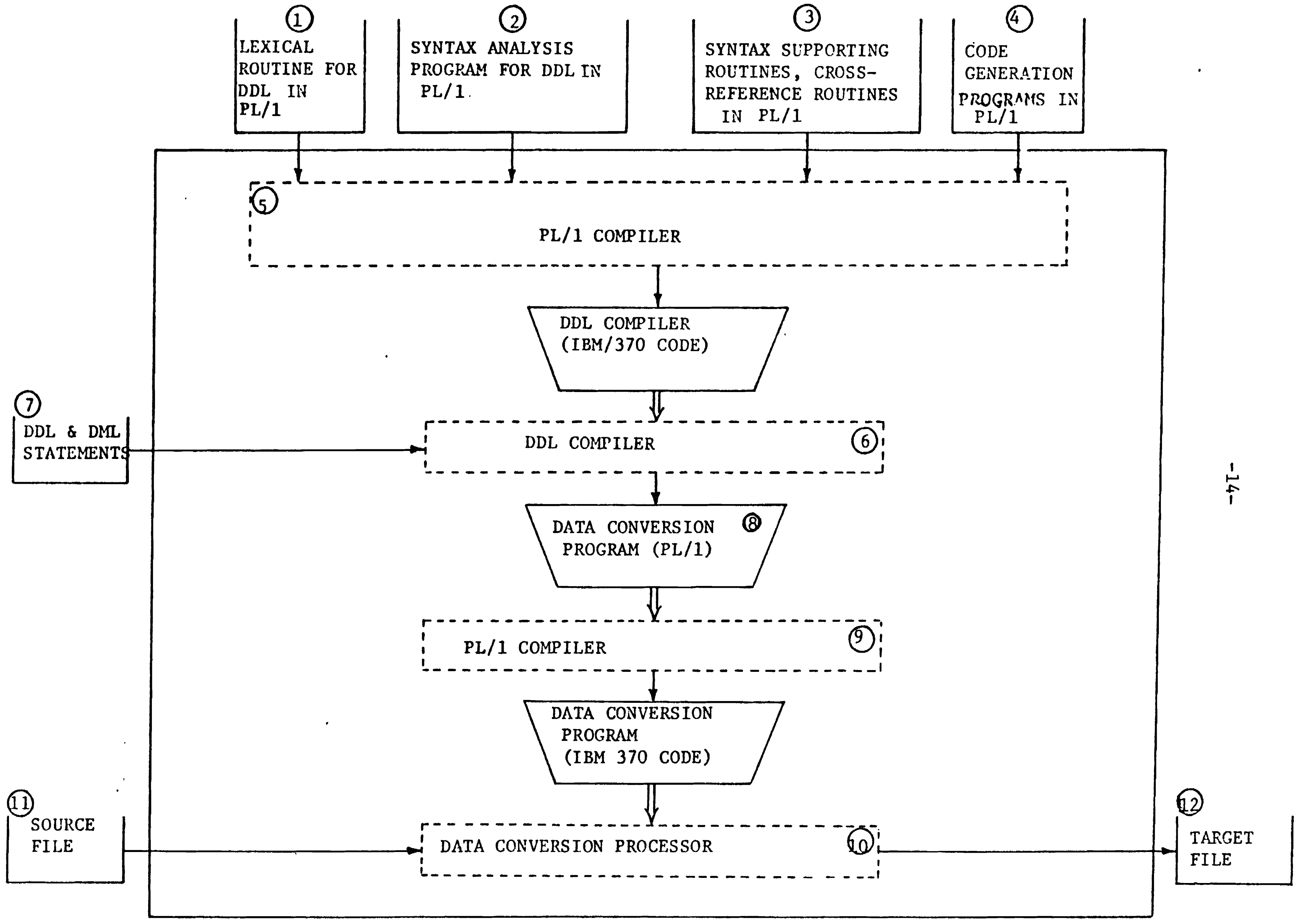

1
LEXICAL ROUTINE FOR DDL IN PL/1
2
SYNTAX ANALYSIS PROGRAM FOR DDL IN PL/1
3
SYNTAX SUPPORTING ROUTINES, CROSS-REFERENCE ROUTINES IN PL/1
4
CODE GENERATION PROGRAMS IN PL/1
5
PL/1 COMPILER
DDL COMPILER (IBM/370 CODE)
7
DDL & DML STATEMENTS
DDL COMPILER
6
DATA CONVERSION PROGRAM (PL/1)
8
PL/1 COMPILER
9
DATA CONVERSION PROGRAM (IBM 370 CODE)
11
SOURCE FILE
DATA CONVERSION PROCESSOR
10
12
TARGET FILE

FIGURE 4
DDL COMPILER INPUT AND OUTPUT COMPONENTS

(5) & (6) The PL/1 Compiler (5) produces the DDL Compiler (6) in IBM/370 machine code.

The remainder of the system is also summarized in Figure 4, as follows:

(7) The DDL statements given by the user describe the structure of his source and target files and mappings from the former to the latter. The DML statements describe the file manipulations to be performed on the source file, such as editing, complex specialized conversion formulas, security testing, report generation and statistical gathering of data.

(8) & (9) The Data Conversion Program, in PL/1 is the input to the PL/1 compiler (9) this produces the Data Conversion Program (10) in machine code for the IBM/370.

(10), (11) and (12) Finally the Data Conversion Program (10) accepts as input the source file (11) and outputs the target file (12).

The compiling process is further expanded in Figure 5. It is performed in two phases, as described below.

PHASE 1 OF THE DDL COMPILER

In phase 1, DDL statements are read by the Lexical Analysis Program (LEX) that forms tokens which are in turn the input to the SAP. The SAP examines the string of tokens to determine whether or not the string obeys certain structural conventions explicit in the syntactic definition of the language. Should an error be discovered, the error-diagnostic routines will be called to output a message informing the user of the location and nature of the misconstruction.

Concurrent with this error detection is the internal table generation. At this time, routines are called whose functions are the capturing of

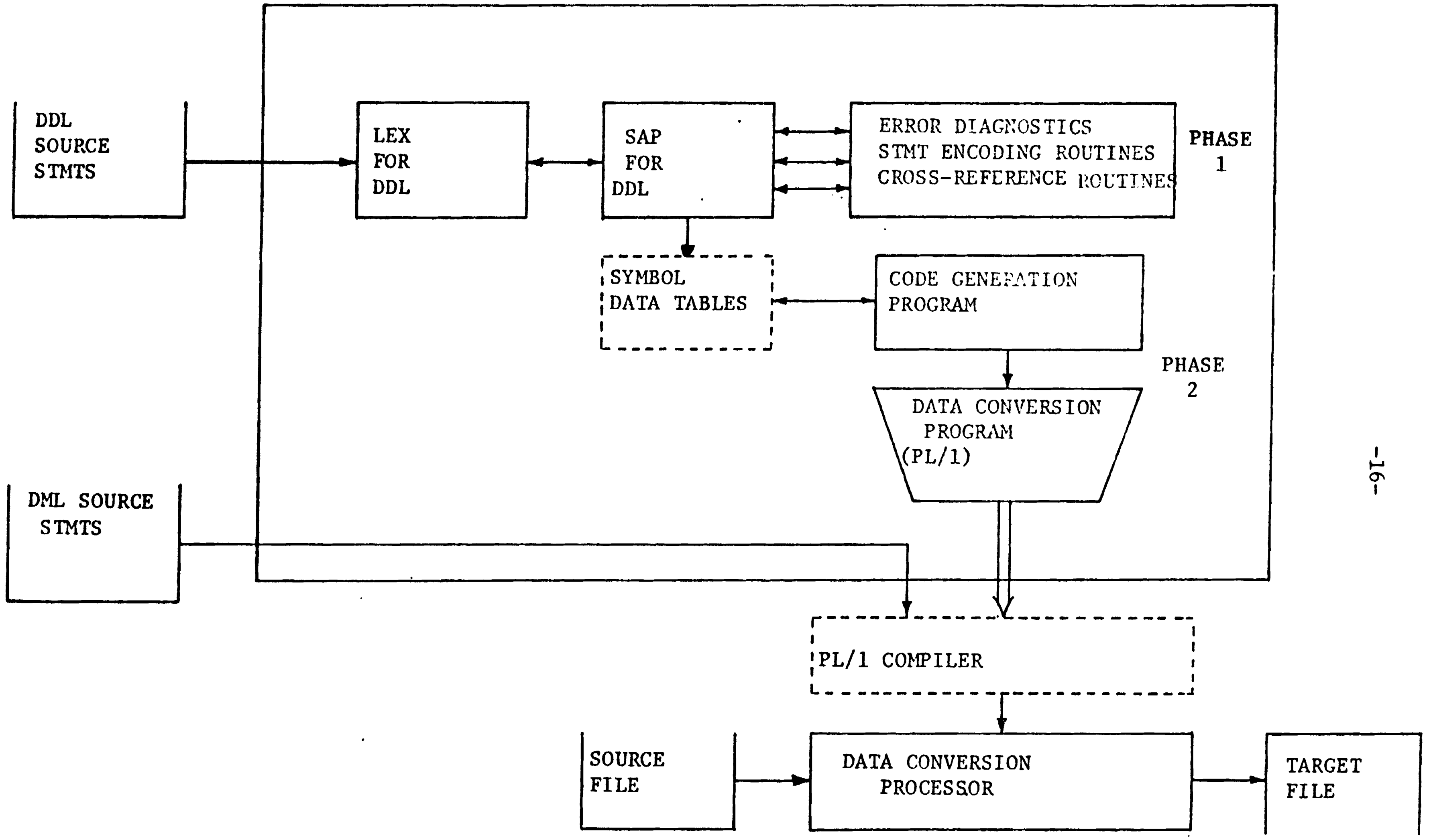


FIGURE 5
DDL COMPILER PHASES

information contained in the DDL source statement and the building of tables to preserve this data in coded form for use during code generation, as well as in the detection of global syntax errors. The internal tables that are formed are the Symbol and Data Tables. If no errors were detected and if a "XREF" option was specified to the DDL Compiler, then the cross-reference table is generated and output for the user's reference.

PHASE 2 OF THE DDL COMPILER

Phase 2 is code generation. The first part of Phase 2 is the execution of a series of programs which steps through the Data Table and generates PL/1 declare statements. When compiled, these will produce a Description Table that contains the following for each field of the source record: (1) data descriptor information, and (2) space for placing pointers to the field at execution time.

After the compiler has produced the Descriptor Table, a call is made to the Data Parsing Code generation routines. These routines generate the code which will set the field pointers in the Descriptor Table entry to point to the fields in the input buffer of the source file at run time.

The second part of Phase 2 is a program which uses the Data Table entries for the target file and generates the data movement code, which moves the source data into the target file. This completes Phase 2 of the DDL compiler. At this point the DML statements are read by the DDL compiler and they are merged with the code produced during the code generation phase.

The creation of the data conversion program in IBM/370 machine language, is performed by the PL/1 compiler. The input to the PL/1 compiler is the PL/1 text produced in Phase 2.

3.5 DATA CONVERSION PROCESSOR

The Data Conversion Processor is composed of the set of programs and data which was produced by the DDL compiler and the DML routines supplied by the user. This is illustrated in Figure 6, it is composed of:

(a) a Data Conversion Program

(b) a Data Structure for the source file, and

(c) a set of DML subroutines

The Data Structure for the Source File, which was produced by the DDL Compiler from the DDL statements, is used by the generated Data Conversion Program to aid in parsing the source data. The other component of the Data Conversion Processor, the user-supplied DML subroutines, perform functions such as character set conversion, extension or truncation of fields, data type conversion, criteria testing, security testing, report generators and gathering of statistical data.

The information flow in the Data Conversion Processor, shown in Figure 6, in fact is a representation of a generalized strategy for data conversion. In terms of a program this strategy is represented by two parts. The first consists of the data definition and control code which is required in a program to allocate buffers, input/output of blocks and records, identify the location of data elements in accordance with a specified data structure and finally perform standard conversion functions. The "flow chart" for the generated code is illustrated in Fig. 7. In a "top-down" [8] concept of a program, the first part constitutes the "top." The second part, which consists of data manipulation routines in DML constitutes the "bottom" level of a program. This general strategy has

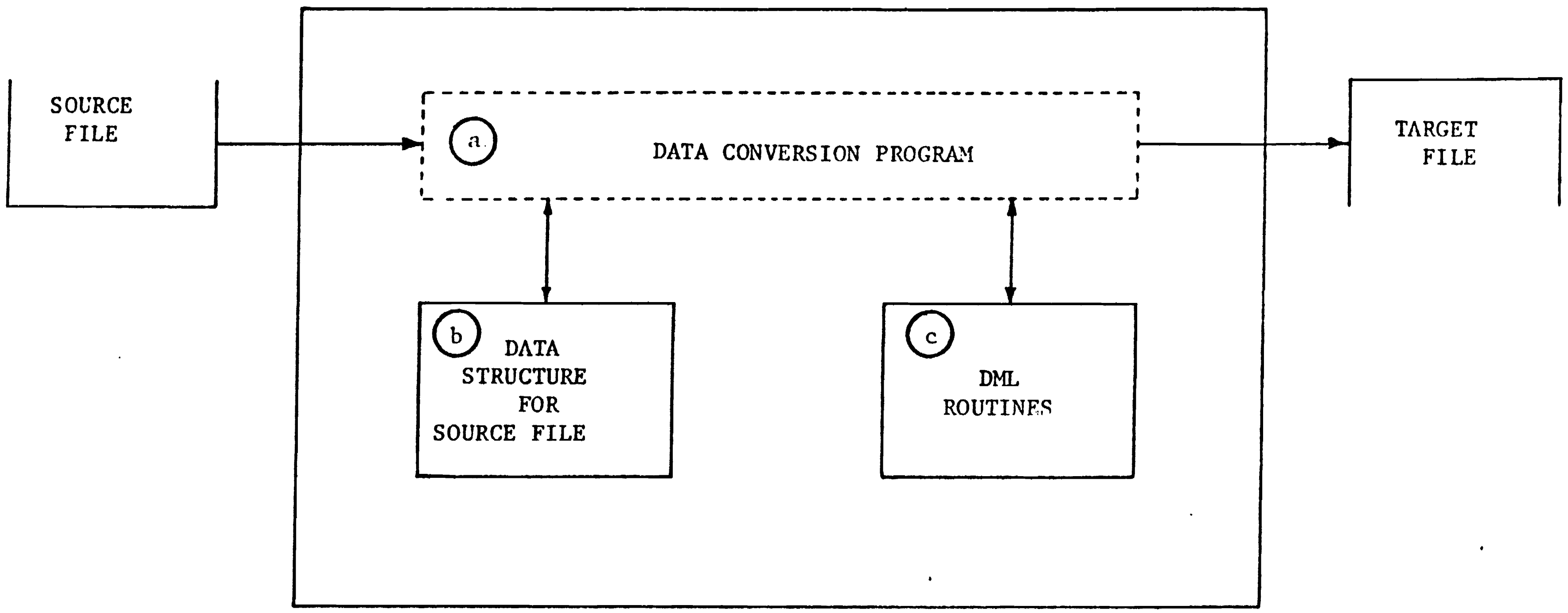


FIGURE 6

DATA CONVERSION PROCESSOR

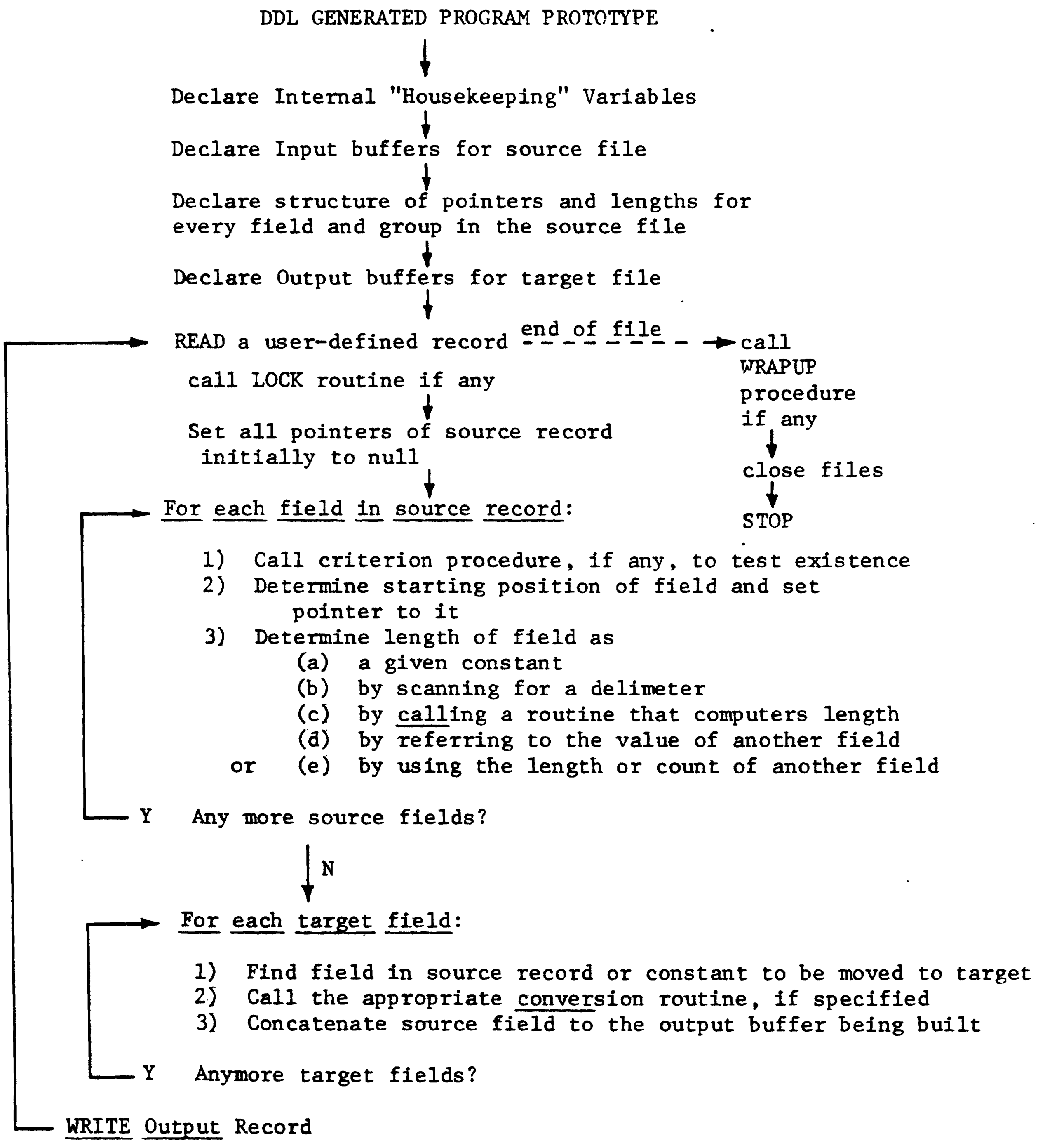

DDL GENERATED PROGRAM PROTOTYPE
Declare Internal "Housekeeping" Variables
Declare Input buffers for source file
Declare structure of pointers and lengths for every field and group in the source file
Declare Output buffers for target file
READ a user-defined record
end of file
call WRAPUP procedure if any
close files
STOP
call LOCK routine if any
Set all pointers of source record initially to null
For each field in source record:
1) Call criterion procedure, if any, to test existence
2) Determine starting position of field and set pointer to it
3) Determine length of field as
(a) a given constant
(b) by scanning for a delimeter
(c) by calling a routine that computers length
(d) by referring to the value of another field
or (e) by using the length or count of another field
Y
Any more source fields?
N
For each target field:
1) Find field in source record or constant to be moved to target
2) Call the appropriate conversion routine, if specified
3) Concatenate source field to the output buffer being built
Y
Anymore target fields?
WRITE Output Record

FIG. 3 ILLUSTRATION OF THE "FLOW CHART" OF DDL/DML PROCESSOR GENERATED CONVERSION PROGRAM

been adopted from the beginning in the supporting and code generation routines. It is this type of strategy that is envisaged to change from one class of potential applications to another.

4. CONCLUSION

The Version 1.0 DDL/DML Processor has been found to be an effective and efficient tool, by the users, in several applications, although it has limited functions and capabilities. Additions of functions and use to gain additional experience are progressing. The authors invite those interested to communicate with them regarding receipt of the system for wider experimentation. Presently, the subjective impressions are that effectiveness in program development, efficiency in processing and reliability are gratifying. However, there have not been a systematic approach to measuring these aspects so far.

The authors however feel that in this study of development, an equally important outcome is the availability of the system as a research tool and backbone for expansion to accept other functional specification languages and to perform for wider classes of data processing applications.

ACKNOWLEDGEMENT

The authors acknowledge with thanks the participation in design and programming by Messrs. H. Solow, P. Gross and A. French, and the preparation of the documentation and this manuscript by Miss B. Weber.

REFERENCES:

[1] Teichrow, D.: Survey Of Languages For Stating Requirements For Computer Based Information Systems, Proc. of the Fall Joint Computer Conference, 1972, pp. 1203-1244.

[2] Ramirez, J.A.: Automatic Generation Of Data Conversion-Programs Using A Data Description Language (DDL), Ph.D. Dissertation, University of Pennsylvania, 1973.

[3] Conference On Data Systems Languages. Data Base Task Group. Report to the CODASYL Programming Language Committee. Association for Computing Machinery

Conference On Data Systems Languages. Data Base Task Group. Report to the CODASYL Programming Language Committee. Association for Computing Machinery, New York, New York, April 1971.

Conference On Data Systems Languages, Systems Committee. Feature Analysis of Generalized Data Base Management Systems, Association for Computing Machinery, New York, New York, May 1971. 511p.

[4] Smith, D.P.: An Approach To Data Description and Conversion, Ph.D. Dissertation, University of Pennsylvania, 1971.

[5] Floyd, R.W.: The Syntax Of A Programming Language - A Survey, IEEE Trans., 1964.

[6] Conway, M.E.: Design Of A Separable Transition - Diagram Compiler. Comm. ACM., Vol. 6, No. 7, 1963.

[7] Fang, Isu.: A Declarative Formal Language Definition System, Ph.D. Dissertation, Stanford University, 1972.

[8] H.D. Mills, Top Down Programming In Large Systems, Debugging Techniques In Large Systems, R. Rustin, Editor, Prentice Hall, 1971, pp. 41-55.

AN OVERVIEW OF A SYSTEM FOR AUTOMATIC GENERATION OF FILE CONVERSION PROGRAMS

N. Adam Rin and Maxine Brown

ABSTRACT

This paper describes a processor which automatically produces file conversion programs based on non-procedural user specifications. The Processor accepts, as input, descriptions of a source file and a desired target file with some auxilliary descriptions of associations between the two. This is specified by a user in a Data Description Language (DDL). To specify validation criteria, complex, conversions not built-in to the system, security criteria, or summary processes, the system also accepts specifications in a Data Manipulation Language (DML). It produces, as an output, a conversion program in PL/1 capable of converting the described source file into the desired target file. The paper describes the structure, system design, capabilities, and applications of the DDL/DML language and processor, including an illustrative example.

AN OVERVIEW OF A SYSTEM FOR AUTOMATIC GENERATION OF DATA CONVERSION PROGRAMS

N. Adam Rin and Maxine Brown

I. AN OVERVIEW OF THE DDL PROCESSOR SYSTEM

A. Introduction

This paper is concerned with research done in the field of automatic generation of data conversion programs; it provides an overview of an existing software system which produces file conversion programs from a user's specification in a data definition language and it discusses possible future work in this area. The system was designed and implemented with support from the Office of Naval Research by a group of graduate students under the supervision of Dr. N.S. Prywes at the Moore School of Electrical Engineering at the University of Pennsylvania.[1]

The system developed is a step towards the automatic generation of data processing programs. The class of data processing programs which the existing processor can automatically generate is file conversion programs; i.e., the system is able to produce a procedural program based on non-procedural user oriented specifications to translate an existing file of known structure into a new desired format. Non-procedural specifications as used here are descriptive statements defining the source and target files and <u>what</u> transformations are to be performed between the two, but do not relate <u>how</u> these transformations will be constructed. The procedural program produced by the processor is

1 For detailed documentation of the system, its implementation and use, see J.A. Ramirez, "Automatic Generation of Data Conversion Programs Using a Data Description Language (DDL)," Ph.D. Dissertation, Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, Pennsylvania, 19174; 1973.

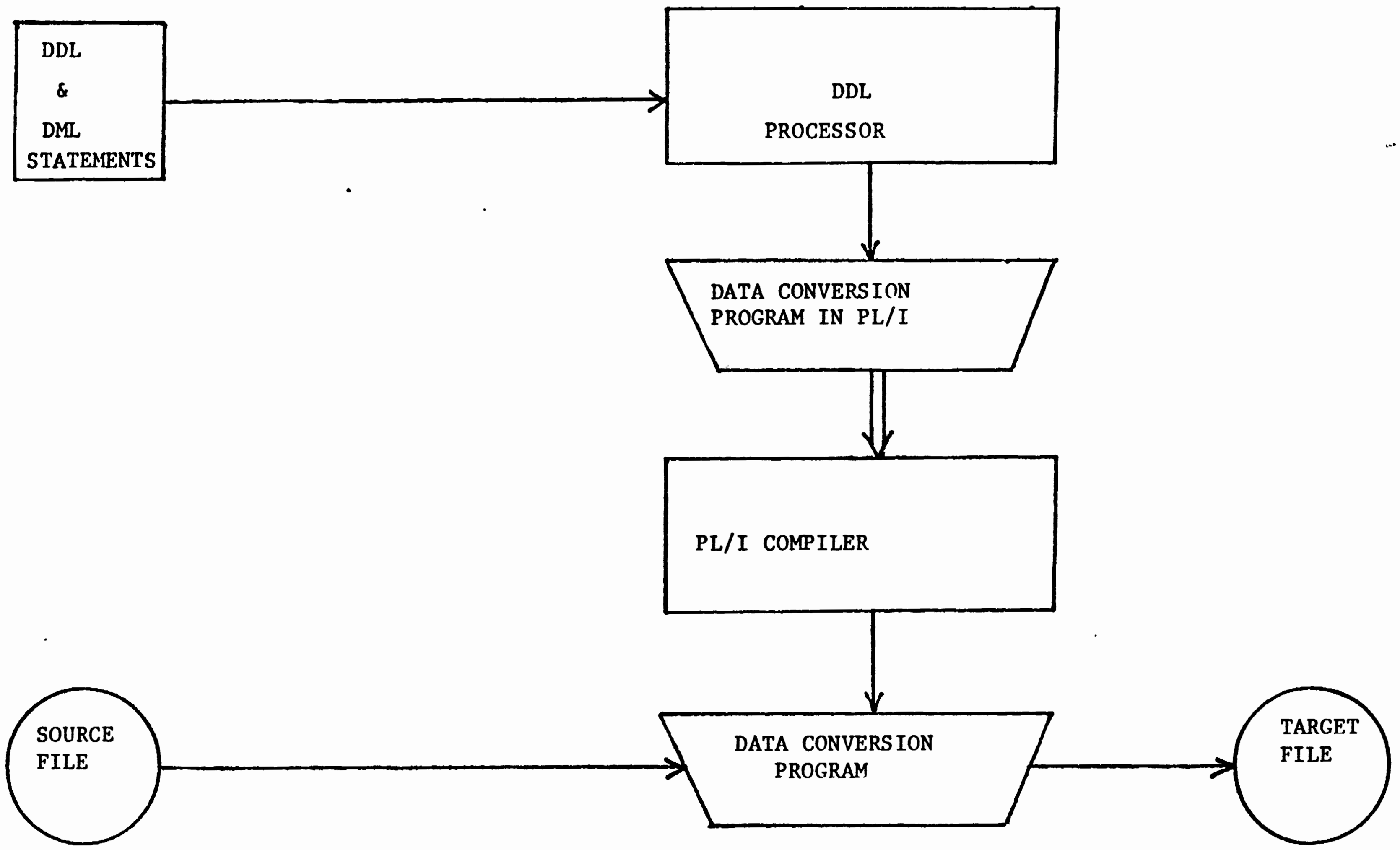


FIGURE 1 THE DDL PROCESSOR BLOCK DIAGRAM

a PL/1 conversion program complete with input/output related instructions, logical sequencing and control, and invocation of special manipulative routines peculiar to the problem.

Figure 1 shows a block diagram of the processor which accepts, as input, descriptions of a source file and a desired target file with some auxilliary associations between the two specified in a Data Description Language (DDL). To specify validation criteria, complex conversions not built into the system, security criteria, and summary processes, the system also accepts as input prescriptive statements in a Data Manipulation Language (DML), a subset of PL/1 with general manipulative capabilities. The processor produces, as output, a procedural PL/1 program capable of converting the source file into the desired target file. The generated program is then compiled into machine language, yielding a tailored-to-need conversion program. The processor can therefore be regarded as an adjunct and enhancement to the PL/1 compiler, adding the facilities of DDL to it.

B. The DDL Language as a Data Description Language

The DDL that has been developed contains all the data definition facilities of COBOL and PL/1 plus other unique capabilities for describing data structures and organization not found in these languages. For example, the DDL language and its processor have the ability to compute the length of a data item at execution time and to accept a variable number of occurrences of variable length data items.

The actual syntax of the DDL developed is not extremely crucial to pinpoint; the way the processor was implemented makes it easy to change the DDL syntax. Namely, the specification of the syntax of DDL is itself an

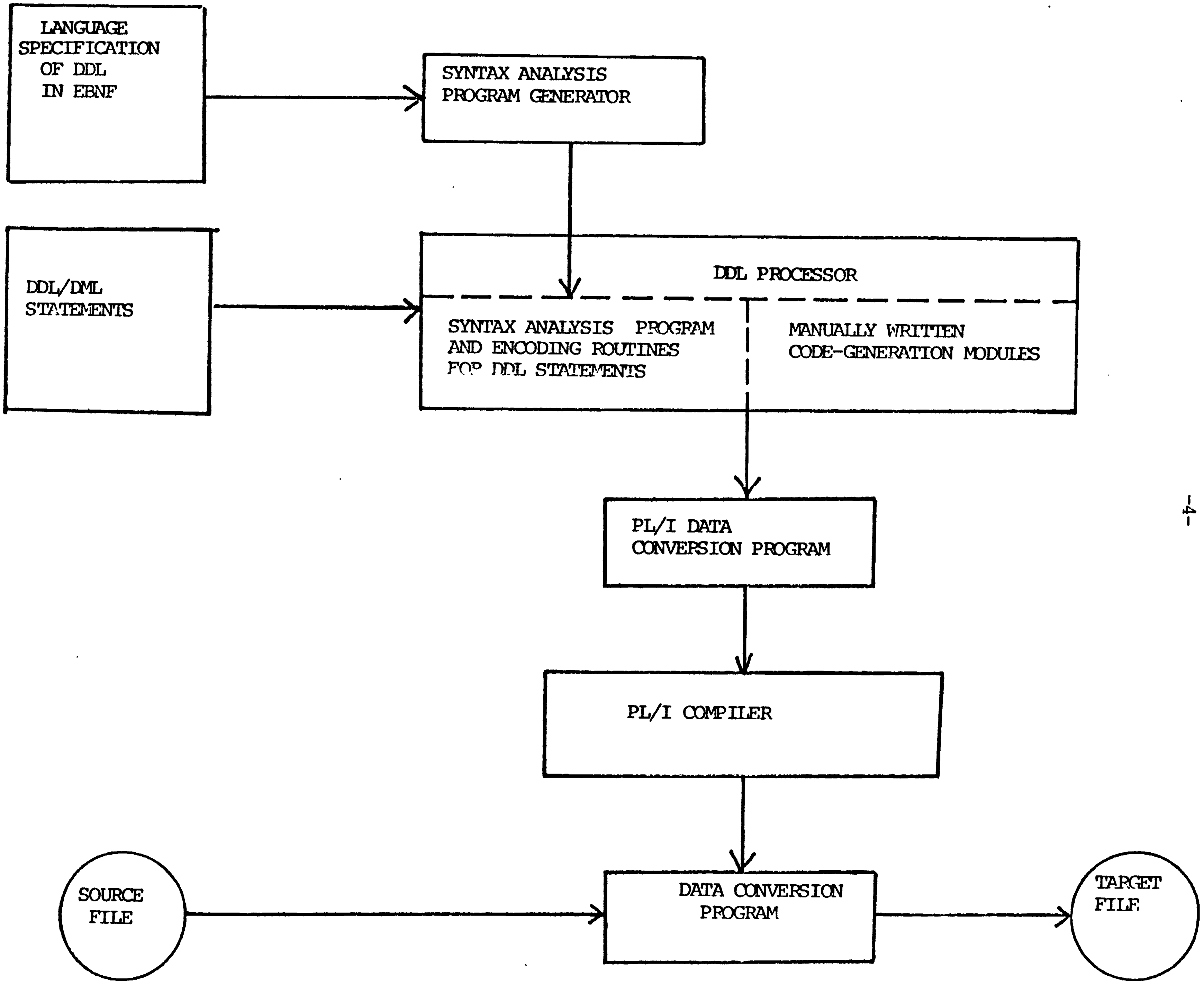

LANGUAGE SPECIFICATION OF DDL IN EBNF
SYNTAX ANALYSIS PROGRAM GENERATOR
DDL/DML STATEMENTS
DDL PROCESSOR
SYNTAX ANALYSIS PROGRAM AND ENCODING ROUTINES FOR DDL STATEMENTS
MANUALLY WRITTEN CODE-GENERATION MODULES
PL/I DATA CONVERSION PROGRAM
PL/I COMPILER
SOURCE FILE
DATA CONVERSION PROGRAM
TARGET FILE

FIGURE 2 BLOCK DIAGRAM OF SAPG AND THE DDL PROCESSOR SYSTEM

input to a meta-processor, developed by this project, which generates a syntax analysis program for DDL (or any other language specified to it). As seen in the more refined block diagram of the DDL processor in Figure 2, this meta-processor, called a Syntax Analysis Program Generator (SAPG), accepts descriptions of a formal language such as DDL in a meta-language which is essentially an extension of BNF. It generates a Syntax Analysis Program (SAP) for that language (which, in our case, is DDL) and invokes manually-written processing routines. The SAP for DDL together with manually written code generation modules form the bulk of the DDL processor which, in turn, accepts the DDL/DML statements and generates a PL/1 program.

The implementation of the SAPG and its use to implement the DDL System allows changes to the syntax of DDL to be made relatively easily and in a short period of time. Thus, this processor could accomodate other Data Description Languages with relatively little effort. This is a crucial fact, for such languages are considered experimental until experience is gained on their use and effectiveness.

C. Capabilities and Applications of DDL and Its Processor

Actually, any application involving one input sequential file and one output sequential file is a candidate for use in the present DDL System. Some examples of the uses of the DDL language and its processor are the following:

(1) A Language for Communication Between Humans About Data Structure

One important application of DDL is as a means of communication between humans about the structure of data. For example, using a DDL, a designer of a data base can describe precisely to an application programmer the exact structure of the data the programmer is to use. Furthermore, even non-technical personnel could learn a DDL-like language to describe data.

(2) Restructuring and Conversion of Files

As a utility, the DDL processor in conjunction with DML enables the user to re-define the structure of his file, reformat it and convert it accordingly. Furthermore, conversion of a file can be done selectively; i.e., those portions of a file that meet a user's criteria can be selectively converted or copied to a new file.

(3) Interfacing Files with Different Programs and Programming Languages

Frequently files created by one program cannot be processed by another program or by another program in a different programming language or computer facility. With the DDL processor, files can be converted into a structure or format which can be processed by another program or machine. In this manner files can be interfaced across programming languages and computing facilities.

(4) Validation of a File

By defining a file in DDL, one can validate it according to user - provided criteria written in DML.

II. USAGE AND CAPABILITIES OF THE DDL PROCESSOR WITH AN ILLUSTRATIVE EXAMPLE

A. Usage and function of the DDL processor

Figure 2A shows a block diagram of the usage and function of the DDL processor. A file conversion problem using the DDL system passes through two states; the user is responsible for formulating and describing the data definitions, mappings, and manipulations in the DDL/DML language, and the DDL processor compiles and executes these descriptions to produce the desired file. Enumerating this process, the user is responsible for producing the following:

(1) a DDL description of the source file,

(2) a DDL description of the target file with the associated mappings from the source file, and

(3) the DML routines for specifying criteria, special computations, and complex conversions not built-in to the system.

The DDL processor, in turn, will generate the following programs and listings:

(1) the desired generated PL/1 program that performs the conversions (this program declares appropriate variables, allocates and opens buffers for the source and target files, issues input commands for reading the source file, issues output commands for writing the target file,

and invokes the user provided DML routines at the appropriate locations);

(2) a listing of the user's DDL/DML statements;

(3) diagnostics for user errors;

(4) a cross-reference table which has an alphabetical listing of all the names which appear in the user's DDL description along with its attribute (field, group, record, etc.) and the line numbers of all the statements in which it appears;

(5) a listing of the required Job Control Language (JCL) cards which describes the input and output files and are needed to run the generated program under the OS 360/370 operating system.

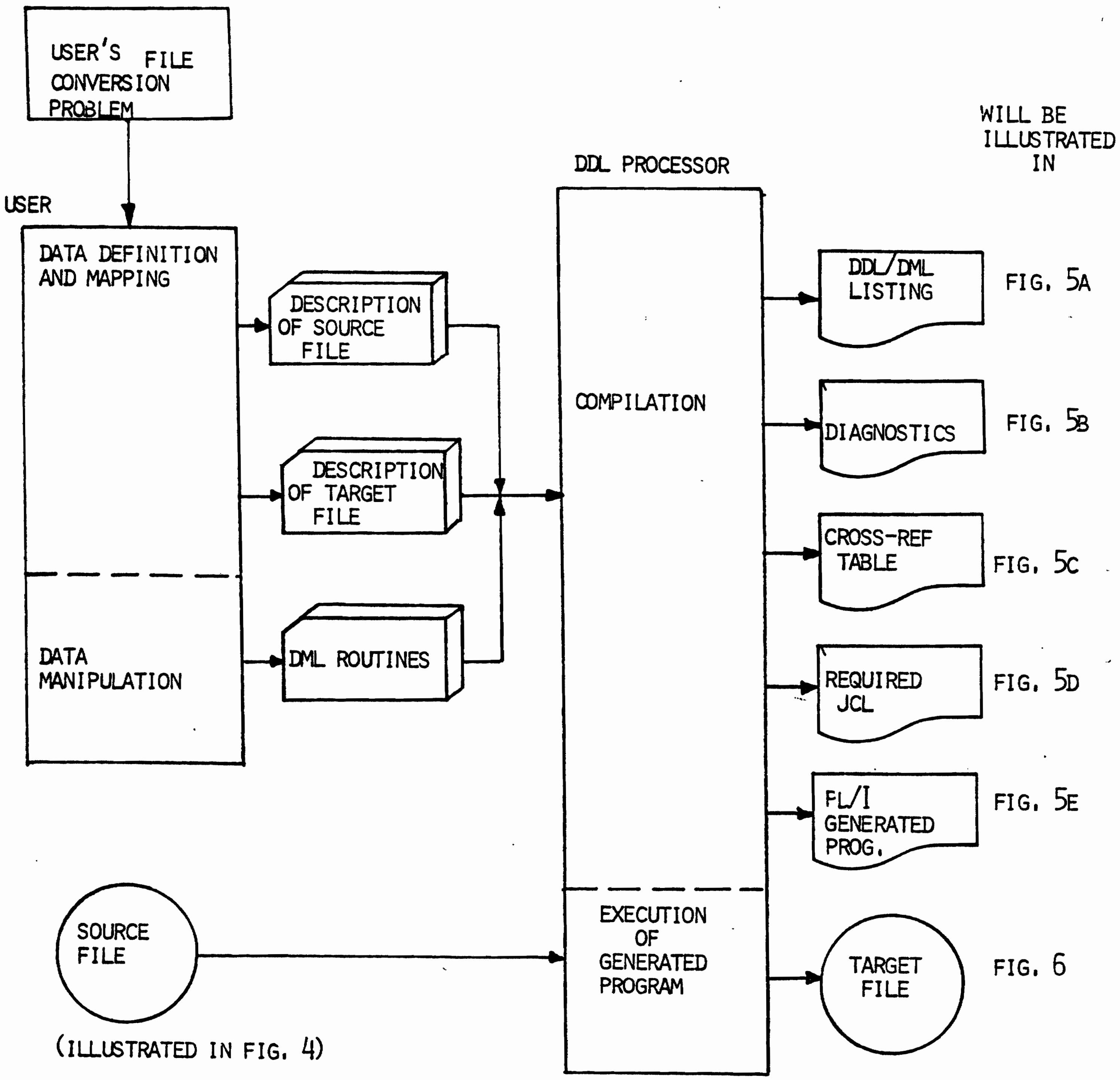


FIGURE 2A: USAGE AND FUNCTION OF THE DDL LANGUAGE AND PROCESSOR

The generated program, integrated with the user-provided DML routines, is input into the PL/1 compiler, compiled into machine language, and executed to produce the desired target file.

The above listings will be illustrated later in Figures 5a - 5e.

B. Statement of Problem

In order to present a brief discourse on how to use the DDL system, a sample problem was devised, implemented, and reproduced in this section for illustrative purposes.

It is desired to convert a given tape consisting of text of news articles into one whose records are of a more legible format. The format of the source records have the hierarchical structure as shown in Figure 3a.

The NUM field contains a 5 digit number which is the number of bytes contained in the field IN_TEXT.

DOCUMENT is a 6-digit field which contains a serial number assigned to the news article found in IN_TEXT. If the article is more than 1 record in length, the same document number appears on all related records.

DATE is a 8-character field containing the date the news event took place.

TIME is a 5-character field containing the time of the event.

TITLE is a variable-length field whose length is to be computed by a user-provided routine called TXXTLEN.

IN_TEXT is a field which contains the actual news article. It has a variable length of up to 4088 bytes and contains either the entire article or a portion thereof. In the latter case, the article is continued in the next record, but in no case is there ever more than one article per record.

The desired program is to perform the following conversions:

(1) the beginning of each article is prefixed by a collection of fields,

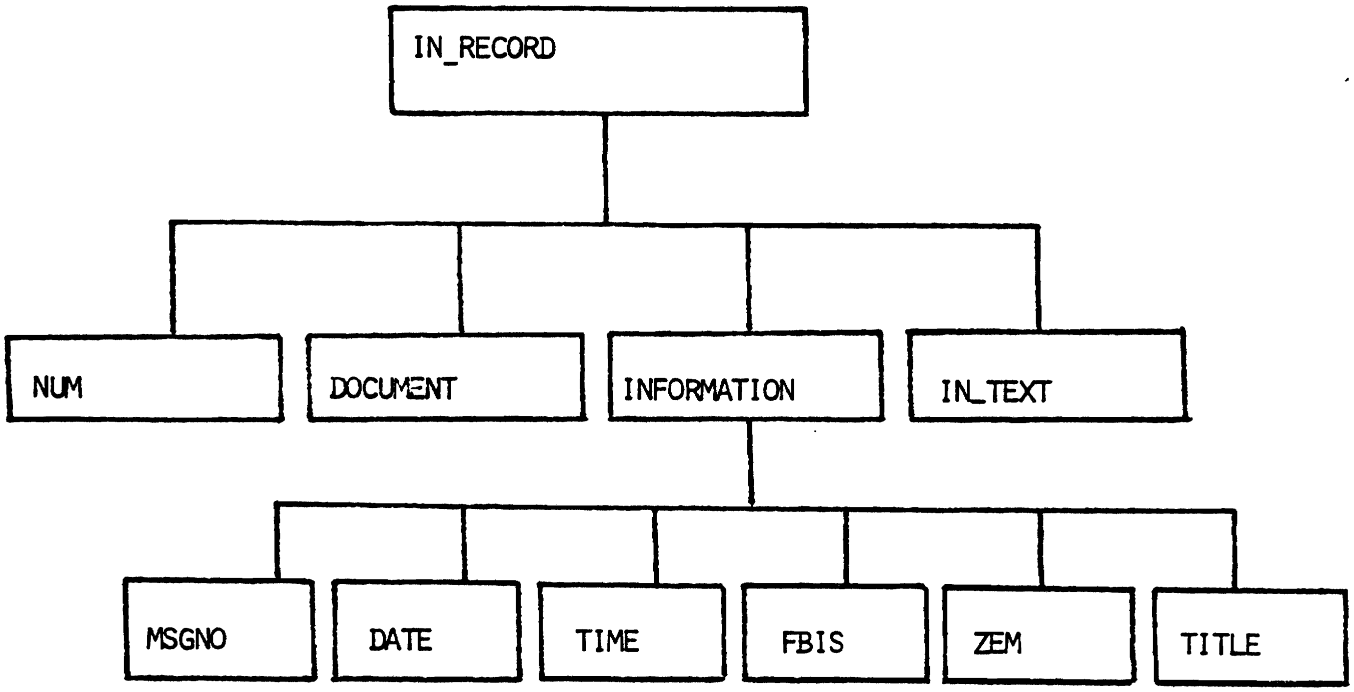


FIGURE 3A SOURCE RECORD STRUCTURE

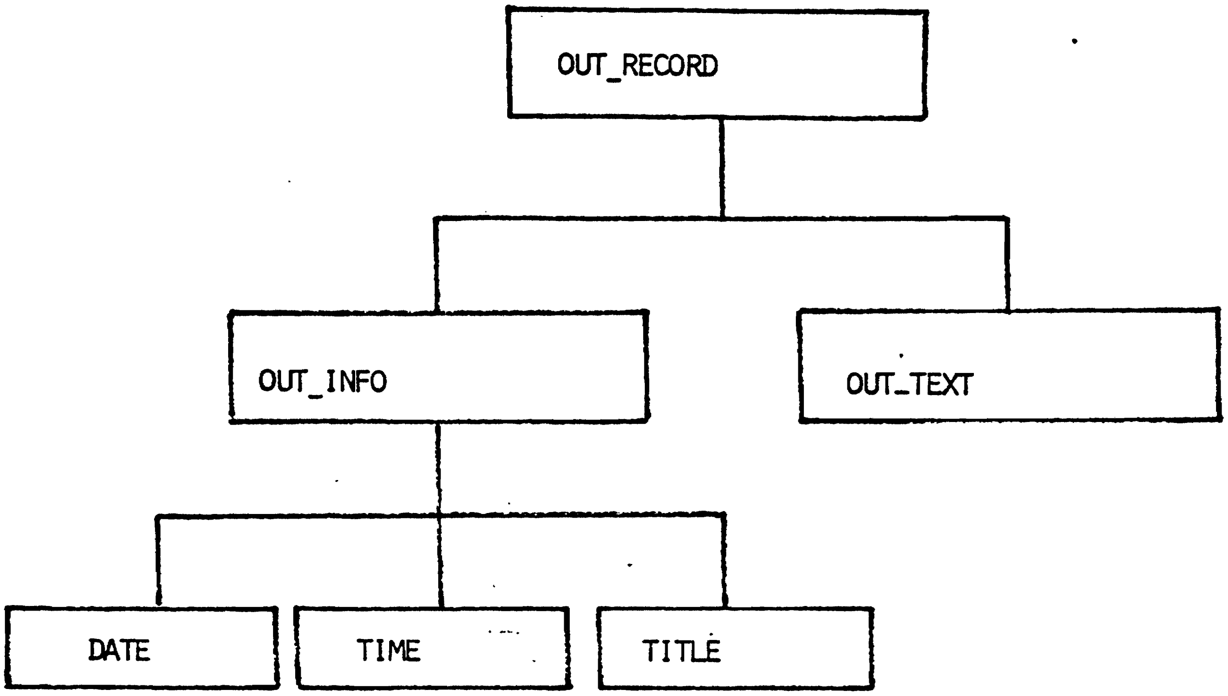


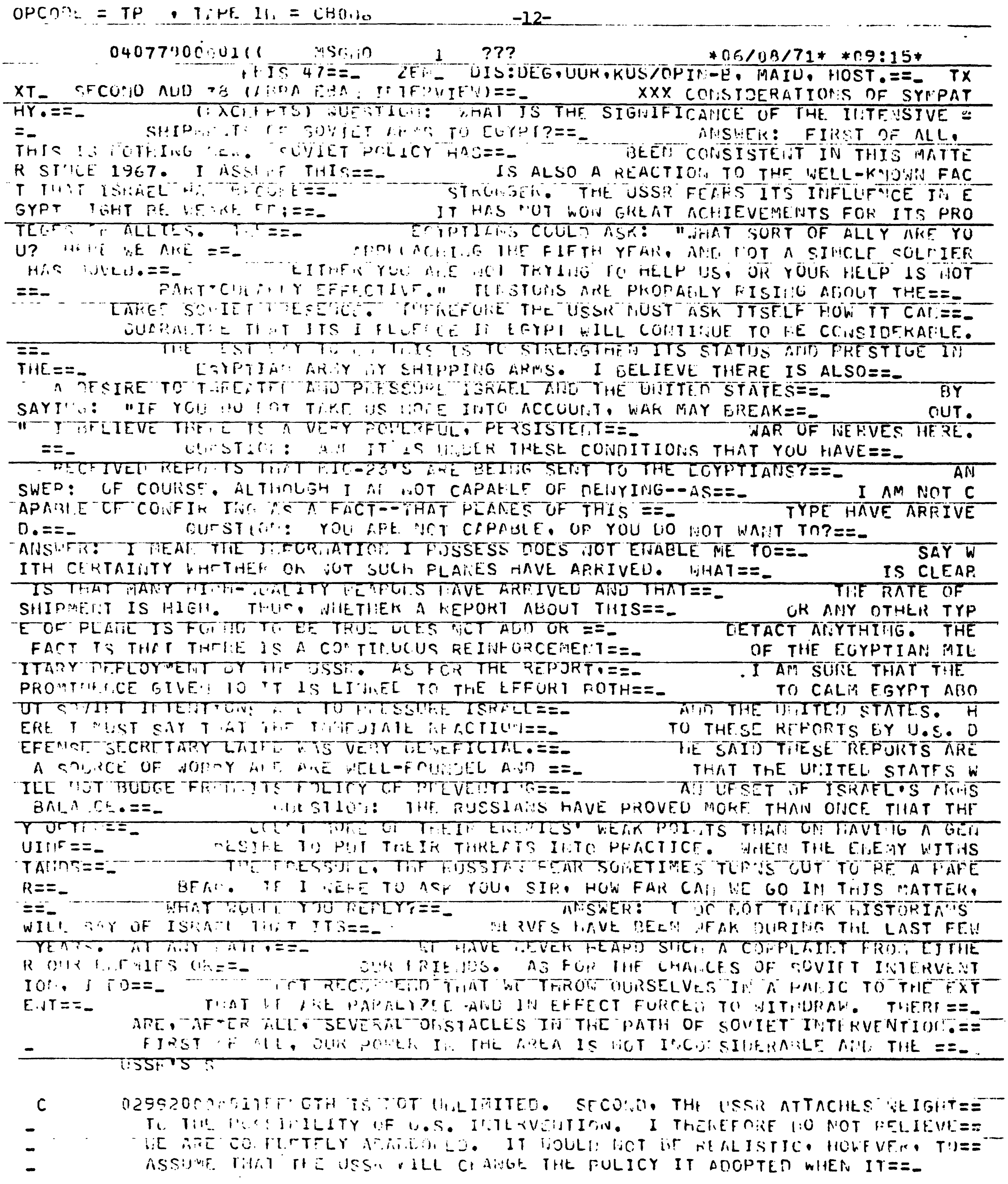
FIGURE 3B TARGET RECORD STRUCTURE

denoted as INFORMATION, from which we wish to extract DATE, TIME, and TITLE.

(2) The body of the news article contains special sequences of characters, which shall be referred to as break characters, interspersed throughout the news article. These are to be scanned for and eliminated.

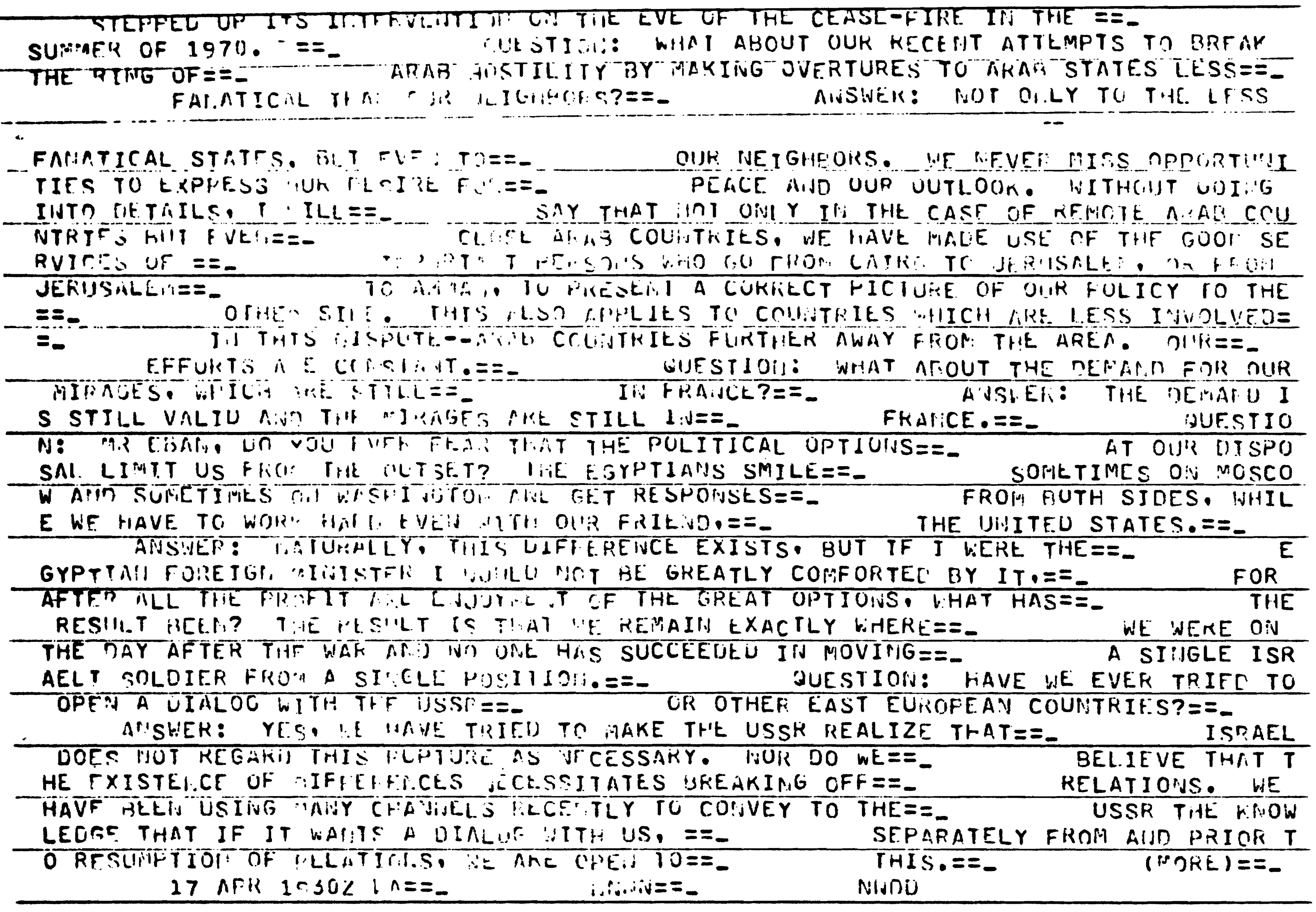
(3) The desired target tape is to contain records with fields DATE, TIME, and TITLE, if apropos, and an OUT_TEXT field, as depicted in Figure 3b, which consists of the preserved text without the special break characters and introduction. The target textual records are to output with a maximum of 75 characters per line for the purpose of legibility when the tape is printed at a later time. For efficiency considerations, however, the 75-byte logical records are blocked to 7500 characters on the output tape.

Figure 4 shows the dump of the first few records of the original tape, while Figures 5a through 5e show the listing produced by the DDL compilation for this sample problem. A dump of the first few records of the desired target tape is given in Figure 6.



```
OPCODE = TP  , TAPE ID = CB006

       0407790000101((     MSGNO       1     ???                *06/08/71* *09:15*
                  PRIS 47==_   ZEM_  DIS:DEG,UUR,KUS/OPIN-B, MAID, HOST.==_   TX
XT_  SECOND ADD 78 (ARPA EBA; INTERVIEW)==_            XXX CONSIDERATIONS OF SYMPAT
HY.==_             (EXCERPTS) QUESTION:  WHAT IS THE SIGNIFICANCE OF THE INTENSIVE =
=_          SHIPMENTS OF SOVIET ARMS TO EGYPT?==_             ANSWER:  FIRST OF ALL,
THIS IS NOTHING NEW.  SOVIET POLICY HAS==_                BEEN CONSISTENT IN THIS MATTE
R SINCE 1967.  I ASSUME THIS==_                IS ALSO A REACTION TO THE WELL-KNOWN FAC
T THAT ISRAEL HAS BECOME==_                STRONGER.  THE USSR FEARS ITS INFLUENCE IN E
GYPT  IGHT BE WEAKENED;==_                 IT HAS NOT WON GREAT ACHIEVEMENTS FOR ITS PRO
TEGES OR ALLIES.  THE==_               EGYPTIANS COULD ASK:  "WHAT SORT OF ALLY ARE YO
U?  HERE WE ARE ==_              APPROACHING THE FIFTH YEAR, AND NOT A SINGLE SOLDIER
 HAS MOVED.==_              EITHER YOU ARE NOT TRYING TO HELP US, OR YOUR HELP IS NOT
==_          PARTICULARLY EFFECTIVE."  TENSIONS ARE PROBABLY RISING ABOUT THE==_
        LARGE SOVIET PRESENCE.  THEREFORE THE USSR MUST ASK ITSELF HOW IT CAN==_
         GUARANTEE THAT ITS INFLUENCE IN EGYPT WILL CONTINUE TO BE CONSIDERABLE.
==_          THE BEST WAY TO DO THIS IS TO STRENGTHEN ITS STATUS AND PRESTIGE IN
THE==_           EGYPTIAN ARMY BY SHIPPING ARMS.  I BELIEVE THERE IS ALSO==_
  A DESIRE TO THREATEN AND PRESSURE ISRAEL AND THE UNITED STATES==_           BY
SAYING:  "IF YOU DO NOT TAKE US MORE INTO ACCOUNT, WAR MAY BREAK==_          OUT.
"  I BELIEVE THERE IS A VERY POWERFUL, PERSISTENT==_              WAR OF NERVES HERE.
  ==_            QUESTION:  AND IT IS UNDER THESE CONDITIONS THAT YOU HAVE==_
 RECEIVED REPORTS THAT MIG-23'S ARE BEING SENT TO THE EGYPTIANS?==_               AN
SWER:  OF COURSE, ALTHOUGH I AM NOT CAPABLE OF DENYING--AS==_             I AM NOT C
APABLE OF CONFIRMING AS A FACT--THAT PLANES OF THIS ==_             TYPE HAVE ARRIVE
D.==_            QUESTION:  YOU ARE NOT CAPABLE, OR YOU DO NOT WANT TO?==_
ANSWER:  I MEAN THE INFORMATION I POSSESS DOES NOT ENABLE ME TO==_            SAY W
ITH CERTAINTY WHETHER OR NOT SUCH PLANES HAVE ARRIVED.  WHAT==_            IS CLEAR
 IS THAT MANY HIGH-QUALITY WEAPONS HAVE ARRIVED AND THAT==_            THE RATE OF
SHIPMENT IS HIGH.  THUS, WHETHER A REPORT ABOUT THIS==_            OR ANY OTHER TYP
E OF PLANE IS FOUND TO BE TRUE DOES NOT ADD OR ==_           DETACT ANYTHING.  THE
 FACT IS THAT THERE IS A CONTINUOUS REINFORCEMENT==_            OF THE EGYPTIAN MIL
ITARY DEPLOYMENT BY THE USSR.  AS FOR THE REPORT,==_          .I AM SURE THAT THE
PROMINENCE GIVEN TO IT IS LINKED TO THE EFFORT BOTH==_            TO CALM EGYPT ABO
UT SOVIET INTENTIONS AND TO PRESSURE ISRAEL==_              AND THE UNITED STATES.  H
ERE I MUST SAY THAT THE IMMEDIATE REACTION==_            TO THESE REPORTS BY U.S. D
EFENSE SECRETARY LAIRD WAS VERY BENEFICIAL.==_             HE SAID THESE REPORTS ARE
 A SOURCE OF WORRY AND ARE WELL-FOUNDED AND ==_            THAT THE UNITED STATES W
ILL NOT BUDGE FROM ITS POLICY OF PREVENTING==_              AN UPSET OF ISRAEL'S ARMS
 BALANCE.==_           QUESTION:  THE RUSSIANS HAVE PROVED MORE THAN ONCE THAT THE
Y OFTEN==_            COUNT MORE ON THEIR ENEMIES' WEAK POINTS THAN ON HAVING A GEN
UINE==_           DESIRE TO PUT THEIR THREATS INTO PRACTICE.  WHEN THE ENEMY WITHS
TANDS==_             THE PRESSURE, THE RUSSIAN FEAR SOMETIMES TURNS OUT TO BE A PAPE
R==_          BEAR.  IF I WERE TO ASK YOU, SIR, HOW FAR CAN WE GO IN THIS MATTER,
==_          WHAT WOULD YOU REPLY?==_             ANSWER:  I DO NOT THINK HISTORIANS
WILL SAY OF ISRAEL THAT ITS==_              NERVES HAVE BEEN WEAK DURING THE LAST FEW
 YEARS.  AT ANY RATE,==_              WE HAVE NEVER HEARD SUCH A COMPLAINT FROM EITHE
R OUR ENEMIES OR==_              OUR FRIENDS.  AS FOR THE CHANCES OF SOVIET INTERVENT
ION, I DO==_              NOT RECOMMEND THAT WE THROW OURSELVES IN A PANIC TO THE EXT
ENT==_            THAT WE ARE PARALYZED AND IN EFFECT FORCED TO WITHDRAW.  THERE==_
        ARE, AFTER ALL, SEVERAL OBSTACLES IN THE PATH OF SOVIET INTERVENTION.==
_         FIRST OF ALL, OUR POWER IN THE AREA IS NOT INCONSIDERABLE AND THE ==_
       USSR'S S

 C       0299200005111RENGTH IS NOT UNLIMITED.  SECOND, THE USSR ATTACHES WEIGHT==
_         TO THE POSSIBILITY OF U.S. INTERVENTION.  I THEREFORE DO NOT BELIEVE==
_         WE ARE COMPLETELY ABANDONED.  IT WOULD NOT BE REALISTIC, HOWEVER, TO==
_         ASSUME THAT THE USSR WILL CHANGE THE POLICY IT ADOPTED WHEN IT==_
```

FIGURE 4: SOURCE FILE LIST



```
  STEPPED UP ITS INTERVENTION ON THE EVE OF THE CEASE-FIRE IN THE ==_
SUMMER OF 1970. ==_          QUESTION:  WHAT ABOUT OUR RECENT ATTEMPTS TO BREAK
THE RING OF==_           ARAB HOSTILITY BY MAKING OVERTURES TO ARAB STATES LESS==_
       FANATICAL THAN OUR NEIGHBORS?==_          ANSWER:  NOT ONLY TO THE LESS

FANATICAL STATES, BUT EVEN TO==_          OUR NEIGHBORS.  WE NEVER MISS OPPORTUNI
TIES TO EXPRESS OUR DESIRE FOR==_          PEACE AND OUR OUTLOOK.  WITHOUT GOING
INTO DETAILS, I WILL==_          SAY THAT NOT ONLY IN THE CASE OF REMOTE ARAB COU
NTRIES BUT EVEN==_         CLOSE ARAB COUNTRIES, WE HAVE MADE USE OF THE GOOD SE
RVICES OF ==_          IMPORTANT PERSONS WHO GO FROM CAIRO TO JERUSALEM, OR FROM
JERUSALEM==_          TO AMMAN, TO PRESENT A CORRECT PICTURE OF OUR POLICY TO THE
==_          OTHER SIDE.  THIS ALSO APPLIES TO COUNTRIES WHICH ARE LESS INVOLVED=
=_          IN THIS DISPUTE--ARAB COUNTRIES FURTHER AWAY FROM THE AREA.  OUR==_
      EFFORTS ARE CONSTANT.==_          QUESTION:  WHAT ABOUT THE DEMAND FOR OUR
 MIRAGES, WHICH ARE STILL==_          IN FRANCE?==_          ANSWER:  THE DEMAND I
S STILL VALID AND THE MIRAGES ARE STILL IN==_          FRANCE.==_          QUESTIO
N:  MR EBAN, DO YOU EVER FEAR THAT THE POLITICAL OPTIONS==_          AT OUR DISPO
SAL LIMIT US FROM THE OUTSET?  THE EGYPTIANS SMILE==_          SOMETIMES ON MOSCO
W AND SOMETIMES ON WASHINGTON AND GET RESPONSES==_          FROM BOTH SIDES, WHIL
E WE HAVE TO WORK HARD EVEN WITH OUR FRIEND,==_          THE UNITED STATES.==_
      ANSWER:  NATURALLY, THIS DIFFERENCE EXISTS, BUT IF I WERE THE==_          E
GYPTIAN FOREIGN MINISTER I WOULD NOT BE GREATLY COMFORTED BY IT.==_          FOR
AFTER ALL THE PROFIT AND ENJOYMENT OF THE GREAT OPTIONS, WHAT HAS==_          THE
 RESULT BEEN?  THE RESULT IS THAT WE REMAIN EXACTLY WHERE==_          WE WERE ON
THE DAY AFTER THE WAR AND NO ONE HAS SUCCEEDED IN MOVING==_          A SINGLE ISR
AELI SOLDIER FROM A SINGLE POSITION.==_          QUESTION:  HAVE WE EVER TRIED TO
 OPEN A DIALOG WITH THE USSR==_          OR OTHER EAST EUROPEAN COUNTRIES?==_
     ANSWER:  YES, WE HAVE TRIED TO MAKE THE USSR REALIZE THAT==_          ISRAEL
 DOES NOT REGARD THIS RUPTURE AS NECESSARY.  NOR DO WE==_          BELIEVE THAT T
HE EXISTENCE OF DIFFERENCES NECESSITATES BREAKING OFF==_          RELATIONS.  WE
HAVE BEEN USING MANY CHANNELS RECENTLY TO CONVEY TO THE==_          USSR THE KNOW
LEDGE THAT IF IT WANTS A DIALOG WITH US, ==_          SEPARATELY FROM AND PRIOR T
O RESUMPTION OF RELATIONS, WE ARE OPEN TO==_          THIS.==_          (MORE)==_
       17 APR 1930Z LA==_          [illegible]==_          NNDD
```

FIGURE 4: CONTINUED

```
CONVERT(SFILE INTO TFILE);
/***********************************************************************/
/*                                                                     */
/*                     DESCRIPTION OF SOURCE FILE                      */
/*                                                                     */
/***********************************************************************/
  SFILE IS FILE(IN_RECORD ,STORAGE=MAG_TAPE);
   IN_RECORD IS RECORD(NUM            /* THE LENGTH OF 'IN_TEXT' */
                       , DOCUMENT  /* SERIAL NO. ASSIGNED TO ARTICLE *
                       , INFORMATION(0:1) ,PRE_CRIT='INTRO'
                                   /* INTRODUCTION TO TEXT */
                       , IN_TEXT   /* ACTUAL TEXT */
                       ,SIZE=VARIABLE(4089));
      NUM IS FIELD(NUM_PICTURE='99999');
      DOCUMENT IS FIELD(CHAR(6));
      INFORMATION IS GROUP ( JUNK01
                           , MSGNO
                           , JUNK02
                           , DATE
                           , JUNK03
                           , TIME
                           , JUNK04
                           , FBIS
                           , ZEM
                           , TITLE);
         JUNK01 IS FIELD (CHAR(6));
         MSGNO IS FIELD (CHAR(11));
         JUNK02 IS FIELD (CHAR(23));
         DATE IS FIELD (CHAR(8));
         JUNK03 IS FIELD (CHAR(3));
         TIME IS FIELD (CHAR(5));
         JUNK04 IS FIELD(CHAR(24));
         FBIS IS FIELD (CHAR('FBISLEN'));
         ZEM IS FIELD (CHAR('ZEMLEN'));
         TITLE IS FIELD (CHAR('TXXTLEN'));
       IN_TEXT IS FIELD(CHAR('PROC3'));
     MAG_TAPE IS TAPE(VARIABLE(4096) ,VOL_NAME=X00279);
/***********************************************************************/
/*                                                                     */
/*                     DESCRIPTION OF TARGET FILE                      */
/*                                                                     */
/***********************************************************************/
  TFILE IS FILE(OUT_RECORD ,STORAGE=MAGAZINE);
  OUT_RECORD IS RECORD( OUT_INFO (0:1) ,PRE_CRIT='HEADER'
                      , OUT_TEXT
                      ,SIZE=VARIABLE(4050)):
  /* EACH 'OUT_RECORD' IS AT MOST 4050 CHARS LONG.  THE ACTUAL
  |  LENGTH IS DETERMINED AFTER AN ASSIGNMENT HAS BEEN MADE TO THE
  |  FIELDS 'OUT_INFO'  AND 'OUT_TEXT'
  */
     OUT_INFO IS GROUP( OUT_DATE_LABEL
                      , OUT_DATE
                      , OUT_TIME_LABEL
                      , OUT_TIME
                      , OUT_TITLE_LABEL
                      , OUT_TITLE);
       OUT_DATE_LABEL IS FIELD(CHAR(10)<-'(DATE)');
```

FIGURE 5A: DDL/DML LISTING

```
              OUT_DATE IS FIELD(CHAR(65)<='DATE');
              OUT_TIME_LABEL IS FIELD(CHAR(10)<-'(TIME)');
              OUT_TIME IS FIELD(CHAR(65)<='TIME');
              OUT_TITLE_LABEL IS FIELD(CHAR(7)<-'(TITLE)');
              OUT_TITLE IS FIELD(CHAR(*)<='TITLE' ,CONV='PROC4A');
            OUT_TEXT IS FIELD(CHAR(*)<='IN_TEXT' ,CONV='PROC4');
        MAGAZINE IS TAPE(FIXED(7500, 75) ,VOL_NAME=SS3548
                                                ,DENSITY=1600
                                               ,TAPE_LABEL=IBM_STD ,INT_NAME=SS3548)
    END;
```

FIGURE 5A; CONTINUED

```
INTRO:PROC;
/*----------------------------------------------------------------------
|  THIS IS A PRE_CRITERIA ROUTINE WHICH DETERMINES WHETHER THE SOURCE |
|  GROUP 'INFORMATION' APPEARS 0 OR 1 TIMES IN THE RECORD.            |
----------------------------------------------------------------------*/
DCL INTRODUCTION  CHAR(11) BASED(EXT_FLD_PTR);
/* THE FIRST 11 CHARACTERS OF THE GROUP 'INFORMATION' ARE
'((     [illegible].'  */
IF INTRODUCTION='((      .SGMO' THEN CRIT_RESULT=TRUE;
                                ELSE CRIT_RESULT=FALSE;
RETURN;
END INTRO;


FBISLEN:PROC;
/*----------------------------------------------------------------------
|  THIS IS A FIELD LENGTH ROUTINE WHICH CALCULATES THE LENGTH OF THE  |
|  SOURCE FIELD 'FBIS'.                                               |
----------------------------------------------------------------------*/
DCL FBIS_FLD CHAR(80) BASED(FBIS.PTR);
/*  THE FIELD 'FBIS' IS DELIMITED BY THE CHARACTER '='.  */
CONST_RESULT=INDEX(FBIS_FLD,'=')+5;
RETURN;
END FBISLEN;


ZEMLEN: PROC;
/*----------------------------------------------------------------------
|  THIS IS A FIELD LENGTH ROUTINE WHICH CALCULATES THE LENGTH OF THE  |
|  SOURCE FIELD 'ZEM'.                                                |
----------------------------------------------------------------------*/
DCL ZEM_FLD CHAR(80) BASED(ZEM.PTR);
/*  THE FIELD 'ZEM' IS DELIMITED BY THE CHARACTER '='.  */
CONST_RESULT=INDEX(ZEM_FLD,'=')+4;
RETURN;
END ZEMLEN;


TXXTLEN: PROC;
/*----------------------------------------------------------------------
|  THIS IS A FIELD LENGTH ROUTINE WHICH CALCULATES THE LENGTH OF THE  |
|  SOURCE FIELD 'TITLE'.                                              |
----------------------------------------------------------------------*/
DCL TITLE_FLD CHAR(200) BASED(TITLE.PTR);
/* LET TITLE BE EVERYTHING UP TO THE LABEL '(EXCERPTS)' OR '(TEXT)' */
CONST_RESULT= MAX(INDEX(TITLE_FLD,'(E'),INDEX(TITLE_FLD,'(T'))-1;
/* IF THE LABELS DO NOT EXIST, LET THE TITLE BE TERMINATED BY '==_'. */
IF CONST_RESULT<0 THEN CONST_RESULT=INDEX(TITLE_FLD,'==_')+11;
RETURN;
END TXXTLEN;
```

FIGURE 5A - CONTINUED

```
    PROC3: PROC;
    /*------------------------------------------------------------------
    I  THIS IS A FIELD LENGTH ROUTINE WHICH CALCULATES THE LENGTH OF THE  I
    I  SOURCE FIELD 'IN_TEXT'.                                            I
    -------------------------------------------------------------------*/
       DCL NUMBER PICTURE'99999' BASED(NUM.PTR);
    /* IF 'INFORMATION' EXISTS, THE LENGTH OF 'IN_TEXT' IS EQUAL TO THE
    CONTENTS OF 'NUM' MINUS THE LENGTH OF 'INFORMATION'; OTHERWISE, THE
    LENGTH OF 'IN_TEXT' IS SIMPLY THE CONTENTS OF 'NUM'. */
    IF CRIT_RESULT THEN CONST_RESULT=NUMBER-(A0+FBIS.LEN+ZEN.LEN+
                            TITLE.LEN);
                   ELSE CONST_RESULT=NUMBER;
      RETURN;
    END PROC3;

    HEADER:PROC;
    /*------------------------------------------------------------------
    I  THIS IS A PRE_CRITERIA ROUTINE WHICH DETERMINES WHETHER OR NOT THE I
    I  TARGET GROUP 'OUT_INFO' EXISTS FOR A PARTICULAR RECORD.             I
    -------------------------------------------------------------------*/
    /* IF THE SOURCE GROUP 'INFORMATION' EXISTS, THEN THE TARGET FIELD
    'OUT_INFO' EXISTS */
    CRIT_RESULT=CRIT_RESULT;
    RETURN;
    END HEADER;
```

FIGURE 5A - CONTINUED

```
PROC4: PROC(P);
/*-----------------------------------------------------------------------
!  THIS IS A CONVERSION ROUTINE WHICH DELETES SPECIAL BREAK CHARACTERS!
!  FROM THE SOURCE FIELD 'IN_TEXT' BEFORE IT IS MAPPED INTO THE TARGET!
!  FIELD  'OUT_TEXT'.                                                  !
-----------------------------------------------------------------------*/
DCL P POINTER;
DCL IN_TEXT CHAR(5000) BASED(P);
DCL TEXT CHAR(5000) VAR;
DCL LINE CHAR (7 );
DCL I, J;
CONV_FLD_CHAR='';
/* ASSIGN THE CONTENTS OF 'IN_TEXT' TO THE LOCAL VARIABLE 'TEXT' */
TEXT=SUBSTR(IN_TEXT,1,IN_TEXT.LEN);
/* DELETE BREAK CHARACTERS.  THEY ARE OF THE FORM:
            '==_            ',
            '==             ',  OR
            '=_             '.  */
LOOP: J=INDEX(TEXT,'=');
FIN: IF J=0 THEN
      DO;
      LINE=TEXT;
      CONV_FLD_CHAR=CONV_FLD_CHAR||LINE;
      RETURN;
      END;
      INNER_LOOP:
      IF (SUBSTR(TEXT,J+1,2)='=_')|
         (SUBSTR(TEXT,J+1,2)='= ')|
         (SUBSTR(TEXT,J+1,2)='_ ') THEN
               DO;
               LINE=SUBSTR(TEXT,1,J-1);
               CONV_FLD_CHAR=CONV_FLD_CHAR||LINE;
               IF (LENGTH(SUBSTR(TEXT,J))>11) THEN TEXT=SUBSTR(TEXT,J+11);
                                               ELSE RETURN;
               END;
           ELSE DO;
               I=INDEX(SUBSTR(TEXT,J+1),'=');
               IF I=0 THEN DO;
                           J=0;
                           GO TO FIN;
                           END;
                      ELSE DO;
                           J=I+J;
                           GO TO INNER_LOOP;
                           END;
               END;
      IF (SUBSTR(TEXT,1,1)=' ') THEN IF LENGTH (TEXT)=1 THEN RETURN;
                                                          ELSE
                                                          TEXT=SUBSTR(TEXT,2);
    GO TO LOOP;
    END PROC4;
```

FIGURE 5A - CONTINUED

```
PROGRAM: PROC(T);
/*-------------------------------------------------------------------
! THIS IS A CONVERSION ROUTINE WHICH DELETES SPECIAL BREAK CHARACTERS!
! FROM THE SOURCE FIELD 'TITLE' BEFORE IT IS MAPPED INTO THE TARGET !
! FIELD 'CUT_TITLE'.                                                !
-------------------------------------------------------------------*/
DCL T POINTER;
DCL TITLE CHAR(240) BASED(T);
DCL TTL CHAR(240) VAR;
DCL LINE CHAR(75);
DCL I,J;
CONV_FLD_CHAR='';
/* ASSIGN THE CONTENTS OF 'TITLE' TO THE LOCAL VARIABLE 'TTL'. */
TTL=SUBSTR(TITLE,1,TITLE.LEN);
/* DELETE BREAK CHARACTERS. THEY ARE OF THE FORM:
                    '==_       ',
                    '==        ', OR
                    '=_        '.                        */
LOOP:J=INDEX(TTL,'=');
FIN: IF J=0 THEN
   DO;
   LINE=TTL;
   CONV_FLD_CHAR=CONV_FLD_CHAR||LINE;
   IF (SUBSTR(CONV_FLD_CHAR,1,7)='TXXT_  ') THEN
   CONV_FLD_CHAR=SUBSTR(CONV_FLD_CHAR,8);
   RETURN;
   END;
INNER_LOOP:
IF (SUBSTR(TTL,J+1,2)='=_')|
   (SUBSTR(TTL,J+1,2)='= ')|
   (SUBSTR(TTL,J+1,2)='_ ') THEN
    DO;
    LINE=SUBSTR(TTL,1,J-1);
    CONV_FLD_CHAR=CONV_FLD_CHAR||LINE;
    IF (LENGTH(SUBSTR(TTL,J))>11) THEN TTL=SUBSTR(TTL,J+11);
    ELSE DO;
         IF (SUBSTR(CONV_FLD_CHAR,1,7)='TXXT_  ') THEN
         CONV_FLD_CHAR=SUBSTR(CONV_FLD_CHAR,8);
         RETURN;
         END;
    END;
 ELSE DO;
    I=INDEX(SUBSTR(TTL,J+1),'=');
    IF I=0 THEN DO;
                J=0;
                GO TO FIN;
                END;
           ELSE DO;
                J=J+I;
                GO TO INNER_LOOP;
                END;
    END;
IF (SUBSTR(TTL,1,1)=' ') THEN IF LENGTH(TTL)=1 THEN
    DO;
    IF (SUBSTR(CONV_FLD_CHAR,1,7)='TXXT_  ') THEN
    CONV_FLD_CHAR=SUBSTR(CONV_FLD_CHAR,8);
    RETURN;
    END;
ELSE TTL=SUBSTR(TTL,2);
GO TO LOOP;
END PROGRAM;
```

FIGURE 5A - CONTINUED

```
DDL COMPILER DIAGNOSTICS

NO ERRORS OR WARNINGS DETECTED
```

FIGURE 5B: DDL COMPILER DIAGNOSTICS

```
STMT_NO        IDENTIFIER                 ATTRIBUTE AND REFERENCES

   1                                      CONVERT

  10           DATE                       FIELD
                                            6, 23,

   5           DOCUMENT                   FIELD
                                            3,

  14           FBIS                       FIELD
                                            6,

   3           IN_RECORD                  RECORD
                                            2,

  17           IN_TEXT                    FIELD
                                            3, 28,

   6           INFORMATION                GROUP
                                            3,

   7           JUNK01                     FIELD
                                            6,

   9           JUNK02                     FIELD
                                            6,

  11           JUNK03                     FIELD
                                            6,

  13           JUNK04                     FIELD
                                            6,

  18           MAG_TAPE                   TAPE
                                            2,

  29           MAGAZINE                   TAPE
                                            19,

   8           MSGNO                      FIELD
                                            6,

   4           NUM                        FIELD
                                            3,

  23           OUT_DATE                   FIELD
                                            21,

  22           OUT_DATE_LABEL             FIELD
                                            21,

  21           OUT_INFO                   GROUP
                                            20,

  20           OUT_RECORD                 RECORD
                                            19,

  28           OUT_TEXT                   FIELD
```

FIGURE 5C: CROSS-REFERENCE TABLE

```
STMT_NO           IDENTIFIER                              ATTRIBUTE AND REFERENCES
                                                           20,

  25              OUT_TIME                                FIELD
                                                           21,

  24              OUT_TIME_LABEL                          FIELD
                                                           21,

  27              OUT_TITLE                               FIELD
                                                           21,

  26              OUT_TITLE_LABEL                         FIELD
                                                           21,

   2              SFILE                                   FILE
                                                             1,

  19              TFILE                                   FILE
                                                             1,

  12              TIME                                    FIELD
                                                             6, 25,

  16              TITLE                                   FIELD
                                                             6, 27,

  15              ZEM                                     FIELD
                                                             6,
```

FIGURE 5C: CONTINUED

REQUIRED DD-CARDS NEEDED TO RUN THE OBJECT (PL/1) PROGRAM:

```
 //DDLSRC DD DSN=DUMMY,VOL=SER=X00279,DISP=OLD,
 // UNIT=NINETRK,LABEL=(1,NL),
 // DCB=(RECFM=V,LRECL=0,BLKSIZE=4096,DEN=2)
 //DDLTAR DD DSN=SS3548,VOL=SER=SS3548,DISP=OLD,
 // UNIT=NINETRK,LABEL=(1,SL),
 // DCB=(RECFM=FB,LRECL=75,BLKSIZE=7500,DEN=3)
```

Figure 5d. Required JCL Needed to Run the Generated Program.

```
STMT LEVEL NEST
  1              [illegible]: PROC OPTIONS(MAIN);
  2     1        DCL DONE_FLAG  BIT(1) ALIGNED STATIC INIT('0'B),
                        CRIT_RESULT BIT(1) ALIGNED STATIC ,
                        LOCK_RESULT BIT(1) ALIGNED STATIC ,
                 ((COST_RESULT,J(400),FINSUB(400),MAXSUB(400),SUP(50),SUP1(50))  FIXED
                BIN STATIC ,
                        TRUE BIT(1) ALIGNED STATIC INIT('1'B),
                        FALSE BIT(1) ALIGNED STATIC INIT('0'B);
  3     1        DCL CEP FIXED BIN STATIC INIT(0B);
  4     1        DCL [illegible]_MAX([illegible]) FIXED BIN STATIC INIT(0B);
  5     1        DCL NEXT_FLD_PTR POINTER;
  6     1       DCL INBUFS CHAR( 4068)VAR STATIC;
  7     1        DCL INBUF( 4068)CHAR(1) BASED(INS);
  8     1       DCL
                1 SFILE,
                 2 [illegible]_RECORD,
                  3 LEN  FIXED BIN,
                  03 [illegible] ,
                   04 PTR POINTER,
                   04 LEN FIXED BIN INIT( 0B),
                  03 RECORD[illegible],
                   04 PTR POINTER,
                   04 LEN FIXED BIN INIT( 0B),
                  03 INFORMATION,
                   04 [illegible],
                    05 PTR POINTER,
                    05 LEN FIXED BIN INIT( 0B),
                   04 [illegible],
                    05 PTR POINTER,
                    05 LEN FIXED BIN INIT( 0B),
                   04 [illegible],
                    05 PTR POINTER,
                    05 LEN FIXED BIN INIT( 0B),
```

FIGURE 5E: PL/I OBJECT PROGRAM LISTING

```
                        04 DATE.
                         05 PTR POINTER,
                         05 LEN FIXED BIN INIT( 0B),
                        04 JUMP03,
                         05 PTR POINTER,
                         05 LEN FIXED BIN INIT( 0B),
                        04 TLF,
                         05 PTR POINTER,
                         05 LEN FIXED BIN INIT( 0B),
                        04 JUMP04,
                         05 PTR POINTER,
                         05 LEN FIXED BIN INIT( 0B),
                        04 LATS,
                         05 PTR POINTER,
                         05 LEN FIXED BIN INIT( 0B),
                        04 ZE  ,
                         05 PTR POINTER,
                         05 LEN FIXED BIN INIT( 0B),
                        04 TITLE,
                         05 PTR POINTER,
                         05 LEN FIXED BIN INIT( 0B),
                       03 IN_TEXT,
                        04 PTR POINTER,
                        04 LEN FIXED BIN INIT( 0B),
                         ADPL_ENTRY CHAR(1);
    9       1         DCL SOURCE_FLD_PROTO CHAR(8000) BASED(SEP);
   10       1         DCL CONV_FLD_CHAR CHAR(8000) VAR STATIC;
   11       1         DCL CONV_FLD_BIN BIT(8000) VAR STATIC;
   12       1         DCL TEMP_CHAR_FLD CHAR(8000) VAR STATIC;
   13       1         DCL [illegible]_FLD ENTRY(CHAR(*));
   14       1         DCL PROC4A  ENTRY(POINTER);
   15       1         DCL PROC4   ENTRY(POINTER);
   16       1        DCL OUTCNT   FIXED BIN STATIC INIT(0B);
   17       1        DCL SYS_OUTBUF CHAR(   75) BASED(SYS_OUTBUFP);
   18       1        DCL OUTBUF CHAR( 4950) STATIC;
   19       1         ON ENDFILE(PPLSRC) GO TO ADPL_FIN;
```

FIGURE 5E: CONTINUED

```
  1        INS=ADDR(INBUFS);
  1        DDL_READ:READ FILE(DDLSPC) INTO(INBUFS);
  1         MODEL_PARSE: CDP,#DDL_MAXCDP,OUTCDP=0B;
  1         SUB(*)=0B;
  1         SUB1(*)=1B;
  1        IN_RECORD.DUF.PTR=NULL;
  1        IN_RECORD.DOCUMENT.PTR=NULL;
  1        IN_RECORD.INFORMATION.JUNK01.PTR=NULL;
  1        IN_RECORD.INFORMATION.MSGNO.PTR=NULL;
  1        IN_RECORD.INFORMATION.JUNK02.PTR=NULL;
  1        IN_RECORD.INFORMATION.DATE.PTR=NULL;
  1        IN_RECORD.INFORMATION.JUNK03.PTR=NULL;
  1        IN_RECORD.INFORMATION.TIME.PTR=NULL;
  1        IN_RECORD.INFORMATION.JUNK04.PTR=NULL;
  1        IN_RECORD.INFORMATION.FBIS.PTR=NULL;
  1        IN_RECORD.INFORMATION.ZEN.PTR=NULL;
  1        IN_RECORD.INFORMATION.TITLE.PTR=NULL;
  1        IN_RECORD.IN_TEXT.PTR=NULL;
  1           IN_RECORD.DUF.PTR=ADDR(INBUF(CDP+1B));
  1           IN_RECORD.DUF.LEN=    5;
  1           CDP=CDP+    5;
  1          #DDL_MAXCDP=MAX(CDP,#DDL_MAXCDP);
  1           IN_RECORD.DOCUMENT.PTR=ADDR(INBUF(CDP+1B));
  1           IN_RECORD.DOCUMENT.LEN=    6;
  1           CDP=CDP+    6;
  1          #DDL_MAXCDP=MAX(CDP,#DDL_MAXCDP);
  1           NEXT_FLD_PTR=ADDR(INBUF(CDP+1B));
  1           CALL  [illegible]  ;
  1           IF [illegible]_RESULT THEN GO TO END001;
  1            IN_RECORD.INFORMATION.JUNK01.PTR=ADDR(INBUF(CDP+1B));
  1            IN_RECORD.INFORMATION.JUNK01.LEN=    6;
  1            CDP=CDP+    6;
  1            IN_RECORD.INFORMATION.MSGNO.PTR=ADDR(INBUF(CDP+1B));
  1            IN_RECORD.INFORMATION.MSGNO.LEN=  11;
  1            CDP=CDP+   11;
  1            IN_RECORD.INFORMATION.JUNK02.PTR=ADDR(INBUF(CDP+1B));
  1            IN_RECORD.INFORMATION.JUNK02.LEN=  23;
  1            CDP=CDP+  23;
```

FIGURE 5E: CONTINUED

```
60    1    IN_RECORD.INFORMATION.DATE.PTR=ADDR(INBUF(CDP+1B));
61    1    IN_RECORD.INFORMATION.DATE.LEN=    6;
62    1    CDP=CDP+    8;
63    1    IN_RECORD.INFORMATION.JUNK03.PTR=ADDR(INBUF(CDP+1B));
64    1    IN_RECORD.INFORMATION.JUNK03.LEN =    3;
65    1    CDP=CDP+    3;
66    1    IN_RECORD.INFORMATION.TIME.PTR=ADDR(INBUF(CDP+1B));
67    1    IN_RECORD.INFORMATION.TIME.LEN=    5;
68    1    CDP=CDP+    5;
69    1    IN_RECORD.INFORMATION.JUNK04.PTR=ADDR(INBUF(CDP+1B));
70    1    IN_RECORD.INFORMATION.JUNK04.LEN=   24;
71    1    CDP=CDP+   24;
72    1    IN_RECORD.INFORMATION.FBIS.PTR=ADDR(INBUF(CDP+1B));
73    1    CALL FBISLEN;
74    1    IN_RECORD.INFORMATION.FBIS.LEN=CONST_RESULT;
75    1    CDP=CDP+CONST_RESULT;
76    1    IN_RECORD.INFORMATION.ZEM.PTR=ADDR(INBUF(CDP+1B));
77    1    CALL ZEMLEN ;
78    1    IN_RECORD.INFORMATION.ZEM.LEN=CONST_RESULT;
79    1    CDP=CDP+CONST_RESULT;
80    1    IN_RECORD.INFORMATION.TITLE.PTR=ADDR(INBUF(CDP+1B));
81    1    CALL TXXTLEN;
82    1    IN_RECORD.INFORMATION.TITLE.LEN=CONST_RESULT;
83    1    CDP=CDP+CONST_RESULT;
84    1   END001;;
85    1   MODL_MAXCDP=MAX(CDP,MODL_MAXCDP);
86    1    IN_RECORD.IN_TEXT.PTR=ADDR(INBUF(CDP+1B));
87    1    CALL PECC3   ;
88    1    IN_RECORD.IN_TEXT.LEN=CONST_RESULT;
89    1    CDP=CDP+CONST_RESULT;
90    1   MODL_MAXCDP=MAX(CDP,MODL_MAXCDP);
91    1   CDP=MODL_MAXCDP;
```

FIGURE 5E: CONTINUED

```
  92      1           OUTBUF=' ';
  93      1              NEXT_FLD_PTR=ADDR(INBUF(CDP+19));
  94      1              CALL HEADER ;
  95      1              IF ¬CRIT_RESULT THEN GO TO END002;
  97      1               SUBSTR(OUTBUF,OUTCDP+1,    10)='(DATE)';
  98      1               OUTCDP=OUTCDP+    10;
  99      1               IF DATE.PTR¬=NULL THEN DO;
 101      1      1        SEP= DATE.PTR;
 102      1      1        SUBSTR(OUTBUF,OUTCDP+1,    65)=SUBSTR(SOURCE_FLD_PROTO,1, DATE.LEN);
 103      1      1        OUTCDP=OUTCDP+    65;
 104      1      1        END;
 105      1               ELSE  CALL #EMPTY_FLD(' DATE');
 106      1               SUBSTR(OUTBUF,OUTCDP+1,    10)='(TIME)';
 107      1               OUTCDP=OUTCDP+    10;
 108      1               IF TIME.PTR¬=NULL THEN DO;
 110      1      1        SEP= TIME.PTR;
 111      1      1        SUBSTR(OUTBUF,OUTCDP+1,    65)=SUBSTR(SOURCE_FLD_PROTO,1, TIME.LEN);
 112      1      1        OUTCDP=OUTCDP+    65;
 113      1      1        END;
 114      1               ELSE  CALL #EMPTY_FLD(' TIME');
 115      1               SUBSTR(OUTBUF,OUTCDP+1,     7)='(TITLE)';
 116      1               OUTCDP=OUTCDP+     7;
 117      1               IF TITLE.PTR¬=NULL THEN DO;
 119      1      1       CALL PROC4A ( TITLE.PTR);
 120      1      1       SUBSTR(OUTBUF,OUTCDP+1,LENGTH(CONV_FLD_CHAR))=CONV_FLD_CHAR;
 121      1      1       OUTCDP=OUTCDP+LENGTH(CONV_FLD_CHAR);
 122      1      1       END;
 123      1              ELSE  CALL #EMPTY_FLD(' TITLE');
 124      1            END002:;
 125      1            IF IN_TEXT.PTR¬=NULL THEN DO;
 127      1      1     CALL PROC4  ( IN_TEXT.PTR);
 128      1      1     SUBSTR(OUTBUF,OUTCDP+1,LENGTH(CONV_FLD_CHAR))=CONV_FLD_CHAR;
 129      1      1     OUTCDP=OUTCDP+LENGTH(CONV_FLD_CHAR);
 130      1      1     END;
 131      1           ELSE  CALL #EMPTY_FLD(' IN_TEXT');
 132      1           ;
 133      1           DO IDOL=1 TO OUTCDP BY     75;
 134      1      1      LOCATE SYS_OUTBUF FILE(DDLTAR);
 135      1      1      SYS_OUTBUF=SUBSTR(OUTBUF,IDOL);
 136      1      1    END;
 137      1           GO TO NEXT_READ;
 138      1           END_FILE::DL_DUMMY=' ';
```

FIGURE 5E: CONTINUED

```
139     1      INTRO:PROC;
               /*-----------------------------------------------------------------------
               !  THIS IS A PRE_CRITERIA ROUTINE WHICH DETERMINES WHETHER THE SOURCE  !
               !  GROUP 'INFORMATION' APPEARS 0 OR 1 TIMES IN THE RECORD.             !
               -----------------------------------------------------------------------*/
140     2      DCL INTRODUCTION CHAR(11) BASED(NEXT_FLD_PTR);
               /* THE FIRST 11 CHARACTERS OF THE GROUP 'INFORMATION' ARE
               '((      MSGNO.'  */
141     2      IF INTRODUCTION='((     MSGNO' THEN CRIT_RESULT=TRUE;
143     2                                      ELSE CRIT_RESULT=FALSE;
144     2      RETURN;
145     2      END INTRO;

146     1      FRISLEN:PROC;
               /*-----------------------------------------------------------------------
               !  THIS IS A FIELD LENGTH ROUTINE WHICH CALCULATES THE LENGTH OF THE  !
               !  SOURCE FIELD 'FRIS'.                                               !
               -----------------------------------------------------------------------*/
147     2      DCL FRIS_FLD CHAR(80) BASED(FRIS.PTR);
               /*  THE FIELD 'FRIS' IS DELIMITED BY THE CHARACTER '='.  */
148     2      CONST_RESULT=INDEX(FRIS_FLD,'=')+5;
149     2      RETURN;
150     2      END FRISLEN;

151     1      ZEMLEN: PROC;
               /*-----------------------------------------------------------------------
               !  THIS IS A FIELD LENGTH ROUTINE WHICH CALCULATES THE LENGTH OF THE  !
               !  SOURCE FIELD 'ZEM'.                                                !
               -----------------------------------------------------------------------*/
152     2      DCL ZEM_FLD CHAR(80) BASED(ZEM.PTR);
               /*  THE FIELD 'ZEM' IS DELIMITED BY THE CHARACTER '='.  */
153     2      CONST_RESULT=INDEX(ZEM_FLD,'=')+4;
154     2      RETURN;
155     2      END ZEMLEN;
```

FIGURE 5E: CONTINUED

```
156   1   TXTLEN: PROC;
          /*---------------------------------------------------------------*
          | THIS IS A FIELD LENGTH ROUTINE WHICH CALCULATES THE LENGTH OF THE |
          | SOURCE FIELD 'TITLE'.                                              |
          *---------------------------------------------------------------*/
157   2   DCL TITLE_FLD CHAR(240) BASED(TITLE.PTR);
          /* LET TITLE BE EVERYTHING UP TO THE LABEL '(EXCERPTS)' OR '(TEXT)' */
158   2   CONST_RESULT=MAX(INDEX(TITLE_FLD,'(E'),INDEX(TITLE_FLD,'(T'))-1;
          /* IF THE LABELS DO NOT EXIST, LET THE TITLE BE DELIMITED BY '==_'. */
159   2   IF CONST_RESULT<0 THEN CONST_RESULT=INDEX(TITLE_FLD,'== ')+11;
161   2   RETURN;
162   2   END TXTLEN;

163   1   PROC3: PROC;
          /*---------------------------------------------------------------*
          | THIS IS A FIELD LENGTH ROUTINE WHICH CALCULATES THE LENGTH OF THE |
          | SOURCE FIELD 'IN_TEXT'.                                            |
          *---------------------------------------------------------------*/
164   2       DCL NUMBER PICTURE'99999' BASED(NUM.PTR);
          /* IF 'INFORMATION' EXISTS, THE LENGTH OF 'IN_TEXT' IS EQUAL TO THE
          CONTENTS OF 'NUM' MINUS THE LENGTH OF 'INFORMATION'; OTHERWISE, THE
          LENGTH OF 'IN_TEXT' IS SIMPLY THE CONTENTS OF 'NUM'. */
165   2   IF CRIT_RESULT THEN CONST_RESULT=NUMBER-(80+ERIS.LEN+ZEM.LEN+
                                         TITLE.LEN);
167   2                          ELSE CONST_RESULT=NUMBER;
168   2    RETURN;
169   2   END PROC3;

170   1   HEADER:PROC;
          /*---------------------------------------------------------------*
          | THIS IS A PRE_CRITERIA ROUTINE WHICH DETERMINES WHETHER OR NOT THE |
          | TARGET GROUP 'OUT_INFO' EXISTS FOR A PARTICULAR RECORD.            |
          *---------------------------------------------------------------*/
          /* IF THE SOURCE GROUP 'INFORMATION' EXISTS, THEN THE TARGET FIELD
          'OUT_INFO' EXISTS */
171   2   CRIT_RESULT=CRIT_RESULT;
172   2   RETURN;
173   2   END HEADER;
```

FIGURE 5E: CONTINUED

```
174   1        PROCNA: PROC(T);
               /*-------------------------------------------------------------------------
               | THIS IS A CONVERSION ROUTINE WHICH DELETES SPECIAL BREAK CHARACTERS|
               | FROM THE SOURCE FIELD 'TITLE' BEFORE IT IS MAPPED INTO THE TARGET  |
               | FIELD 'CUT_TITLE'.                                                 |
               ------------------------------------------------------------------------*/
175   2        DCL T POINTER;
176   2        DCL STITLE CHAR(240) BASED(T);
177   2        DCL TTL CHAR(240) VAR;
178   2        DCL LINE CHAR(75);
179   2        DCL I,J;
180   2        CONV_FLD_CHAR='';
               /* ASSIGN THE CONTENTS OF 'TITLE' TO THE LOCAL VARIABLE 'TTL'. */
181   2        TTL=SUBSTR(STITLE,1,TITLE.LEN);
               /* DELETE BREAK CHARACTERS.  THEY ARE OF THE FORM:
                                  '==_        ',
                                  '==         ', OR
                                  '=_         '.                  */
182   2        LOOP:J=INDEX(TTL,'=');
183   2        FIN: IF J=0 THEN
184   2           DO;
185   2   1       LINE=TTL;
186   2   1       CONV_FLD_CHAR=CONV_FLD_CHAR||LINE;
187   2   1       IF (SUBSTR(CONV_FLD_CHAR,1,7)='TXXT_   ') THEN
188   2   1       CONV_FLD_CHAR=SUBSTR(CONV_FLD_CHAR,8);
189   2   1       RETURN;
190   2   1       END;
191   2        INTER_LOOP:
               IF (SUBSTR(TTL,J+1,2)='=_')|
                  (SUBSTR(TTL,J+1,2)='= ')|
                  (SUBSTR(TTL,J+1,2)='_ ') THEN
192   2           DO;
193   2   1       LINE=SUBSTR(TTL,1,J-1);
194   2   1       CONV_FLD_CHAR=CONV_FLD_CHAR||LINE;
195   2   1       IF (LENGTH(SUBSTR(TTL,J))>11) THEN TTL=SUBSTR(TTL,J+11);
197   2   1       ELSE DO;
198   2   2          IF (SUBSTR(CONV_FLD_CHAR,1,7)='TXXT_   ') THEN
199   2   2          CONV_FLD_CHAR=SUBSTR(CONV_FLD_CHAR,8);
200   2   2          RETURN;
201   2   2          END;
202   2   1       END;
```

FIGURE 5E: CONTINUED

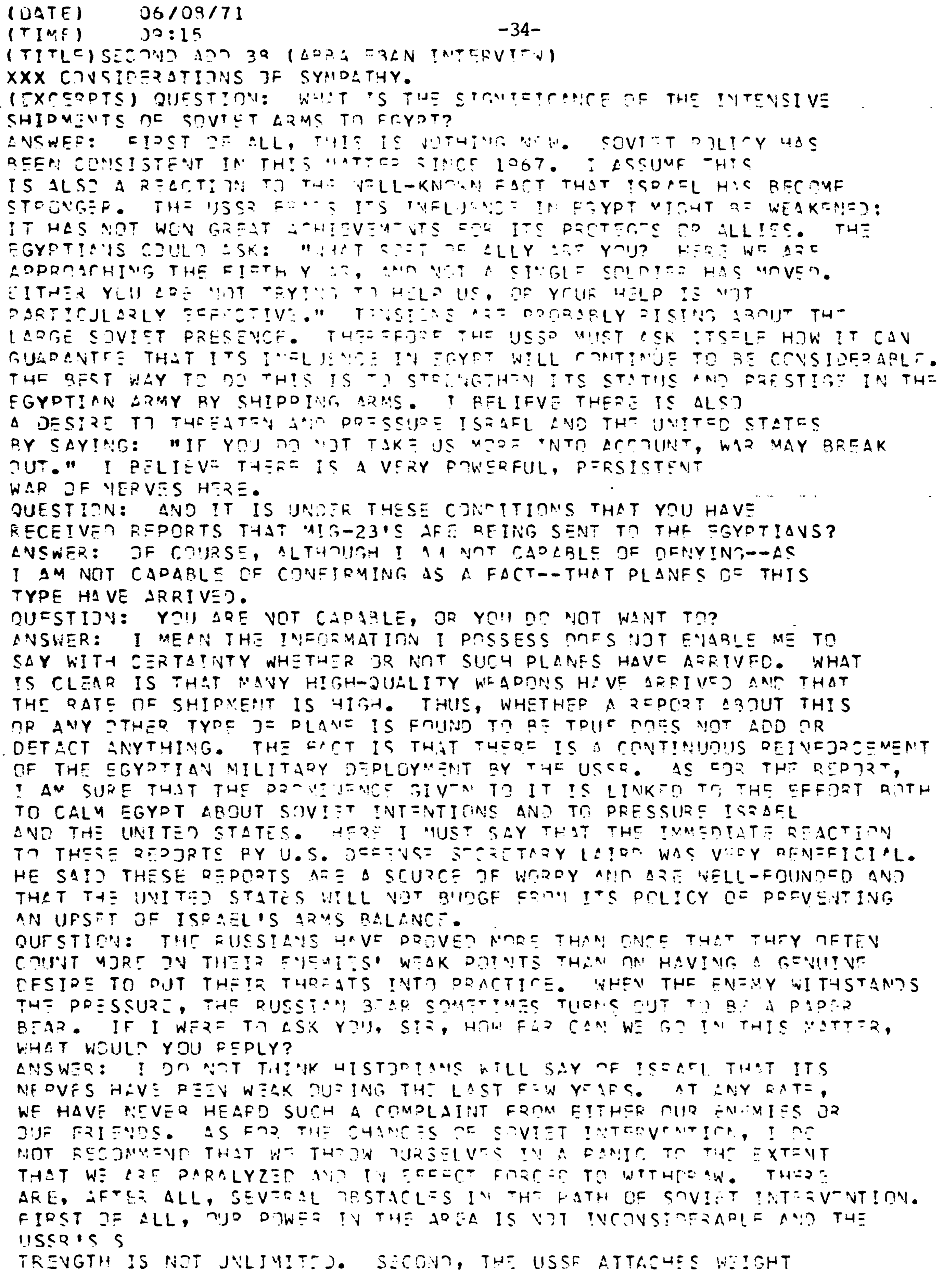
-31-

```
233    2        LOOP: J=INDEX(TEXT,'=');
234    2        FIN: IF J=0 THEN
235    2             DO;
236    2    1        LINE=TEXT;
237    2    1        CONV_FLD_CHAR=CONV_FLD_CHAR||LINE;
238    2    1        RETURN;
239    2    1        END;
240    2        INNER_LOOP:
                IF  (SUBSTR(TEXT,J+1,2)='=_')|
                    (SUBSTR(TEXT,J+1,2)='= ')|
                    (SUBSTR(TEXT,J+1,2)='_ ') THEN
241    2                  DO;
242    2    1             LINE=SUBSTR(TEXT,1,J-1);
243    2    1             CONV_FLD_CHAR=CONV_FLD_CHAR||LINE;
244    2    1             IF  (LENGTH(SUBSTR(TEXT,J))>11)  THEN  TEXT=SUBSTR(TEXT,J+11);
246    2    1                                                ELSE  RETURN;
247    2    1             END;
248    2             ELSE DO;
249    2    1             I=INDEX(SUBSTR(TEXT,J+1),'=');
250    2    1             IF  I=0  THEN   DO;
252    2    2                             J=0;
253    2    2                             GO TO FIN;
254    2    2                             END;
255    2    1                       ELSE  DO;
256    2    2                             J=I+J;
257    2    2                             GO TO INNER_LOOP;
258    2    2                             END;
259    2    1             END;
260    2        IF (SUBSTR(TEXT,1,1)=' ') THEN IF LENGTH (TEXT)=1 THEN RETURN;
                                                                    ELSE
262    2                                                             TEXT=SUBSTR(TEXT,2);
263    2
264    2        GO TO LOOP;
265    2        END PROC;
266    1        EMPTY_FLD:PROC(MODL_FLD_NAME);
267    2        DCL MODL_FLD_NAME  CHAR(*);
268    2        PUT EDIT('THERE WAS AN ATTEMPT TO MOVE AN EMPTY FLD')
                (SKIP,A);
269    2        PUT EDIT('FLD_NAME=',MODL_FLD_NAME,'WAS NOT MOVED')
                (SKIP,A,A,X(5),A);
270    2        RETURN;
271    2        END;
272    1        END [illegible];
```

FIGURE 5E: CONTINUED

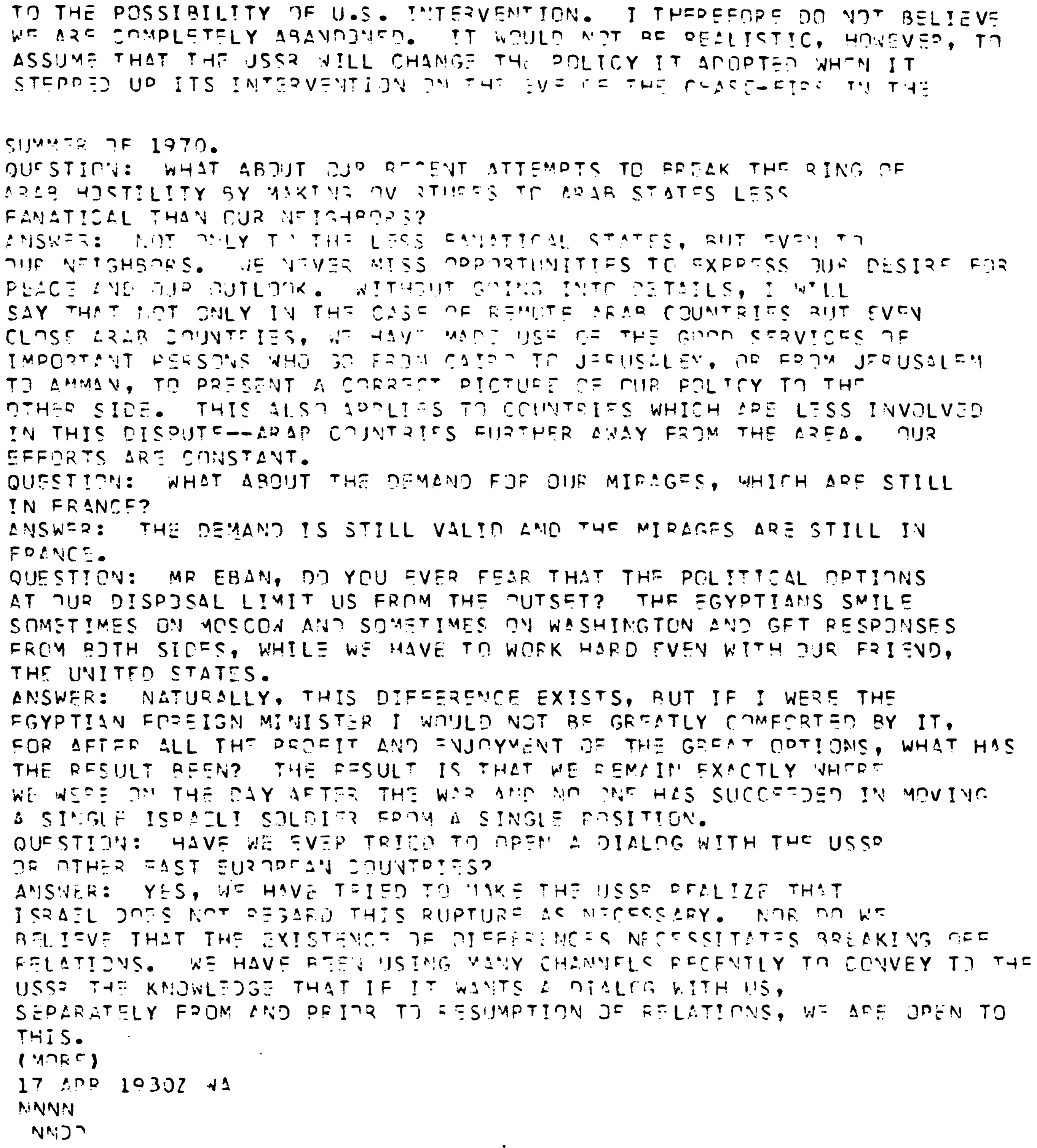
```
 203     2           ELSE DO;
 204     2    1         I=INDEX(SUBSTR(TTL,J+1),'=');
 205     2    1         IF I=0 THEN DO;
 207     2    2                 J=0;
 208     2    2                 GO TO FIN;
 209     2    2                 END;
 210     2    1             ELSE DO;
 211     2    2                 J=I+J;
 212     2    2                 GO TO INNER_LOOP;
 213     2    2                 END;
 214     2    1         END;
 215     2           IF (SUBSTR(TTL,1,1)=' ') THEN IF LENGTH(TTL)=1 THEN
 217     2             DO;
 218     2    1         IF (SUBSTR(CONV_FLD_CHAR,1,7)='TEXT_  ') THEN
 219     2    1         CONV_FLD_CHAR=SUBSTR(CONV_FLD_CHAR,8);
 220     2    1         RETURN;
 221     2    1         END;
 222     2           ELSE TTL=SUBSTR(TTL,2);
 223     2           GO TO LOOP;
 224     2           END PRECHA;

 225     1           PRECH: PROC(P);
                     /*---------------------------------------------------------------
                     | THIS IS A CONVERSION ROUTINE WHICH DELETES SPECIAL BREAK CHARACTERS|
                     | FROM THE SOURCE FIELD 'IN_TEXT' BEFORE IT IS MAPPED INTO THE TARGET|
                     | FIELD 'OUT_TEXT'.                                                  |
                     ----------------------------------------------------------------*/
 226     2           DCL P POINTER;
 227     2           DCL IN_TEXT CHAR(5000) BASED(P);
 228     2           DCL TEXT CHAR(5000) VAR;
 229     2           DCL FILL CHAR (75);
 230     2           DCL I, J;
 231     2           CONV_FLD_CHAR='';
                     /* MOVE THE CONTENTS OF 'IN_TEXT' TO THE LOCAL VARIABLE 'TEXT' */
 232     2           TEXT=SUBSTR(IN_TEXT,1,IN_TEXT.LEN);
                     /* DELETE ILLEGAL CHARACTERS. THEY ARE OF THE FORM:
                              '==_          ',
                              '==           ',  01
                              '=_           '.  */
```

FIGURE 5E: CONTINUED



```
(DATE)     06/08/71
(TIME)     09:15                         -34-
(TITLE)SECOND ADD 38 (ABBA EBAN INTERVIEW)
XXX CONSIDERATIONS OF SYMPATHY.
(EXCERPTS) QUESTION:  WHAT IS THE SIGNIFICANCE OF THE INTENSIVE
SHIPMENTS OF SOVIET ARMS TO EGYPT?
ANSWER:  FIRST OF ALL, THIS IS NOTHING NEW.  SOVIET POLICY HAS
BEEN CONSISTENT IN THIS MATTER SINCE 1967.  I ASSUME THIS
IS ALSO A REACTION TO THE WELL-KNOWN FACT THAT ISRAEL HAS BECOME
STRONGER.  THE USSR FEARS ITS INFLUENCE IN EGYPT MIGHT BE WEAKENED:
IT HAS NOT WON GREAT ACHIEVEMENTS FOR ITS PROTEGES OR ALLIES.  THE
EGYPTIANS COULD ASK:  "WHAT SORT OF ALLY ARE YOU?  HERE WE ARE
APPROACHING THE FIFTH YEAR, AND NOT A SINGLE SOLDIER HAS MOVED.
EITHER YOU ARE NOT TRYING TO HELP US, OR YOUR HELP IS NOT
PARTICULARLY EFFECTIVE."  TENSIONS ARE PROBABLY RISING ABOUT THE
LARGE SOVIET PRESENCE.  THEREFORE THE USSR MUST ASK ITSELF HOW IT CAN
GUARANTEE THAT ITS INFLUENCE IN EGYPT WILL CONTINUE TO BE CONSIDERABLE.
THE BEST WAY TO DO THIS IS TO STRENGTHEN ITS STATUS AND PRESTIGE IN THE
EGYPTIAN ARMY BY SHIPPING ARMS.  I BELIEVE THERE IS ALSO
A DESIRE TO THREATEN AND PRESSURE ISRAEL AND THE UNITED STATES
BY SAYING:  "IF YOU DO NOT TAKE US MORE INTO ACCOUNT, WAR MAY BREAK
OUT."  I BELIEVE THERE IS A VERY POWERFUL, PERSISTENT
WAR OF NERVES HERE.
QUESTION:  AND IT IS UNDER THESE CONDITIONS THAT YOU HAVE
RECEIVED REPORTS THAT MIG-23'S ARE BEING SENT TO THE EGYPTIANS?
ANSWER:  OF COURSE, ALTHOUGH I AM NOT CAPABLE OF DENYING--AS
I AM NOT CAPABLE OF CONFIRMING AS A FACT--THAT PLANES OF THIS
TYPE HAVE ARRIVED.
QUESTION:  YOU ARE NOT CAPABLE, OR YOU DO NOT WANT TO?
ANSWER:  I MEAN THE INFORMATION I POSSESS DOES NOT ENABLE ME TO
SAY WITH CERTAINTY WHETHER OR NOT SUCH PLANES HAVE ARRIVED.  WHAT
IS CLEAR IS THAT MANY HIGH-QUALITY WEAPONS HAVE ARRIVED AND THAT
THE RATE OF SHIPMENT IS HIGH.  THUS, WHETHER A REPORT ABOUT THIS
OR ANY OTHER TYPE OF PLANE IS FOUND TO BE TRUE DOES NOT ADD OR
DETACT ANYTHING.  THE FACT IS THAT THERE IS A CONTINUOUS REINFORCEMENT
OF THE EGYPTIAN MILITARY DEPLOYMENT BY THE USSR.  AS FOR THE REPORT,
I AM SURE THAT THE PROMINENCE GIVEN TO IT IS LINKED TO THE EFFORT BOTH
TO CALM EGYPT ABOUT SOVIET INTENTIONS AND TO PRESSURE ISRAEL
AND THE UNITED STATES.  HERE I MUST SAY THAT THE IMMEDIATE REACTION
TO THESE REPORTS BY U.S. DEFENSE SECRETARY LAIRD WAS VERY BENEFICIAL.
HE SAID THESE REPORTS ARE A SOURCE OF WORRY AND ARE WELL-FOUNDED AND
THAT THE UNITED STATES WILL NOT BUDGE FROM ITS POLICY OF PREVENTING
AN UPSET OF ISRAEL'S ARMS BALANCE.
QUESTION:  THE RUSSIANS HAVE PROVED MORE THAN ONCE THAT THEY OFTEN
COUNT MORE ON THEIR ENEMIES' WEAK POINTS THAN ON HAVING A GENUINE
DESIRE TO PUT THEIR THREATS INTO PRACTICE.  WHEN THE ENEMY WITHSTANDS
THE PRESSURE, THE RUSSIAN BEAR SOMETIMES TURNS OUT TO BE A PAPER
BEAR.  IF I WERE TO ASK YOU, SIR, HOW FAR CAN WE GO IN THIS MATTER,
WHAT WOULD YOU REPLY?
ANSWER:  I DO NOT THINK HISTORIANS WILL SAY OF ISRAEL THAT ITS
NERVES HAVE BEEN WEAK DURING THE LAST FEW YEARS.  AT ANY RATE,
WE HAVE NEVER HEARD SUCH A COMPLAINT FROM EITHER OUR ENEMIES OR
OUR FRIENDS.  AS FOR THE CHANCES OF SOVIET INTERVENTION, I DO
NOT RECOMMEND THAT WE THROW OURSELVES IN A PANIC TO THE EXTENT
THAT WE ARE PARALYZED AND IN EFFECT FORCED TO WITHDRAW.  THERE
ARE, AFTER ALL, SEVERAL OBSTACLES IN THE PATH OF SOVIET INTERVENTION.
FIRST OF ALL, OUR POWER IN THE AREA IS NOT INCONSIDERABLE AND THE
USSR'S S
TRENGTH IS NOT UNLIMITED.  SECOND, THE USSR ATTACHES WEIGHT
```

FIGURE 6: TARGET FILE LISTING



TO THE POSSIBILITY OF U.S. INTERVENTION. I THEREFORE DO NOT BELIEVE
WE ARE COMPLETELY ABANDONED. IT WOULD NOT BE REALISTIC, HOWEVER, TO
ASSUME THAT THE USSR WILL CHANGE THE POLICY IT ADOPTED WHEN IT
STEPPED UP ITS INTERVENTION ON THE EVE OF THE CEASE-FIRE IN THE

SUMMER OF 1970.
QUESTION: WHAT ABOUT OUR RECENT ATTEMPTS TO BREAK THE RING OF
ARAB HOSTILITY BY MAKING OVERTURES TO ARAB STATES LESS
FANATICAL THAN OUR NEIGHBORS?
ANSWER: NOT ONLY TO THE LESS FANATICAL STATES, BUT EVEN TO
OUR NEIGHBORS. WE NEVER MISS OPPORTUNITIES TO EXPRESS OUR DESIRE FOR
PEACE AND OUR OUTLOOK. WITHOUT GOING INTO DETAILS, I WILL
SAY THAT NOT ONLY IN THE CASE OF REMOTE ARAB COUNTRIES BUT EVEN
CLOSE ARAB COUNTRIES, WE HAVE MADE USE OF THE GOOD SERVICES OF
IMPORTANT PERSONS WHO GO FROM CAIRO TO JERUSALEM, OR FROM JERUSALEM
TO AMMAN, TO PRESENT A CORRECT PICTURE OF OUR POLICY TO THE
OTHER SIDE. THIS ALSO APPLIES TO COUNTRIES WHICH ARE LESS INVOLVED
IN THIS DISPUTE--ARAB COUNTRIES FURTHER AWAY FROM THE AREA. OUR
EFFORTS ARE CONSTANT.
QUESTION: WHAT ABOUT THE DEMAND FOR OUR MIRAGES, WHICH ARE STILL
IN FRANCE?
ANSWER: THE DEMAND IS STILL VALID AND THE MIRAGES ARE STILL IN
FRANCE.
QUESTION: MR EBAN, DO YOU EVER FEAR THAT THE POLITICAL OPTIONS
AT OUR DISPOSAL LIMIT US FROM THE OUTSET? THE EGYPTIANS SMILE
SOMETIMES ON MOSCOW AND SOMETIMES ON WASHINGTON AND GET RESPONSES
FROM BOTH SIDES, WHILE WE HAVE TO WORK HARD EVEN WITH OUR FRIEND,
THE UNITED STATES.
ANSWER: NATURALLY, THIS DIFFERENCE EXISTS, BUT IF I WERE THE
EGYPTIAN FOREIGN MINISTER I WOULD NOT BE GREATLY COMFORTED BY IT,
FOR AFTER ALL THE PROFIT AND ENJOYMENT OF THE GREAT OPTIONS, WHAT HAS
THE RESULT BEEN? THE RESULT IS THAT WE REMAIN EXACTLY WHERE
WE WERE ON THE DAY AFTER THE WAR AND NO ONE HAS SUCCEEDED IN MOVING
A SINGLE ISRAELI SOLDIER FROM A SINGLE POSITION.
QUESTION: HAVE WE EVER TRIED TO OPEN A DIALOG WITH THE USSR
OR OTHER EAST EUROPEAN COUNTRIES?
ANSWER: YES, WE HAVE TRIED TO MAKE THE USSR REALIZE THAT
ISRAEL DOES NOT REGARD THIS RUPTURE AS NECESSARY. NOR DO WE
BELIEVE THAT THE EXISTENCE OF DIFFERENCES NECESSITATES BREAKING OFF
RELATIONS. WE HAVE BEEN USING MANY CHANNELS RECENTLY TO CONVEY TO THE
USSR THE KNOWLEDGE THAT IF IT WANTS A DIALOG WITH US,
SEPARATELY FROM AND PRIOR TO RESUMPTION OF RELATIONS, WE ARE OPEN TO
THIS.
(MORE)
17 APR 1930Z WA
NNNN
NNDD

FIGURE 6: CONTINUED

C. Features and Capabilities of the DDL Language and Processor

(1) DDL language - data definition

A descriptive statement is of the form

name IS attribute (description);

where name is an identifier given to a particular attribute,

attribute can be a FILE, RECORD, GROUP, FIELD, TAPE, DISK, or CARD, and

description is a statement of the characteristics peculiar to that particular attribute name.

To categorize and illustrate these points, examine the following table:

ATTRIBUTE	DESCRIPTION	ILLUSTRATION (SEE FIG. 5A)
FILE	STATES THE NAME OF THE STORAGE MEDIUM ON WHICH THE FILE IS FOUND STATES THE NAME OF THE RECORDS WHICH COMPRISE THE FILE.	SFILE (LINE 2) IS THE SOURCE FILE. IT IS STORED ON MAG_TAPE AND IS COMPRISED OF RECORDS IN_RECORD.
RECORD	LISTS THE NAMES OF THE COMPONENT MEMBERS OF THE RECORD. GIVES THE LENGTH, IN BYTES OF A LOGICAL (USER) RECORD.	IN_RECORD (LINE 3) IS COMPOSED OF FIELDS CALLED NUM, DOCUMENT, AND IN_TEXT, AND OF THE GROUP CALLED INFORMATION. IT HAS A LOGICAL RECORD LENGTH OF 4088 BYTES.
GROUP	LISTS ALL THE SUB-GROUPS AND FIELDS WHICH COMPRISE IT.	INFORMATION (LINE 6) IS A GROUP COMPOSED OF 10 FIELDS.
FIELD	DESCRIBES THE ATTRIBUTES OF THE FIELD.	MSGNO (LINE 8) IS A FIELD 11 CHARACTERS LONG.
STORAGE MEDIA		
TAPE DISK CARD	THESE STATEMENTS GIVE A DESCRIPTION OF THE STORAGE MEDIUM AND THE PHYSICAL CHARACTERISTICS OF THE FILE, SUCH AS BLOCK SIZE, RECORD LENGTH, LABELS, EXTERNAL NAMES, ETC.	MAG_TAPE (LINE18) IS THE NAME OF A DESCRIPTION OF A TAPE WITH VARIABLE BLOCK LENGTH OF 4096 BYTES AND EXTERNAL NAME OF X00279. ALL OTHER ATTRIBUTES FOR THIS TAPE ARE DEFAULT TO THE SYSTEM.

(2) DDL language - data mappings

SYMBOL	MEANING	ILLUSTRATION (SEE FIGURE 5A)
<-	MAPS A CONSTANT INTO A TARGET FIELD	OUT_DATE_LABEL (LINE 22) IS A TARGET FIELD INTO WHICH IS MAPPED THE CONSTANT '(DATE)'
<=	MAPS A SOURCE FIELD INTO A TARGET FIELD	OUT_DATE (LINE 23) IS A TARGET FIELD INTO WHICH IS MAPPED THE CONTENTS OF THE SOURCE FIELD DATE

NOTE: THE DDL PROCESSOR MAKES IT VERY EASY TO ELIMINATE FIELDS FOUND IN THE SOURCE FILES WHEN CONSTRUCTING THE TARGET FILE. SEVERAL OF THE SOURCE FIELDS, SUCH AS ZEM AND FBIS, HAVE BEEN EXTRACTED BUT ARE NOT COPIED ONTO THE TARGET FILE, AS SEEN IN THE TARGET FILE DESCRIPTION IN FIGURE 5A.

(3) DML routines - data manipulation

TYPE OF DML ROUTINE	WHEN THE USER NEEDS TO PROVIDE A DML PROCEDURE	ILLUSTRATION (SEE FIGURE 5A)
LOCK	THIS TYPE PROCEDURE DETERMINES WHETHER OR NOT A PARTICULAR RECORD IS TO BE PROCESSED	-
LENGTH	THIS TYPE ROUTINE PERMITS VARYING FIELD LENGTHS FROM ONE RECORD TO THE NEXT BY COMPUTING THE LENGTHS AT RUN-TIME	THE LENGTH OF FBIS (LINE 14) IS COMPUTED DYNAMICALLY BY THE DML ROUTINES FBISLEN.
CRITERIA	THIS TYPE ROUTINE ALLOWS A GROUP OR FIELD TO HAVE A VARIABLE NUMBER OF MEMBERS BY CHECKING THE EXISTENCE OF EACH SUCCEEDING MEMBER.	INFORMATION (LINE 3) IS DECLARED TO APPEAR 0 TO 1 TIMES IN THE SOURCE RECORD DEPENDING ON CERTAIN CRITERIA CONSIDERED IN THE USER PROVIDED DML ROUTINE INTRO. THE TARGET GROUP OUT_INFO (LINE 20) APPEARS IF AND ONLY IF INFORMATION DOES.
CONVERSION	SPECIAL ROUTINES WHICH WILL MODIFY THE CONTENTS OF A FIELD	PROC 4 (LINE 28) SCANS FOR AND ELIMINATES SPECIAL BREAK CHARACTERS IN IN_TEXT BEFORE THE SOURCE FIELD IS STORED IN OUT_TEXT.

(4) The example shows the specification of a tape to tape copying and conversion of a file (MAG_TAPE (line 18) MAGAZINE (line 29)); conversion between files on other storage media can be handled as well.

D. Evaluation

Some interesting statistics regarding the lines of code, core requirements, and time requirements during the various phases of processing for this sample run are given in Figure 7. The statistics regarding the number of records processed and the time it took to transfer and manipulate the data reveal that the efficiency of the generated program is comparable to one written by the conventional manual manner. At the same time, however, it took a person a lot less time and lines of DDL statements than required in order to write the program by hand.

STORAGE AND TIME REQUIREMENTS:

	CORE	CPU SECONDS	I/O SECONDS	NUMBER OF STATEMENTS OF CODE
DDL COMPILATION	190K	2.82	14	30 DDL 125 DML
PL/I COMPILATION	IRRELEVANT SINCE PL/1 COMPILER USES AS MUCH CORE AS AVAILABLE FOR EFFICIENCY	2.15	4	272
EXECUTION	104K	33.9	27	3222 BYTES OF MACHINE LANGUAGE INSTRUCTION

OTHER EXECUTION-TIME STATISTICS:

	NUMBER OF BLOCKS TRANSFERRED	NUMBER OF LOGICAL RECORDS PER BLOCK	BLOCK LENGTH IN BYTES
INPUT FILE	143	1	4096
OUTPUT FILE	427	100	7500

MISCELLANEOUS FIGURES:
CPU seconds to produce each logical output record -- .0011
Channel seconds to produce each logical output record -- .0006
Minutes for trained human to write DDL/DML statements - 40
Number of Runs for trained humans to get a
bug-free program ------------------------------ 3

NOTE: All the above figures are based on a run of the example made on an IBM/168 Computing Facility

Figure 7 QUANTITATIVE STATISTICS

III. CURRENT RESEARCH

Current research is concentrated on developing a Processor whose principle is similar to DDL's, but which will be able to produce programs of a much wider class. The current processor is limited to generating programs which accept a single sequential file and produce a single target sequential file. The processor is currently being extended to enable it to accept descriptions of multiple files whose structure is not necessarily sequential. Such a processor would then generate programs which would accept and process any number of files and produce as output any number of desired files.

The processor would involve new techniques and procedures not present in the current processor. These techniques would have to be capable of producing programs which would input the files and sequence their reading in the most efficient manner and transform them into a set of desired output files, all this from non-procedural user specifications describing his source and target files. This would involve construction of networks of all involved data elements interacting with routines that process them, followed by designing and structuring the object program that will process the data most efficiently. The generated program would consist of all the I/O related instructions, the logical sequencing and control, and invocations of DML routines. Thus, generated modules would again be the top-level of the desired program with the low level data manipulation to be provided by the user's DML routines. Such a processor would be another step towards the automatic generation of data processing programs and would obviously cover a much wider class of applications.

IV. OTHER POSSIBLE FUTURE WORK

Other possible future extensions to the system might include the following areas in order to make the processor of wider use:

(a) If automatic generation of COBOL rather than PL/1 programs is desired, this could be effected by rewriting the output modules of our processor.

(b) As we have mentioned above, we could accept another specification language other than our DDL with the aid of our SAPG.

(c) Research could be carried out to develop modules to generate reports logically. That is, a processor which accepts non-procedural and logical description of desired reports could be developed to design and produce programs for report writing much in the same way that the DDL Processor generates files.

Complete automation of generating programs is, of course, a long way off, but the above-described system is a small step in that direction. For a more complete analysis of the role of automatic program generation in software development, see [2].

2. N.S. Prywes, "The Role of Automatic Program Generation in Software Development," Moore School of Electrical Engineering, University of Pennsylvania.

ACKNOWLEDGEMENTS

The author would like to thank Dr. N.S. Prywes, the project supervisor, for his direction and supervision of the project. Acknowledgement should be made to Dr. J.A. Ramirez who, as a graduate student, was system designer and original project leader of the team that implemented the first version of the DDL processor.

AUTOMATIC GENERATION OF SOFTWARE SYSTEMS - A SURVEY

N. PRYWES

Department of Computer and Information Sciences, University of Pennsylvania

ABSTRACT

Research and development on automatic programming has been underway for nearly twenty-five years, and will continue for many years. An ultimate objective is the development of methods and facilities to produce software automatically, on demand, for the businessman, industrialist or scientist. This paper considers this proposition and examines a number of related questions: what is a plausible sequence of developments?, what is the social-economic need for such automatic programming advances?, what is the extent of the potential impact of such advances?, what are the envisaged methodologies? how will they be employed? and what are the open research questions? The paper is intended to provide motivation for, and critique of, research toward the above ultimate goal of automatic programming.

AUTOMATIC GENERATION OF SOFTWARE SYSTEMS - A SURVEY *

Noah S. Prywes

Department of Computer and Information Sciences, University of Pennsylvania
Philadelphia, Pa., 19174

1. INTRODUCTION:

Research and development on automatic programming has been underway since early applications of digital computers, for nearly twenty-five years, and will continue for many years. It has started with the use of the computer to relieve the programmer of miscellaneous detail[1]. Its ultimate objective is envisaged as a situation where software would be automatically generated for the businessman, industrialist or scientist, on demand. The computer would have to access the combined knowledge of the respective field of the application and of computer system design and programming in order to generate automatically the needed software, ready for use. This ultimate objective of automatic programming is the main subject of this paper.

The paper is organized in two parts. The first part discusses the wide spread concern that cost and effectiveness of software development threaten the continued growth of use of computers. This is offered as a motivation for acceleration of the research on automatic programming to achieve the above cited ultimate objective. The second part of the paper attempts an assessment of the methods of achieving the ultimate objective of all automatic software generation.

The first part of this paper is divided into two sections. Historic and economic contexts of advances in automatic programming are discussed briefly in the next section, as a basis for evaluation and projection of progress.

1. For instance, early development of mathematical routines at Harvard University, University of Pennsylvania and Cambridge University.

*Work supported by ONR Information Systems Program, Contract #N00014-67-A-0216-0014

From midway vintage point between the beginning and ultimate goal, it is possible to see that progress on automatic programming has followed a natural sequence where earlier work established foundations for later advances. Also, that rate of progress has been a function of the economic pressures of demand and costs for programming in employment of computers in society.

Then, in the section that follows, the software development cycle is analyzed,to identify the components that need to be automated and to assess the benefits from their automation.

Subsequent two sections constitute the second part of the paper, which discusses future advances in automatic generation of software. The advances described in the first of the two sections are currently underway, and are believed to be capable of replacing the human programmers who are engaged in translating non-procedural specifications into computer programs for processing data. This is referred to in the following as automatic design and implementation of program modules. It is required that a computer engaged in performing this function, have access to knowledge on computer system design and programming, but that it may be independent of knowledge in the field of a specific application. The second part, automatic generation of overall problem requirements, is based on computer access also to knowledge in respective specific application fields.

PART I CURRENT PROBLEMS IN SOFTWARE DEVELOPMENT

2. OVERVIEW OF HISTORIC AND ECONOMIC CONSIDERATIONS

The total process of development and use of data processing systems has been pictured as consisting of many layers. Figure 1 illustrates such a multi-layer structure by showing thirteen such layers. The examination of this structure is interesting in that it provides information on future areas of progress of automatic programming. The layers in Figure 1 are named to indicate their general function. A reader desirous of more specific definitions

AUTOMATION DEMAND	OVERALL PROBLEM REQUIREMENTS
EVALUATION OF ALTERNATIVE AUTOMATION PLANS	
FUNCTIONAL SPECIFICATIONS OF COMPUTER SYSTEM	
EVALUATION OF FILE STRUCTURE ALTERNATIVES	PROGRAM DESIGN AND IMPLEMENTATION
PROGRAM MODULE IDENTIFICATION	
PROGRAM MODULE INPUT/OUTPUT LOGICAL SPECIFICATIONS	
PROGRAM DESIGN	
HIGH LEVEL (COMPILER, JCL&DM) LANGUAGE PROGRAMMING	
HIGH LEVEL TO ASSEMBLY LANGUAGE	PROGRAM TRANSLATION AND OPTIMIZATION
ASSEMBLY LANGUAGE TO MACHINE CODE	
PROBLEM AND SUPERVISOR MACROS	EXECUTION
PROBLEM INSTRUCTION SEQUENCES	
MICRO INSTRUCTION PROCESSING OF DATA	

FIG. I TRANSLATION/DEVELOPMENT LAYERS IN DATA PROCESSING SYSTEMS AND IN THEIR APPLICATIONS

for the layers may have to add additional layers. The top three layers in Figure 1 are associated with the process of determining automation requirements, which sometimes requires definition of the problem. Note also that typically, a problem environment includes several other components beside the computer. The next five layers concern the design and implementation of software, in a language convenient for the problem. The two layers below, are concerned with the translation and optimization of the programs. Finally, the three lowest levels are concerned with the execution of the programs to perform the desired automation functions.

One way of regarding these layers is as a bottom-up sequence, in the sense that the automation of a bottom layer is utilized in building the layer above it. Historically, a bottom-up approach has been taken in automating these layers. The first layers to be automated were those associated with program execution. This was followed by the layers associated with language translation and program optimization. To date, a great emphasis has also been placed on the efficiency and utility of programming languages[2]. Improved facilities were quickly put to a test in a variety of applications. Relatively, little automation has been applied to the top seven layers in Figure 1 and the translations from one layer to the one below it have been done manually by appropriate software specialists.

Another, top-down view of Figure 1, pictures the layers as a series of transducers - translators, where the flow is downward so that output of a top layer is used as an input to the layer below it. (A feedback flow is used for corrections and modifications)

While improvements in hardware, system software and programming languages

2 Jean E. Sammet, "Programming Languages: History and Future," Comm. of the ACM, July 1972, Vol. 15, Number 7, pp. 607-610.

continue to make programming more effective, they are not sufficient to compensate for the dramatic projected increases in demand for software. One approach to projection of demand of software is as follows: The ratio of cost of software to hardware, which in 1970 was 2:1, is projected to increase by 1985 to 10:1[3]. The U.S. Bureau of Labor Statistics 1970 estimates show a work force of 360,000 engaged in software development in the United States. Of these 137,000 were classified as business programmers, 97,000 as business systems analysts and the remainder were employed in systems and scientific software[4]. These estimates are on the low side as they exclude some segments of the U.S. economy and were registered at a low economic point of 1970. In spite of the decreasing costs of computer equipment, total U.S. revenues from computing hardware have been increasing since 1970 at a real rate that may be averaged at 10% annually. Though costs of computers have been decreasing, considering the large numbers of potential users and new areas of industrial applications, this rate of increase is projected to continue for the period of 1970 to 1985. It will amount to quadrupling the real annual cost of new computer equipment by 1985. When this is coupled with the 1985 projection of 10:1 ratio of cost of programming to the cost of hardware, this would amount to 20 fold increase in programming manpower by 1985 (a 22% annual growth rate). Obviously such a requirement would constitute not only a financial obstacle to advances in use of computers, but also such a labor force cannot be recruited, trained, or controlled effectively. In order to keep software manpower

3 Barry W. Boehn, "Software and Its Impact: A Quantitative Assessment," Datamation, May 1973, pp. 48-59.

4 B Gilchrist and R.E. Weber, "Employment of Trained Computer Personnel-A Quantitative Survey," AFIPS Conference Proceedings, Vol. 40, May 1972, pp. 641-648.

increases at a pace with increases of total hardware costs (10% annual increase), the manpower requirements will have to be reduced by 80% by 1985, as compared with present (mostly manual) methods.

Continuing the bottom-up sequence of development, there are considerable efforts underway on the complete automation of the five layers in Figure 1 that constitute program design and implementation. The bottom two of the five layers, will provide automatic generation of programs, based on program module logical specifications; namely, the automatic production of Ad Hoc programs in compiler languages such as COBOL or PL/1. This will reduce, particularly, the requirements for business programmers. As indicated, these programmers represent 60% of those employed in business programming. Subsequently, work of business system analysts, who are engaged in the top three of the five layers mentioned, will be reduced as well.

The manpower associated with the top four layers in Figure 1, of determination of overall problems requirements, represents currently a relatively small amount of manpower, which is primarily trained in the respective applications rather than in computer systems. Most probably the bottom-up sequence of developments will continue and will gradually effect the top layers in Figure 1. However, the automation of determining overall problems requirements may be delayed until automatic program design and generation methodology (in lower layers) has been well established. The methods under development are described in Part II below.

3. ANALYSIS OF THE SOFTWARE DEVELOPMENT PROCESS

Advances in automation of programming are needed in large scale software development projects, particularily in the business data processing area which employs the majority of programmers and where there is likely to be the most growth in demand for such services. Examples of such systems are a new financial system required by top management to provide new reporting for improved financial control and planning, or systems to support new products such as reservations services, medical services or a new insurance plan. Large scale projects are not the creation of a single or a small group of managers. Such systems have an impact on top management, business specialists in areas such as accounting or finance, operational staff of the organization, and finally the public that interact with the organization such as customers, vendors, etc. This is one of the reasons why large scale software development projects span several years of development time and involve activities of various groups within an organization. The objective below is to put programming automation in the context of such projects, to determine the requirements of research in greater detail. Figure 2 gives an overview of the software development process, indicating the respective phases, by whom they are performed, the end products and the languages used in the documentation.

Data on distribution of labor and costs in software development phases is difficult to obtain for two reasons. First, phase classification in accounting of development costs is rarely adhered to systematically[5]. Second, costs of software development projects vary greatly, depending on the quality of planning and execution. Costs frequently vary in the ratio of 2:1 for similar project and in some instances the ratio is much higher (20:1). Therefore, the estimates that are made below indicate widely accepted typical values.

5 Nelson, E.A., "Management Handbook For The Estimation of Computer Programming Costs," SDCTM 3224, 1966.

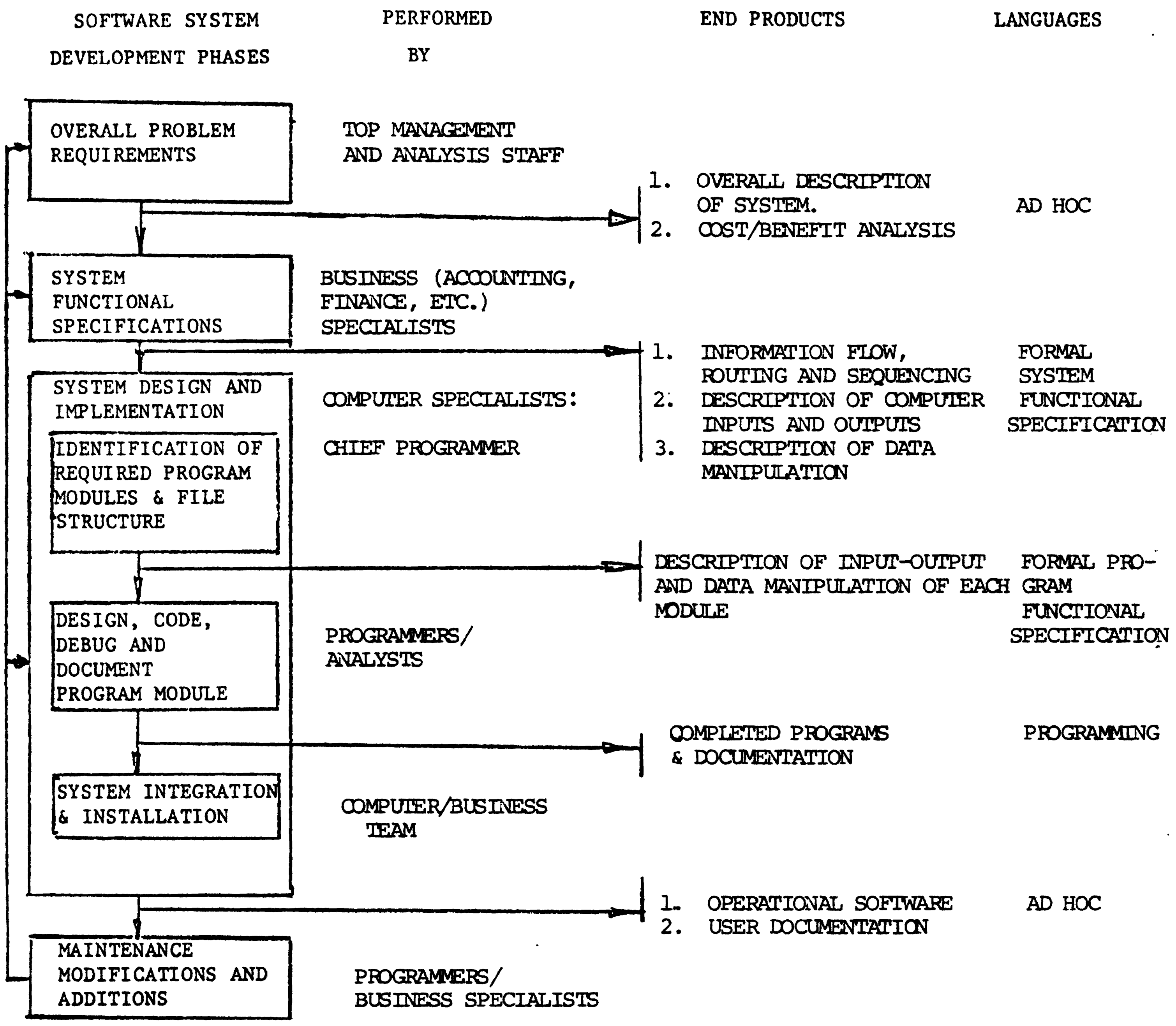

SOFTWARE SYSTEM DEVELOPMENT PHASES
PERFORMED BY
END PRODUCTS
LANGUAGES
OVERALL PROBLEM REQUIREMENTS
TOP MANAGEMENT AND ANALYSIS STAFF
1. OVERALL DESCRIPTION OF SYSTEM.
2. COST/BENEFIT ANALYSIS
AD HOC
SYSTEM FUNCTIONAL SPECIFICATIONS
BUSINESS (ACCOUNTING, FINANCE, ETC.) SPECIALISTS
1. INFORMATION FLOW, ROUTING AND SEQUENCING
2. DESCRIPTION OF COMPUTER INPUTS AND OUTPUTS
3. DESCRIPTION OF DATA MANIPULATION
FORMAL SYSTEM FUNCTIONAL SPECIFICATION
SYSTEM DESIGN AND IMPLEMENTATION
COMPUTER SPECIALISTS:
CHIEF PROGRAMMER
IDENTIFICATION OF REQUIRED PROGRAM MODULES & FILE STRUCTURE
DESCRIPTION OF INPUT-OUTPUT AND DATA MANIPULATION OF EACH MODULE
FORMAL PRO- GRAM FUNCTIONAL SPECIFICATION
DESIGN, CODE, DEBUG AND DOCUMENT PROGRAM MODULE
PROGRAMMERS/ ANALYSTS
COMPLETED PROGRAMS & DOCUMENTATION
PROGRAMMING
SYSTEM INTEGRATION & INSTALLATION
COMPUTER/BUSINESS TEAM
1. OPERATIONAL SOFTWARE
2. USER DOCUMENTATION
AD HOC
MAINTENANCE MODIFICATIONS AND ADDITIONS
PROGRAMMERS/ BUSINESS SPECIALISTS

FIG. 2 OVERVIEW OF SOFTWARE SYSTEMS DEVELOPMENT PROJECTS

These estimates that are based on published reports[3,6,7,18], and discussions with colleages. The estimates are made to illustrate an approach to the projection of the possible impacts of automating software development. A reader whose experience indicates different estimates can substitute his own estimates and verify the validity of the conclusions. The estimates below are to be considered as averages for all business data processing projects. Similar analysis can be performed for other areas of programming applications.

In the discussion below, reference is made to the phases in Figure 2.

Overall Problem Requirements - This is a project kick-off phase concerned primarily with determining what information management needs and indications of the resources needed for the development. The initiation of large scale software development projects is usually prompted by important needs determined by top management. The phase includes preparing preliminary estimates supported by cost/benefits analysis. This activity requires expertise in the application area, as well as sizing computer requirements. The written description of such activity together with the indication of required resources and the decision to proceed are then communicated to individuals charged with carrying out the decision. To give a relative weight, this phase is estimated at 8% of total project cost.

System Functional Specifications - These specifications describe the information flow involving humans, communications and computers. They define the transactions that would be involved as well as management reports necessary for control of the activities in the system.

6 F.T. Baker, "Chief Programmer Team Management of Production Programming," IBM Systems Journal, Vol. II, I, 1972, pp. 57-73.

7 J.D. Aron, "Estimating Resources For Large Programming Systems," 2nd NATO Conference, Software Engineering Report, Rome 1969, pp. 68-79, NATO, Brussels.

The computer is envisaged as a black box where the functional specifications describe all the transactions that are entered into the computer system, and the management reports and user information produced by the system. The group that prepares the specifications need not include computer specialists. However, as will be related further in Part II, it has been found useful to employ a formal functional specification language in documenting the system, which can be processed automatically to indicate incompleteness or inconsistency. Ineffectiveness and waste result where the functional specifications are incomplete or ambiguous thus causing redesign and reprogramming. The discipline imposed by the use of such specification languages has been considered as beneficial to reducing total development costs, especially in controlling the progress and flow of work during a development project. 22% of the total costs of software development projects is estimated for this phase.

The remaining 70% of the cost of a software development project is estimated to be for development of program modules that perform in accordance with the functional specification. If the functional specifications are complete, the work in this phase requires only computer oriented skills. Once the total environment and hardware requirements have been preliminarily determined the software development in this phase may be subdivided into three steps.

File Structure and Module Design - The first step in this phase is the specification of file structures and program modules. The functional specifications written by the business specialists in the previous step are now broken down into distinct logical specifications for each program module. Furthermore, the data structures on which the modules act can now be specified.

A program module is typically associated with a transaction, an updating or a reporting function. In a typical $1,000,000 project there would be 50 to 100 program modules each consisting of 1000 to 2000 lines of high level language code. The objective of this step is to obtain a sequence of input-output operations and data organization that minimize the cost of data processing. The product of the design activity is the generation of file structure specifications and program module logical specifications which incorporate the system functional specification on a per-module basis. 15% of project cost is estimated to go into creating this specification. (In many projects this step is integrated with program design, described below, and it is not possible then to estimate costs separately).

Program Design, Code and Debug - The next step is the design of each program module, its coding, debugging and documentation. The difficulties encountered currently in this step are due to having to supervise large numbers of programmers, and the low management visibility of progress or lack thereof. Therefore, problems are discovered late during the debugging of individual program modules, even later or during installation, resulting in delays and costs beyond original estimates. This step is estimated to require currently 25% of the total project cost.

System Integration and Installation - The last step in the programming phase consists of integration of individual program modules into the total system, testing and sometimes parallel operation of the system. It is during this phase that problems that have not been previously visible to management come to light. They may appear as malfunctions that must be corrected. If difficulties are discovered in user operation and control of the system, the indicated changes must be communicated to modify the functional specification of either the entire system or specific program modules and only then can modifications be

TABLE I SUMMARY OF ESTIMATES OF PERCENTAGES OF SOFTWARE DEVELOPMENT COSTS ATTRIBUTABLE TO PHASES, AND ENVISAGED REDUCTIONS DUE TO AUTOMATION

PHASE DESCRIPTION		ESTIMATED % OF TOTAL COST WITH TRADITIONAL PROGRAMMING APPROACH	ESTIMATE OF POTENTIAL REDUCTIONS (IN % OF COST) DUE TO:				
			AUTOMATIC PROGRAM GENERATION	AUTOMATIC FILE AND MODULE DESIGN	AUTOMATIC FUNCTIONAL SPECS GENERATION	AUTOMATIC PROBLEM REQUIREMENTS DETERMINATION	TOTAL % PROGRAMMING COSTS WITH ALL-AUTOMATIC FACILITIES
DETERMINATION OF PROBLEM REQUIREMENTS		8				-6	2
GENERATION OF SYSTEM FUNCTIONAL SPECS		22	+5		-20	-5	2
PROGRAMMING:	70						
FILE AND MODULE DESIGN		15		-10	-2		3
CODE, TEST, DEBUG AND DOCUMENT		25	-17	-5			3
INTEGRATION & INSTALLATION		30	-20				10
TOTAL		100	-32	-15	-22	-11	20

made. Depending on the quality of testing prior to installation this last sub-phase may account for 30% of the total costs of the project.

Maintenance - After the system has been in operation, it will require maintenance to correct errors not determined during the installation process. But even more so it will require maintenance to modify and add facilities. This is a post-software-development phase which must be facilitated during the development itself. The software development should provide a method by which modifications can be orderly entered in the specifications, the program modules, the documentations and the operational system itself.

The automation of any one phase imposes a discipline on the entire process which brings savings in the adjacent phases of the process as well. For instance, the discipline provided by a requirement to specify program modules in a formal functional specification language and an automatic generation of programs would assure that relatively bug free programs be provided to the system integration phase.

Table 1 summarizes the estimates of distribution of costs over the phases of the software development process as indicated previously. These estimates are necessarily subjective as indicated previously. A bottom up sequence of development is assumed in Table 1, with automatic program design and generation accomplished first* followed by automatic file and module design, automatic functional specification and finally automation of the determination

* The estimates of savings due to automation of program generation are inclusive of savings due to improved operating systems and language facilities shown in the five bottom layers of Figure 1.

problem requirements. The objective of such a succession of developments is to reduce the cost of software development by 80% over a period lasting to 1985, as illustrated in Fig. 3. The estimates of savings of cost due to the four envisaged increments in automation of programming, represent a guess of the author, after extensive discussions with co-workers in this field. The estimates of potential contributions of automation of software development phases are also based on considering the potential methodologies as described below. Therefore, more will be said on these estimates, at the end of Part II.

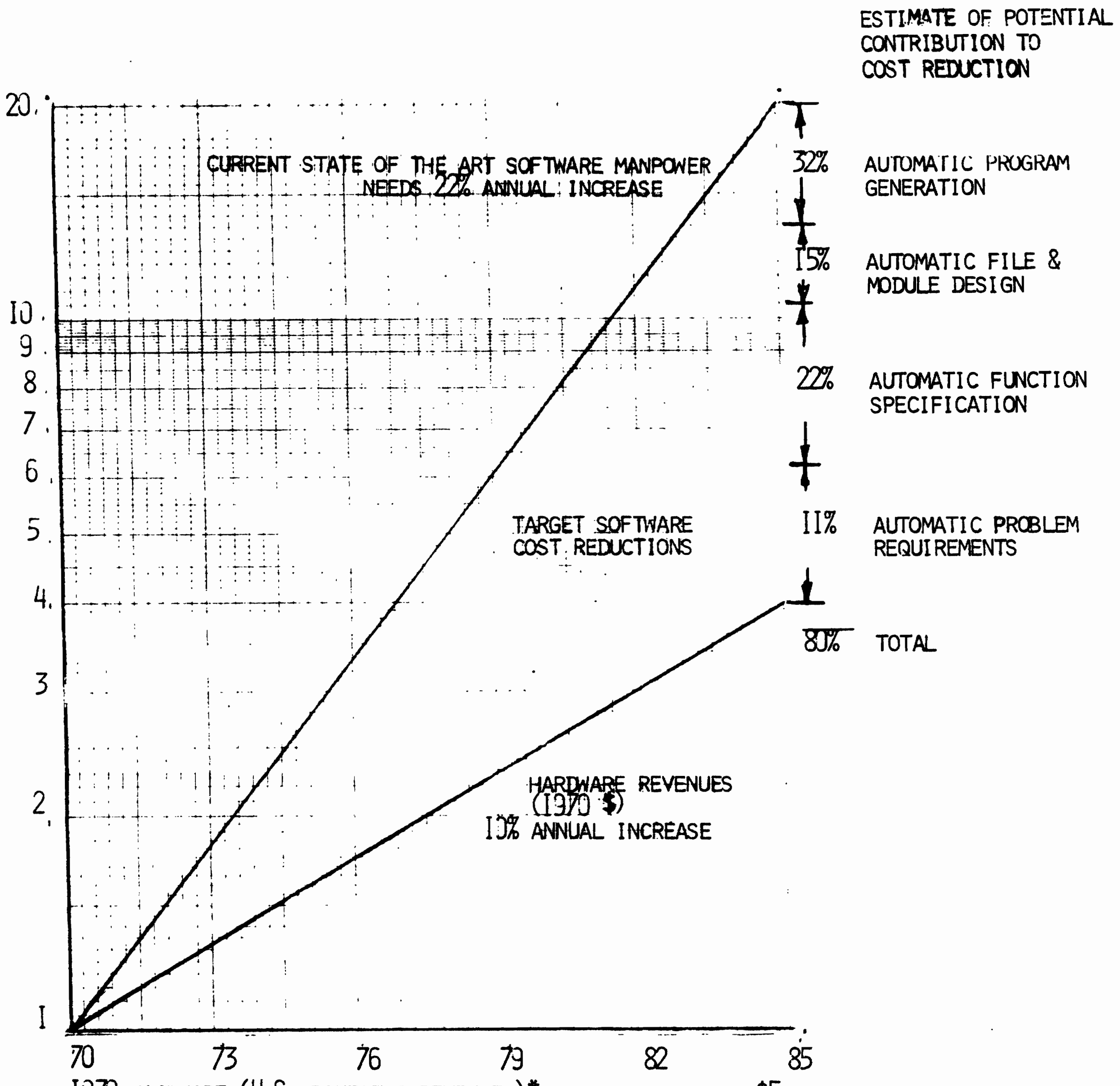
PART II FUTURE ADVANCES IN AUTOMATION OF SOFTWARE DEVELOPMENT

4 AUTOMATIC DESIGN AND IMPLEMENTATION OF PROGRAMS

This section examines the automation of the five layers in Figure 1 shown as constituting program design and implementation. This activity requires only computer related knowledge. The automation activities of the higher layers in Figure 1 that require knowledge of specific applications are discussed in a subsequent section. The "pure" computer art dependent structures are on a higher abstraction level than the business functions of a specific type of industry. The automating of the generation of these structures can serve as a foundation, and can be used in a variety of businesses and industry applications.

The discussions in this section involve only logical-abstract descriptions of the automation. The translations from the logical-abstract models to physical devices are performed in the control programs in lower five layers of Figure 1, such as Data and Teleprocessing Access Methods, etc.

However, it's important to note at this point that approaches to automating software development, where application and computer related knowledge



1970 HARDWARE (U.S. DOMESTIC REVENUES)* $5 BILLION

1970 SOFTWARE	BUSINESS PROGRAMMERS	137000	
	BUSINESS ANALYSTS	97000	
	SYSTEM & SCIENTIFIC	66000	
	TOTAL		360,00

*AFIPS, "THE STATE OF THE COMPUTER INDUSTRY IN THE U.S." MONTVALE, N.J. 1973

FIG. 3. SUMMARY OF SOFTWARE/HARDWARE COSTS AND POTENTIAL CONTRIBUTIONS TO COST REDUCTIONS

were integrated, are being pursued as well. One such approach consists of using prefabricated program components oriented to a specific type of business or industry. The components are appropriately parametrized so that they can be adapted to use by many organizations. Typically a potential user selects functional components that are required in his operation. Questionnaires and check-lists are provided to facilitate gathering the user requirements. Also an Editor and a Report Generator are provided to enter user parameters and report formats. Such systems are used by service organizations and computer manufacturers to install small business systems for wholesale and distribution industries[8]. There are shortcomings with this approach on two levels. First there is a continuing requirement for programmer skills for selecting components and determining the values of the parameters. Also frequently, necessary functions cannot be performed by previously prefabricated components and additional special purpose programs are required. On another level, use of such packages requires molding the needs for automation of a business into the structure of the prefabricated software packages. No provisions are made to determine management's critical decisions regarding efficient operation and whether data for decision making is indeed made available to management. This approach can be based on a more general and powerful program generation method described below instead on prefabricated components[9] and will be discussed further in the next section.

Following the bottom-up sequence (Figure 1), automatic program module generation is discussed below first, and is followed by the automatic file

8 Examples are IBM Customizing Service for users of IBM System 3 and the proprietary services of Keydata and others for the distribution industry.

9 A.C. Hax and W.A. Martin, "Automatic Generation of Customized, Model Based Information Systems For Operations Management," Proc. of the Wharton Conference on Research on Computers in Organizations, Philadelphia, Pennsylvania, October 1975, H.L. Morgan, Editor, pp. 117-121.

structure and module selection methodology.

Automatic program module generation: This methodology is based on the use of processors capable of generating broad classes of programs. Figure 4 illustrates this methodology. Box (1) at the bottom of Figure 4 illustrates a broad class of programs, characterized as intended for business applications where they have to process a number of input data [or message] files and to produce a number of output files [or reports]. Box (2) illustrates a processor which can generate the program module for (1). The Program Module Generator (2) consists of two parts, for processing the input formal non-procedural functional specification of the desired program module, and for performing design and code generation.

The design and code generation programs in box (2), embody a mathematical model of a program design process. They check the consistency and the completeness of input specifications by tracing each value in the output data to its sources. From these traces as well as from requirements imposed by file structures of the respective files [e.g. sequential or indexed] they determine the sequencing of the input commands, computation and output commands to attain program module efficiency.

The program module specifications used as input in this process are a subset of the overall problem non-procedural functional specifications. Several formal languages for stating non-procedural functional specifications have been developed and some have been in use[10] (see also ref. 2, 18 and 19).

10 D. Teichrow, "Survey of Languages For Stating Requirements for Computer Based Information Systems," AFIPS Conference Proc., Vol. 42, Fall 1972, pp. 1203-1244.

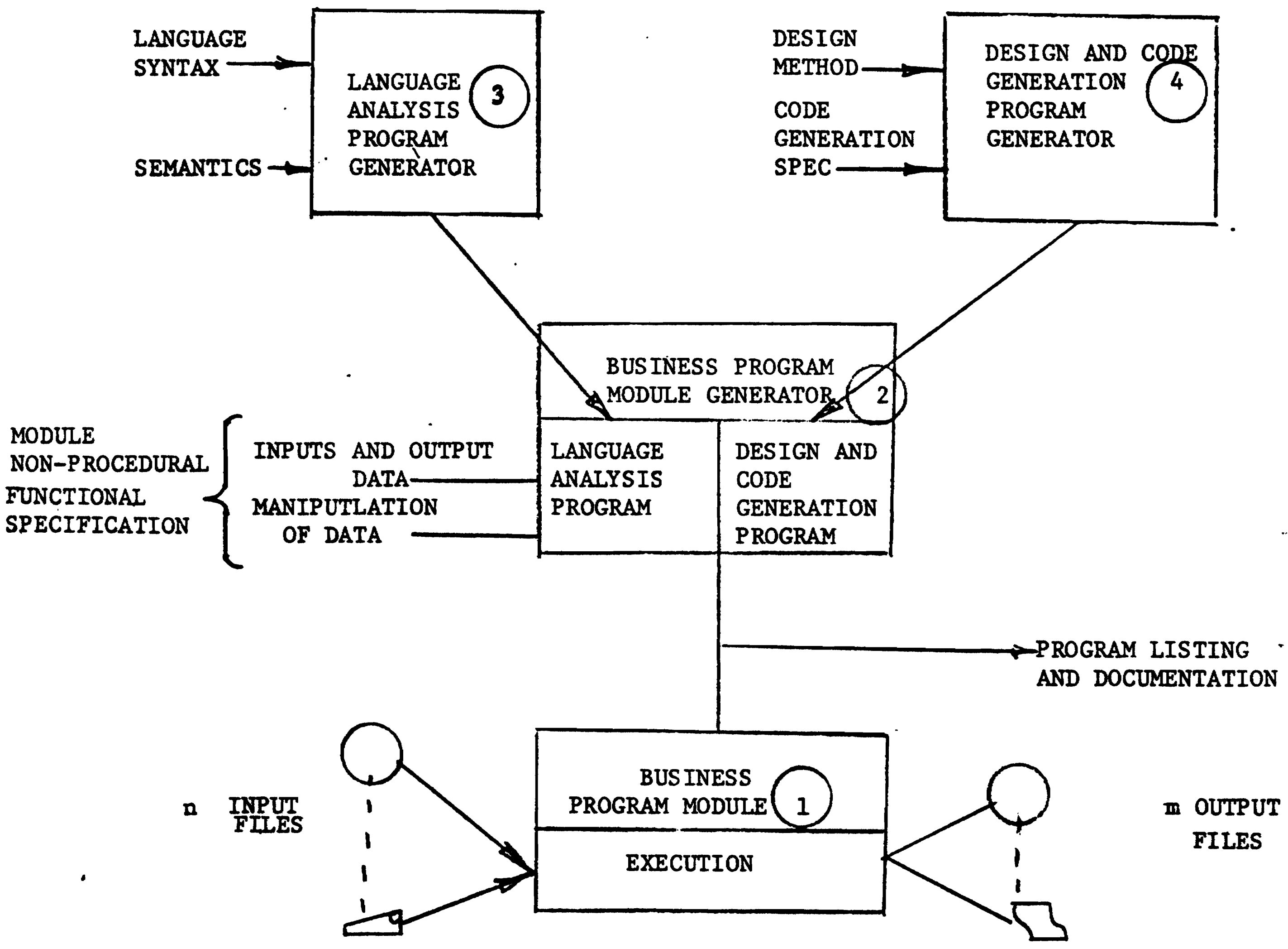

LANGUAGE SYNTAX
SEMANTICS
LANGUAGE ANALYSIS PROGRAM GENERATOR
3
DESIGN METHOD
CODE GENERATION SPEC
DESIGN AND CODE GENERATION PROGRAM GENERATOR
4
BUSINESS PROGRAM MODULE GENERATOR
2
LANGUAGE ANALYSIS PROGRAM
DESIGN AND CODE GENERATION PROGRAM
MODULE NON-PROCEDURAL FUNCTIONAL SPECIFICATION
INPUTS AND OUTPUT DATA
MANIPUTLATION OF DATA
PROGRAM LISTING AND DOCUMENTATION
n INPUT FILES
BUSINESS PROGRAM MODULE
1
EXECUTION
m OUTPUT FILES

FIG. 3 INTERACTIONS IN AUTOMATIC GENERATION OF A PROGRAM MODULE

Because of the emphasis in these languages on facilities to describe data, they are very similar to languages used to describe data bases[11,12]. It is important to be able to process program module specifications that are given in any combination of several forms, such as in a formal language, table format or in a question-answer format. Also, these languages are still in an experimental phase and little usage experience is available for their evaluation. In order to have a capability to accept several selected languages or formats, and to modify them, it is desireable to generate automatically the language analysis programs. This capability is indicated in Figure 4 by a higher level language analysis program generator processor box (3) which automatically generates the module specification analysis programs based on specifications of language syntax and semantics.

The specification of a desired program module that is input to the program module generator comes in <u>two</u> parts which can be related to top and bottom level parts of program code, in a top-down program structure.

11 CODASYL Data Base Task Group Report, Report to the CODASYL Programming Languages Committee, ACM, New York, 1971.

12 N.S. Prywes and D. Pirog Smith, "Organization of Information," in Annual Review of Information Science and Technology, Vol. 7, C.A. Cuadra, Ed., ASIS, Washington, D.C., 1972, pp. 103-158.

The quotation below from H. Mills[13] explains the top-down programming structure:

> "We can begin a process which we can repeat over and over until we get the whole program defined. This process is to formulate a one page skeleton program which represents that hundred page program. We do this by selecting some of the most important lines of code in the original program and then filling in what lies between those lines by names. Each new name will refer to a new segment to be stored in a library and called by a macro facility. In this way, we produce a program segment with something under 50 lines, so that it will fit on one page. This program segment will be a mixture of control statements and macro calls with possibly a few initializing, file, or assignment statements as well."

In a similar approach, two levels are defined below. The top level of the program is defined to consist of all the data definitions, input/output commands and control logic statements. This part constitues also macro level documentation, dispensing with the need for flow-chart documentation. In the bottom-level would be routines that provide the more detailed documentation of data manipulation. These two levels have been identified because they can be related to two parts of the module specification. The top level macro design activity is largely based on the module input-output data specification. The bottom level of the program is based entirely on the data manipulation specification (except where input and output data items, records or files are associated by common names and the direct relationship of input to output is implicit). This relationship is summarized in Table 2.

A system of the type described in Figure 4, with a restriction of only one input and one output file for a program module (n=m=1), has been developed at the University of Pennsylvania by Ramirez[14]. Expansion of the

13 Harlen Mills, "Top down Programming In Large System," Courant Computer Science Symposium I, July 1970. Debugging Techniques In Large Systems, Randall Rustin Ed., Prentice Hall, 1971, pp. 41-55.

14 J. Ramirez, "Automatic Generation of Data Conversion Programs Using A Data Description Language," Ph.D. Dissertation, Univ. of Pennsylvania, 1973.

NON PROCEDURAL INPUT LANGUAGE	PROCEDURAL	
	CODE LEVEL	CODE STATEMENTS GENERATED AUTOMATICALLY
DDL	TOP	DEFINITIONS CONTROL LOGIC I/O
DML	BOTTOM	MANIPULATION-COMPUTATION CONVERSION EVALUATION DDL/DML MODULES

TABLE 2: SUMMARY OF RELATION OF DDL/DML TO CODE IN THE AUTOMATICALLY GENERATED PROGRAM

system for handling multiple input and output files is currently underway. The Ramirez system includes automatic capability for generating language analysis programs based on an extended BNF syntax specification and subroutine calls that express some of the semantics (other semantics are in hand coded code generation programs). A non-procedural specification language is used for input to the program generator. It is similar in structure to that developed by CODASYL[12]. It is also a modified subset of a language developed by Smith[15]. The data manipulation language [DML] used is a subset of PL/1. The generated program module code is in PL/1, requiring a subsequent compilation to produce a load module. The Ramirez system produces also the necessary JCL statements and a variety of documentation, in addition to the PL/1 program module listing.

The design process in the Program Module Generator [box 2 in Figure 4] required a painstaking analysis to specify an acceptable human program design process and to state it in terms of a mathematical model, which has been programmed to constitute a part of the Module Generator. While there exist extensive work on automatically generating language analysis programs[16] there has been only very limited research on automatic production of code generation which is one of the most laborous tasks in constructing program generators. Artificial intelligence research in the area of automatic problem solving[17] appears applicable to the automatic generation of programs to perform design in accordance with specified methods.

15 D. Pirog Smith, "An Approach to Data Description and Conversion," Ph.D. Dissertation, University of Pennsylvania, 1971.

16 W.N. McKeeman, et.al., "A Compiler Generator," Prentice Hall, 1970.

17 For instance, G.S. Sussman and D. McDermott, "Why Conniving is Better Than Planning," AFIPS Conference Proceeding, Fall 1972.

The methodology to generate automatically design and code generation programs is the least developed part of the technology that is necessary to produce program module generators efficiently. Until such technology is developed and applied it is possible only to proceed slowly and laborously by manually producing design and code generation programs. As will be further discussed below, this inability to easily "teach" a computer how to employ a method, that a human can easily learn is an extremely difficult obstacle to surmount on the way toward all-automatic programming.

File Structure and Program Module Definitions: This activity is based on functional specification of the total computer system, consisting of a non-procedural functional specification of the total input and output data and a preliminary determination of the system hardware and software that is required. The outcome of this process are the non-procedural functional specifications of the respective modules. This is a more global software design activity then the module design. Refinements are presently carried out iteratively, where a human designer relies on automatic simulation for evaluation. Recent developments of automation of this are reviewed below[18].

The first step of the process is to analyze the overall system non-procedural functional specifications to determine completeness and consistency. An example of this capability is an analyzer developed for functional specifications expressed in the ADS language[19]. Analysis reports include indices

18 J. Daniel Couger, "Evolution of Business System Analysis Techniques," Computing Surveys, Vol. 5, No. 3, September 1973, pp. 167-198.

19 J.F. Nunmaker, et.al., "A Non Procedural High Level Language For Automated Design of Application Systems," Computer Science Dept., Purdue University, W. Lafayette, Indiana 47906.

See also, J.F. Nunmaker, Jr., "A Methodology for the Design and Optimization of Information Processing System," AFIPS Proc., 1971, SJCC, pp. 283-294.

of data elements, and cross referencing of data manipulation routines and the respective source (input) and target (output) data elements. Groups of data elements that are connected through hierarchial relationship are identified. Next these groups are also cross referenced with the routines that process the data elements in respective groups. A network can be generated where groups of data elements that interact in computations would be connected. Closely connected groups of data element, constitute candidate files. The related processing functions represent candidate program modules. Attempts are then made to consolidate or partition modules and files to increase efficiency. For instance, if two processing functions occur in the same processing cycle and if the preliminarily selected data files overlap greatly in having common data elements, the respective processing functions and files may be consolidated. To make such decisions it is necessary to refer not only to the logical structure of the data and the data manipulation rules but also to the frequencies and cycles of the processing functions. Partitioning of the processing files, as well as use of intermediate files sometimes improve efficiency. For instance, effect of pre or post sorting to order the data may be an important consideration for efficiency. A third type of consideration is to evaluate impact of alternative file organizations. For example, whether the data is accessible sequentially or on a random access basis has impact on efficiency of processing. Such alternatives may be evaluated through simulation[20].

20 A.F. Cardenas, "Evaluation and Selection of File Organization-A Model and System," Comm. ACM 16,9, Sept., 1973, pp. 540-548.

The analysis, synthesis and evaluations of alternatives can be performed piecewise with the aid of automatic simulation and evaluation.[21] The integration of these functions requires human guidance. An all-automatic process will require a data base of global computer design knowledge accessible to a computer. It will be necessary to research first how to formalize such knowledge or how to input it to a computer in a natural language ad-hoc manner. Until such capability is developed, it will be very costly to enter such information through manual modelling and programming. Therefore, the achievement of an all automatic process with no human participation will be necessarily delayed for some time.

5 AUTOMATIC GENERATION OF NON-PROCEDURAL SPECIFICATIONS

The top-level system design activity discussed in this section corresponds to the top three layers in Figure 1. The top layer, named in Figure 1 "automation demand", is concerned with determining what information would management need to evaluate economic and business alternatives and to make decisions for their overall organizational progress and effectiveness. The second and third layers are concerned with developing an automatic system that would collect and make the indicated information available to management. The second layer consists of generating candidate operational concepts and computer configurations and the evaluation of these computer configurations to establish cost/benefits of alternative proposed information systems. The third layer consists of specifying the selected system in a formal manner to be directly applicable in lower layers. Simulation technology for performing such evaluations is highly developed[18,21].

21 J. Yeh and J. Minker, "Key Word In Context Index and Bibliography on Computer Systems Evaluation Techniques, University of Maryland, Computer Science Center, College Park, Maryland, TR-246, June 1973.

Performing the top layer activity in an automatic fashion presents the greatest difficulty. It is the most complex and imaginative part of the total process. It has little to do, if any, with computer methodology, which becomes important only in lower layers. Traditionally this process involves interacting with many participants, with expertise in different disciplines: with top management, staff specialists, operational staff and sometimes with outside the organization such as customers, vendors financial and government organizations. Presumably, a future computer system that could perform this activity automatically would need access to the combined knowledge of the present participants in this process. Current techniques for entering into a computer knowledge on how to evaluate complex economic or business situations consists of coding the knowledge in program form, which requires an enormous amount of manual analysis and structuring. This is the major problem in automating this activity. Two reported approaches to this problem, by Hax and Martin[9] and by Balzer[22] are summarized and reviewed below.

The Hax and Martin approach has already been briefly described previously, in connection with use of prefabricated program components for the operational side of a business. Other features of their approach are: 1) specialize in a relatively narrow application field, reportedly in Inventory Control. 2) Incorporate simulation models for evaluating economics and business questions or operational methods. 3) use artificial intelligence techniques for communicating with the computer interactively in English language for the entry of additionally needed information into the computer.

Two areas of concern in this approach are discussed below. First, whether due to the prefabricated programs feature, the structure may be too confining

22 R.M. Balzer, "A Global View of Automatic Programming," Also memorandum on 'Automatic Programming,' September 1972, USC Information Sciences Institute 4676 Admiralty Way, Marina Del Ray, California, 90291.

and restrictive to be used in situations where technology, or business and economic conditions change rapidly. Second, since the area of application is highly specialized and relatively narrow, the cost of the software generation system development may not be justified by potential utilization opportunities. Both the usefulness and end value of such a generalized Inventory Control system could have been tested, for instance, during the situation of major shortages in essential materials that arose at the end of 1973. Could the system for instance, have generated an inventory control system for oil products that would be effective for conserving oil and for planning utilization of oil products to minimize impact of shortages on the economy? The almost instantaneous availability of such a system would have demonstrated its value as surely being sufficiently great to justify the costs of development.

As indicated previously, existing systems of prefabricated program components have editors and question-answer facilities to collect and enter user requirements. It is not clear how the use of cited artificial intelligence techniques[17,23,24] would materially enhance such facilities.

Balzer[22] proposes the construction of a computer system which will have a generalized learning capability that could be utilized to enter applications related knowledge into the computer. The quotation below summarizes the functions of the computer system in his proposed concept of its operation:

23 T. Winograd, "Understanding Natural Languages," Academic Press, 1972.

24 C. Hewitt, "PLANNER: A Language For Proving Theorems In Robots," Proc. of Intl. Joint Conference on Artificial Intelligence, Mitre Corp., 1969, pp. 245-301.

"1. Problem statement in natural language in terms of the problem domain.
2. Knowledge about the domain acquired interactively in natural language in terms of the complete model of the problem domain.
3. Resulting programs which are optimized with respect to data representations, control structure, and code.

This approach requires significant advances in Artificial Intelligence techniques, in such areas as knowledge representation, inference systems, learning, and problem solving, and in the codification of programming knowledge in the areas of data representations, algorithm selection, and optimization techniques."

Item 1 and 2 in the above quotation refer to the top level design activities addressed in this section. Item 3 refers to program generation and optimization activities similar to those discussed in the previous section. The reference to "domain" is similar to the use of the word "application" above. The starting knowledge in the computer is assumed to be confined to that of computer system analysis and design. As stated, in items 1 and 2, the description of the problem that needs to be solved, together with the knowledge that is needed to solve the problem would be imparted to the computer system through interactive user-computer question-answer sessions conducted in natural English language. The computer would then learn from the user about his business to be able to determine the information needed for business decisions and system requirements. The dependence of this type of activity on artificial intelligence methodology is stated in the second paragraph in the above quotation.

Balzer admits that his proposed system concept is conjectural. There are questions in regard to both effectiveness of the process and feasibility. In regard to effectiveness, the approach implies, for instance, that a president of a company would find it beneficial to teach his problem and his business to a computer that is equipped with only computer oriented knowledge, so that the computer could integrate the two types of knowledge to provide an effective solution to the president's problems. Additionally, some of the information would have to be obtained from the vice-presidents for operations, marketing and finance. Consider the problem of all these participants entering information in a consistent manner so that it all can be integrated. If the artificial intelligence techniques, cited by Balzer, are to be used, the humans interacting with the computer would be required to have knowledge about how the computer acquires knowledge.

A basic assumption of this approach is that interactive communication in English is an effective way to teach knowledge to a computer, although this is a relatively slow process in teaching humans. Note also, that when humans are taught by interactive communication, they already have a basic knowledge of word meanings and relationships, which would not be true for the computer system. This problem is discussed further below. It seems that it would have been less conjectural if it was proposed that the computer could accept text-books as input materials and incorporate the respective knowledge into the system. The text material would than also constitute a base line of the knowledge in the computer on which further knowledge incorporation may be based. Also, such a system could be continuously tested as the information was entered. For instance, at the end of entry of a Chapter from an Introductory Economics textbook, the problems at the end of the Chapter would be submitted as well, and the computer would be required to use the knowledge in the Chapter to produce answers for the problems.

Another problem area is the need to enter into the computers large vocabularies and the inadequacy of present artificial intelligence techniques cited by Balzer for handling voluminous English language communications with the computer. The systems cited by Balzer[17,23,24] to demonstrate that the technology is available, have vocabularies of few hundred words. To communicate applications knowledge would require vocabularies of tens of thousands of words. The problem is not simply in the larger number of words needed for English language communication of application knowledge, but in the amount of work that it takes to enter multiple word meanings and relationships. Development of methodology to tabulate multiple meanings and relationships of words has been reported[25]. It estimates that there are 8000 high usage words which have acquired many meanings and usages. These words, as well as many other words, of lesser usage, will have to be integrated within detailed models of particular applications. To enter all this information, word by word, in an interactive process would require a long time. Other techniques, by which a computer may acquire knowledge of words purely from the usage in text, must be developed to enable rapid entry of an application oriented knowledge base into a computer system. This would eliminate the need to tabulate meaning and relationships in vocabularies. Processes of this type have been considered and developed by workers in the areas of information storage and retrieval, content analysis and text processing[26].

All the questions that have been raised above seem to be open research questions that should be the subject of future research. It appears, therefore, that the adequacy of artificial intelligence techniques for implementing the

25 Louis L. Earl, "Use of Word Government in Resolving Syntactic and Semantics Ambiguities," Conference Proc. Computer Text Processing and Scientific Research, Office of Naval Research, Pasadena, California, 15 March 1973, pp. 55-96.

26 N. Prywes, A. Lang and S. Zagorsky, "A Posteriori Indexing Classification and Retrieval of Textual Data," to be published in Information Storage and Retrieval.

type of system envisaged by Balzer is far from proven and feasibility of such a system will still need to be demonstrated. Therefore, also an effective automation of this generalized high level analysis and design process cannot be forseen with confidence.

6. CONCLUSIONS

Predictions or projections of technological developments are always hazardous, especially when made for a prolonged period such as the period lasting to the year 1985. Therefore, it is in order to comment on reasons for confidence, or lack of, in the projections.

The projections of growth of software manpower requirements are not reliable. They are based on a study of future U.S. Airforce software requirements which have been applied to the total U.S. economy. New industrial uses of computers through 1985, that should figure prominently in such projections, have not been considered. The objective in making the growth projection was for estimating a target for manpower savings from advances in automatic programming. Even if the growth projections are 50% too high, there would still be need for the advances described in the paper.

One assumption made in the paper was that software development will continue in a bottom-up fashion indicated by the layers in Figure 1. The alternative would be some major breakthroughs which would upset the past order of progress. For instance, breakthroughs in the field of artificial intelligence, by which computers could quickly "learn" methods could have such an impact. But as indicated in Section 5, there are important research questions that need to be answered before such breakthroughs can be predicted.

Large and extensive research and development activities in the areas of automatic program generation and systems and files design and evaluation are underway. These efforts are indicated in the references in this paper, and especially in the survey paper[18] and bibliography[21] that have been cited. This, as well as examination of proposed methods, form the main basis for confidence that the automation of program design and implementation would be successfully developed and receive wide industrial and business use by 1985. This will take place in a number of increments as discussed in Section 4.

Finally, the present prospect is that determination of system requirements, leading to a system functional specification would be performed semi-automatically, with business specialists serving as system innovators and integrators, and that they will be aided by various evaluation and language processing systems. The automatic integration of these systems would have to await advances in lower layers of Figure 1 and probably also great advances in artificial intelligence methods.