WHAT IS DETAB-X?

Solomon L. Pollack

October 1962

P-2608-1

WHAT IS DETAB-X?

Solomon L. Pollack*

The RAND Corporation, Santa Monica, California

SUMMARY

The Air Force, like many other large users of computers, continually seeks to reduce its computer programming staffs, costs, and time-lags. Toward this end, it has been exploring the use of computer-independent languages for describing its problems.

This Memorandum describes one of the latest of these languages, DETAB-X (Decision-Tables, Experimental). In an effort to illustrate some of the features of DETAB-X, it is compared with COBOL-61 (Common Business-Oriented Language), using examples of data and procedures written in both languages.

---

*Any views expressed in this paper are those of the author. They should not be interpreted as reflecting the views of The RAND Corporation or the official opinion or policy of any of its governmental or private research sponsors. Papers are reproduced by The RAND Corporation as a courtesy to members of its staff.

This paper was presented at the GUIDE Meeting of the CODASYL Development Committee in Philadelphia, Pennsylvania, November 1-2, 1962.

I. INTRODUCTION

Computer programming for commercial applications consumes large amounts of dollars and manpower. In addition, industry often has to repeat much of the programming when the application changes significantly, or even to process similar applications on different computers. One approach toward reducing programming effort and lag-times is the use of common business languages.

Many people in the business world are already aware of COBOL-61[1] and in fact are beginning to use it in their organizations. The purpose of this Paper is to introduce the reader to DETAB-X.

DETAB-X (Decision Tables, Experimental), developed by the Systems Group of the CODASYL Development Committee, is a language structured with decision-tables based on COBOL-61. Before proceeding with the detailed description of DETAB-X, we will examine the organizational framework within which it was developed and review some background information on business languages.

CODASYL (Conference on Data System Languages) was organized in early 1959. Slide 1 depicts its present organization. Although the Systems Group was originally charged only with the responsibility for the specifications of an improved business language, it has developed both the specifications and structure of the language we call DETAB-X. DETAB-X is not an official language of CODASYL, however; it is experimental. It will be provided on a test basis to those working in business data processing who are willing to experiment with it and provide feedback to the Systems Group on its merits and defects.

Furthermore, DETAB-X does not replace COBOL-61; it augments it. The reason for choosing COBOL-61 was utilitarian: it has already been implemented by several computer manufacturers, and is scheduled to be implemented by a majority of manufacturers by December, 1962. People will therefore be encouraged to test DETAB-X because compilers will be available to process programs written in DETAB-X.

Because decision-rules written in decision-tables are structured differently from the free-form procedure

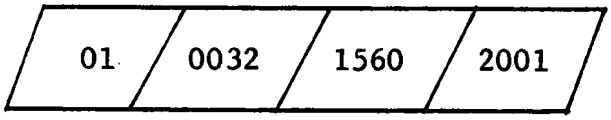
statements of COBOL-61, some modifications to COBOL-61 are required. They will be held to a minimum, however, and will be designed so as to enable a relatively simple preprocessor* to convert the decision-table statements to COBOL-61 statements which can then, in turn, be processed by a COBOL-61 compiler (or processor).

---

*We define a simple preprocessor as a processor that in one or few passes through the source program (in DETAB-X) converts it to a source program in COBOL-61.

II. HISTORY OF THE EVOLUTION OF COMPUTER PROGRAMS

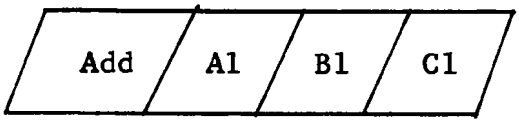
Slide 2 describes the evolution of computer programs. In the very early days the programmer had to write all instructions to the computer in machine language. For example,



might be an instruction to the computer to add (01 is the add operator) the contents of Memory Locations 0032 and 1560 and store the sum in Memory Location 2001. If an application required 2500 machine instructions, the programmer had to write every one of them. He also had to allocate the computer resources, e.g., main memory, drum memory, and magnetic tapes.

Many people realized then (about 1950) that ways were needed to improve the effectiveness of programmers. One way was to introduce subroutines, i.e., sets of instructions common to several programs that could be written once and then adapted for each program. For example, a square root subroutine comprising 15 to 30 machine instructions could be used by any one who needed to compute the square root of a number as part of his program.

To relieve the programmer of assigning storage space for variables, the technique of writing programs in relative (or symbolic) form was developed about 1953-1956. A programmer could write such instructions as:



and provide a dictionary describing where A_1 to A_n, B_1 to B_n, and C_1 to C_n were to be stored in the memory. An assembly routine would then convert these instructions

to machine instructions. This technique, in addition to not requiring the programmer to assign many individual storage locations, enabled him to modify his computer program easily if the dimensions of his variables changed.

The problem of communication between programmers and non-programmers was diminished only slightly by writing in symbolic computer form. A big step forward in this area has been the recent development of symbolic non-computer form languages such as FORTRAN and ALGOL for scientific problems, and AIMACO, COMTRAN, FACT, COBOL, and hopefully, DETAB-X for business problems. Non-programmers can read one of these languages and understand at least half, often all of it.

III. STEPS IN THE DEVELOPMENT OF COMPUTER PROGRAMS

How do we use these languages to get our applications processed by a computer? Slide 3 depicts a source program which describes what we process and how we process. This source program is converted by a translator into the set of machine instructions called an object program, which in turn is processed by the computer to update the files and produce the required reports.

Slide 4 gives a more detailed breakdown of these elements. Here we see that the source program consists of two major parts: the business-oriented problem description and the procedures description. The former describes what needs to be done in the business system; the latter describes how the data processing is to be accomplished. At present, all languages, including DETAB-X, intermix the what and how of the data processing. It is the System Group's hope that DETAB-X will lead to a language that will enable the system designer to describe his application separately from his procedures. Then, if a change occurs, the system designer can implement it by changing the problem description, without having to rewrite the entire computer source program.

In addition, if the problem changes so much that the computer data processing equipment configuration has to be altered, only the procedures description has to be changed.

The programmers and a compiler (a computer program) translate the computer source program to the computer object program. Decision tables can be effective in reducing the number of programmers required to translate the source program by describing characteristics of each decision rule, such as its priority and relative frequency of occurrence. (This point is discussed more fully later when the DETAB-X specifications for decision tables are given.)

IV. DESIRABLE FEATURES FOR PROBLEM-ANALYSIS TECHNIQUES

It will be useful for us to bear in mind the desirable features for a problem-analysis technique (See Slide 5).

Any such technique must first provide an orderly method of documentation. People who have been responsible for computer programs know how devastating it can be when a programmer who has done little or no documentation of his programs leaves the organization. A good technique will make it relatively simple to document a job. DETAB-X can serve this role in business data processing.

A problem-analysis technique that is independent of the processing method enables the statement of the problem to retain its usefulness even if the processing method changes later on. Too often, in the past, the entire application had to be restated when one computer replaced another, only because the processing method was interwoven with the statements of what needed to be done.

When an analysis technique provides the system designer with the flexibility to change portions of his analysis, he has greater freedom of action in documenting the application. For example, if the analysis technique does not imply that actions must be performed in the sequence in which they are written, the designer is free to move from application to application, documenting as he goes along. He can then later determine the order of processing based on parameters he selects.

Most of the data processing systems we are interested in are processed by computers; therefore, a problem-description technique that enables the source program to be readily interpreted by a compiler, with minimum use of programmers, is preferable to one that depends heavily on the services of programmers.

It is highly desirable to have formats that help the analyst visualize complex relationships. In any business system, there are numerous complex relationships among the data in the system. They are extremely difficult to visualize and analyze when they are

in free-form English or quasi-English; but when they are laid out in particular format like decision tables, the system analyst can see and analyze them better.

The failure to consider a particular condition is a continual threat to those responsible for developing the data processing system. An omission in the source program may not be discovered until the computer object program has been operating many months. Consider a stock-inventory system that checks stock balances for reorder points and sufficiency levels, but fails to check to see if the balance has gone negative. This kind of omission can degrade the data processing system to the point that it becomes dangerously ineffective. Therefore, problem description techniques must make it relatively easy to review any system description for omissions and inconsistencies.

V. COMPARISON OF COBOL-61 AND DETAB-X

Let us now turn our attention to COBOL-61, the language DETAB-X is based on. As emphasized above, DETAB-X is an experimental language; that is, its specifications are preliminary, and data processing specialists are invited to experiment with it to provide feedback to the Development Committee System Group on its merits and defects. Furthermore, DETAB-X is not a replacement of official COBOL-61, but rather a particular problem-description technique that uses official COBOL-61 with minor changes. The rules of DETAB-X are prescribed so as to enable source programs written in it to be translated to official COBOL-61 programs by people or by computer programs; we shall call these programs preprocessors.

Slide 6 lists the four divisions of COBOL-61 source programs and what they comprise. DETAB-X source programs will also consist of these four divisions. The specifications for the Identification Division and the Environment Division of DETAB-X will be the same as for official COBOL-61. Those for the Data Division and Procedures Division of DETAB-X differ from those of official COBOL-61, primarily in form. Whereas the Data Division in official COBOL-61 is written in free-form quasi-English, data in DETAB-X are entered into tables with headings. In the Procedures Division of official COBOL-61, all data-transformation rules are written in free-form quasi-English paragraphs and sections. In DETAB-X, those groups of data transformation rules that are based on decisions (or comparisons) are written as entries in decision tables.

To illustrate the difference between COBOL-61 and DETAB-X data description,[2] we show an example of each in Slides 7 and 8. Notice in the COBOL-61 example that we write "Usage is computational"; in DETAB-X we have a header "USE" and the entry for each data item is "C". This convention greatly reduces the amount of writing by the system analyst or programmer.

Also, because in DETAB-X data-description we have the different data attributes (characteristics) in the

heading of each table, we decrease the chances of leaving out a necessary attribute of any data we are describing in our system.

From a communications point of view, the system designer will find the people in the applications area more disposed to helping him check the data description if it is in tabular form as contrasted with the free-form style of COBOL-61. Since the table is not as cluttered as free-form English style, it is much easier to read.

While we have made some improvements in data description for DETAB-X, the big payoff is in the Procedure Division. In this area we use decision tables for describing the many decision rules that exist in business operations. To illustrate the differences between COBOL-61 and DETAB-X in describing procedures, I have extracted an example from Jean Sammet's article on COBOL-61 in the May issue of the ACM Communications and copied it onto Slide 9.

Note that COBOL-61 is serial in nature. The comparisons and the actions based on those comparisons must occur in the order in which they are specified. Note also that this form does not lend itself easily to analysis or to checks for completeness and accuracy. It is difficult to tell whether all the appropriate comparisons on stock-on-hand, current order, and secondary-supply have been made. Also, if a comparison is made against several values, it is very difficult to spot wrong values, because corresponding values appear in different paragraphs, some distance from each other.

Let us turn to Slide 10 which shows these same rules in decision-table form. Notice that having the conditions laid out in tabular form enables the system designer to better determine if he has considered all the possible combinations of conditions that might occur. He knows for example that if there are 3 conditions that can be satisfied or not satisfied, there is a total of 2^3 or 8 different rules he might form.

DETAB-X differs from COBOL-61 in that rules in the table do not have to be executed in the order they have been written, i.e., rule 1 does <u>not</u> have to be executed

first. This gives the compiler freedom to determine the order of rule execution based on some parameter such as frequency of occurrence. For example, if a particular rule is executed 90% of the time while the remaining rules are executed only 10%, is certainly more efficient to have that 90% rule executed first. The format of DETAB-X makes it easy to specify the parameter for each rule so that more efficient object programs can be developed.

When the rules of a table have been specified, the system designer can add a final rule to the table - ELSE GO TO TABLE ___. This rule, by definition, is always referred to last, i.e., if the conditions of each of the other rules have not been satisfied. This feature will prove very valuable to business systems. If after data processing system has become operational, a condition arises that was not anticipated by the system designers, this "else rule" will bring this condition to the attention of the staff. For example, suppose Rule 3 in Slide 10 had been omitted from the table and some time after the system became operational the stock-on-hand was not less than the current-order. Rules 1 and 2 would be tested and found to be unsatisfied. The "else rule" would then be automatically referred to and the computer could print out that specified rules had been satisfied. Thus an incomplete table could be spotted the first time the missing conditions were met.

To further illustrate the difference between COBOL-61 and DETAB-X, Slides 11 and 12 describe the rules for computing depreciation and lease expenses.

The language used in the decision tables of DETAB-X is modified COBOL-61. The deviations of DETAB-X from COBOL-61 (deletions and additions) are described in the DETAB-X Specifications Manual that have been issued by the CODASYL Systems Group. Let me again emphasize that source programs written in DETAB-X (using modified COBOL-61) can readily be translated to standard COBOL-61.

One more point, there is little point to forcing a series of unconditional actions into a decision-table structure. DETAB-X therefore allows portions of the Procedures Division to be written in COBOL-61 sections

and paragraphs. However, where there are decision rules (sets of actions based on sets of conditions), we strongly recommend that decision table structures be used.

In Slide 13 we have listed some desired goals for future business languages. It is our hope that DETAB-X is a step in this direction. We feel that DETAB-X can help users in documenting their system and that programs written in DETAB-X will provide improved communication between system designers, programmers, and functional specialists. DETAB-X is also expected to increase the accuracy and completeness of problem statement achievable by existing languages. It is available to anyone willing to try it and the Development Committee would appreciate receiving any information on the merits and defects of the language.

REFERENCES

1. Department of Defense, COBOL 1961 Report to CODASYL (Conference on Data Systems Languages), Government Printing Office, Washington, D. C., 1961.

2. Pollack, Solomon L., and Kendall Wright, Data Description for DETAB-X (Decision Table, Experimental), The RAND Corporation, Memorandum RM-3010-PR, March, 1962.

3. Grad, Burton, "Tabular Form in Decision Logic," Datamation, July, 1961.

4. Evans, Orren Y., Advanced Analysis Method for Integrated Electronic Data Processing, International Business Machines Corporation General Information Manual #F20-8047.

5. Bromberg, Howard, "COBOL and Compatibility," Datamation, February, 1961, pp. 30-34.

6. Phillips, Charles A., "Current Status of COBOL," Proceedings of the USAF World Wide Data Systems and Statistics Conference, October 2-6, 1961.

CODASYL ORGANIZATION CHART

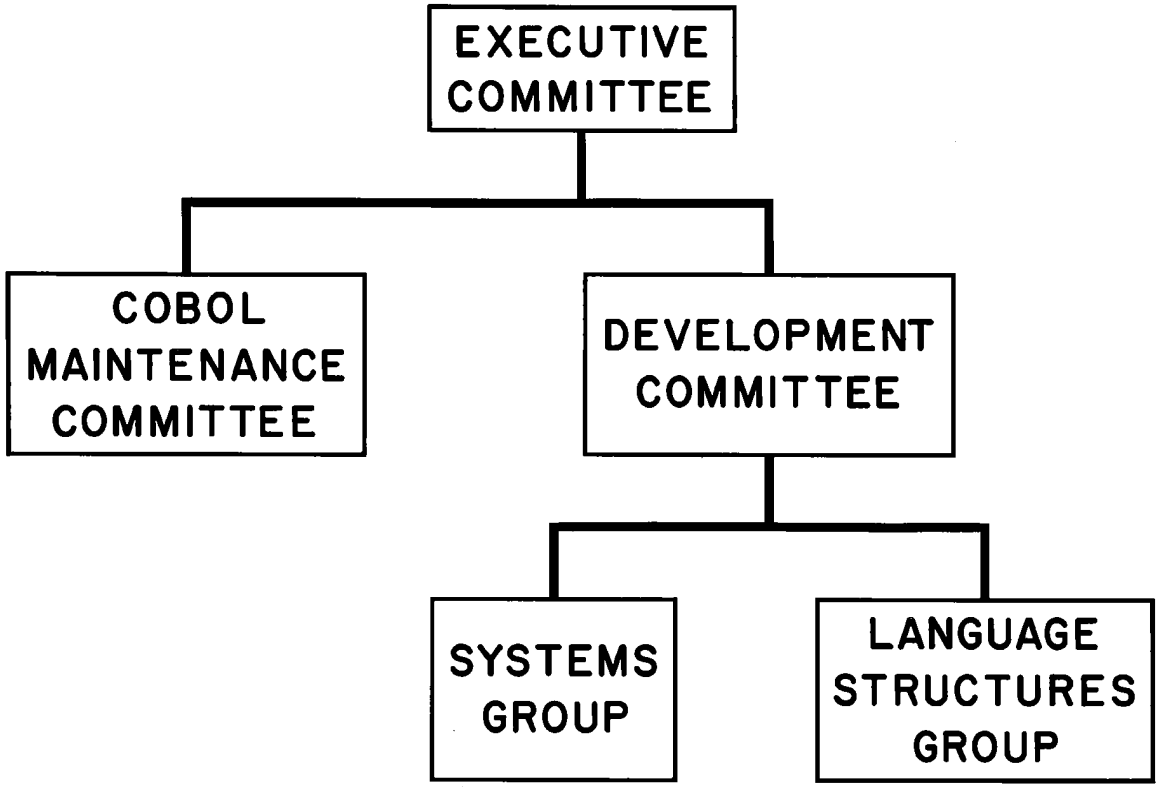


Fig. 1

EVOLUTION OF COMPUTER INSTRUCTIONS

- MACHINE INSTRUCTIONS

01	0032	1560	2001

- SUBROUTINES

  SQUARE ROOT OF X

- SYMBOLIC (ASSEMBLY)-COMPUTER FORM

ADD	A1	B1	C1

- SYMBOLIC (COMPILER)-NON COMPUTER FORM

  SCIENTIFIC - FORTRAN, ALGOL

  BUSINESS - AIMACO, COBOL, DETAB

Fig. 2

DEVELOPMENT OF COMPUTER PROGRAM

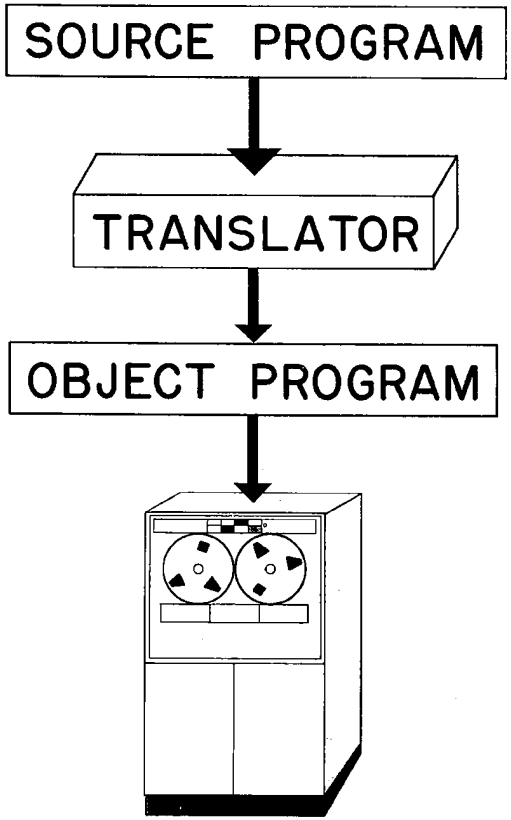


Fig. 3

STAGES IN THE DEVELOPMENT OF A COMPUTER PROGRAM

- BUSINESS-ORIENTED PROBLEM DESCRIPTION
  - DATA DESCRIPTION
  - DATA TRANSFORMATIONS
- PROCEDURES DESCRIPTION
  - EQUIPMENT CONFIGURATION
  - ADDITIONAL DATA TRANSFORMATIONS
- TRANSLATION TO COMPUTER PROGRAM
  - COMPILER
  - PROGRAMMER

Fig. 4

FEATURES OF A PROBLEM-ANALYSIS TECHNIQUE

- ORDERLY METHOD OF DOCUMENTING
- INDEPENDENT OF PROCESSING METHOD
- FLEXIBILITY FOR CHANGING PORTIONS OF THE ANALYSIS
- CAPABLE OF INTERPRETATION BY A COMPILER
- FORMAT AIDS ANALYST IN VISUALIZING NUMEROUS AND COMPLEX RELATIONSHIPS AND ALTERNATIVES
- CROSS-REFERENCES TO ENABLE REVIEW FOR OMISSIONS AND INCONSISTENCIES

Fig. 5

DIVISIONS OF A COBOL SOURCE PROGRAM

- IDENTIFICATION
- ENVIRONMENT
- DATA
- PROCEDURES

Fig. 6

SAMPLE COBOL-61 DATA DESCRIPTION

```
Ø1 INVENTORY RECORD; CLASS IS AN
Ø3 PART-NAME; USAGE IS DISPLAY; PICTURE IS LA(11)
Ø3 ON-HAND-QUANTITY; USAGE IS COMPUTATIONAL;
      SIZE IS 6
Ø3 ON-ORDER-QUANTITY; USAGE IS COMPUTATIONAL;
      PICTURE IS 9(6)
Ø1 SALES-REPORT; CLASS IS AN
Ø2 DISTRICT-SALES; OCCURS 1Ø3 TIMES
Ø3 DISTRICT-NUMBER; USAGE IS DISPLAY; PICTURE IS 999
Ø3 UNIT-SALES; USAGE IS COMPUTATIONAL;
      PICTURE IS 9999999V99
Ø2 TOTAL-SALES; USAGE IS COMPUTATIONAL;
      PICTURE IS 9999999V99
```

Fig. 7

SAMPLE DETAB-X DATA DESCRIPTION

LINE NO.	LEVEL NO.	NAME	ABBR	USE CODE	DESC TYPE	PIC REF OR VALUE	REPETS	
							MIN	MAX
ØØ1	Ø1	INVENTORY-RECORD	INV-REC					
ØØ2	Ø3	PART-NAME		D	P	LA(11)		
ØØ3	Ø3	ON-HAND-QUANTITY	OHQ	C	P	9(6)		
ØØ4	Ø3	ON-ORDER-QUANTITY	OOQ	C	P	9(6)		
ØØ5	Ø1	SALES-REPORT	SAL-REPT					
ØØ6	Ø2	DISTRICT-SALES	DI-SALES				1Ø3	1Ø3
ØØ7	Ø3	DISTRICT-NUMBER	DI-NR	D	P	999		
ØØ8	Ø3	UNIT-SALES		C	P	9999999V99		
ØØ9	Ø2	TOTAL-SALES	TOT-SALE	C	P	9999999V99		

Fig. 8

SAMPLE COBOL-61 PROCEDURE

```
IF STOCK-ON-HAND IS LESS THAN CURRENT-ORDER
THEN IF CURRENT-ORDER IS GREATER THAN SECONDARY-
SUPPLY GO TO EMERGENCY-ORDER-ROUTINE; OTHERWISE
PERFORM SECONDARY-SUPPLY-ROUTINE: OTHERWISE
SUBTRACT CURRENT-ORDER FROM STOCK-ON-HAND.
```

Fig. 9

SAMPLE DETAB-X PROCEDURE

	RULE 1	RULE 2	RULE 3
STOCK-ON-HAND LR CURRENT-ORDER	Y	Y	N
CURRENT-ORDER GR SECONDARY-SUPPLY	Y	N	N
SUBTRACT CURRENT-ORDER FROM SOH	—	—	X
DO SECONDARY-SUPPLY TABLE	—	X	—
GO TO EMERGENCY-ORDER TABLE	X	—	—
GO TO TABLE Ø15	—	X	X

Fig. 10

SAMPLE COBOL-61 PROCEDURE

DEPRECIATION EXPENSE OR LEASE EXPENSE

```
1000  IF ASSET-LEASED GO TO 1050.
1010  IF PROPERTY-CLASS IS LESS THAN "A" GO
          TO ERROR-ROUTINE.
1020  IF PROPERTY-CLASS IS GREATER THAN "J"
          GO TO ERROR-ROUTINE.
1030  IF ASSET-NEW-WHEN-PURCHASED COMPUTE
          SUM-OF-DIGITS-EXPENSE; GO TO 1070.
1040  COMPUTE STRAIGHT-LINE-DEPRECIATION;
          GO TO 1070.
1050  IF GOVERNMENT-COST-FREE-ASSET
          WRITE LOCATION RECORD.
1060  COMPUTE CURRENT-LEASE-AMOUNT.
1070  ADD CURRENT-EXPENSE TO EXPENSE-TO-DATE.
```

Fig. 11

SAMPLE DETAB-X TABLE

DEPRECIATION EXPENSE OR LEASE EXPENSE

	RULE 1	RULE 2	RULE 3	RULE 4	ELSE
ASSET-LEASED	N	N	Y	Y	—
PROPERTY-CLASS ≥ "A"	Y	Y	—	—	—
PROPERTY-CLASS ≤ "J"	Y	Y	—	—	—
ASSET-NEW-WHEN-PURCHASED	Y	N	—	—	—
ASSET-GOVERNMENT-COST-FREE	—	—	N	Y	—
DO	TABLE 5	TABLE 6	TABLE 7	—	TABLE 3Ø
ADD CURRENT-EXPENSE TO EXPENSE-TO-DATE	X	X	X	—	—
WRITE LOCATION-RECORD	—	—	—	X	—

TABLE 5 — COMPUTES CURRENT DEPRECIATION EXPENSE (SUM OF DIGITS) BASED ON PROPERTY CLASS CODE

TABLE 6 — COMPUTES CURRENT DEPRECIATION EXPENSE (STRAIGHT LINE) BASED ON PROPERTY CLASS CODE

TABLE 7 — COMPUTES CURRENT LEASE AMOUNT (SUM OF DIGITS) OVER 5YR

TABLE 30 — WRITES ERROR MESSAGE

Fig. 12

GOALS FOR FUTURE BUSINESS LANGUAGES

1. IMPROVED COMMUNICATION AND DOCUMENTATION
2. INCREASED EFFICIENCY OF COMPUTER PROGRAM
3. REDUCED COMPUTER-PROGRAM CHECKOUT TIME
4. INCREASED ACCURACY IN PROBLEM STATEMENT
5. COMPLETENESS OF PROBLEM STATEMENT

Fig. 13