TITLE: ELECTRONIC FUNDS TRANSFER LANGUAGE

AUTHORS: GERALDINE CHASE
NOSRATOLLAH KHALILI
NORMAN ELDRIDGE

AUTHORS' POSITION: STUDENTS

ORGANIZATION: MISSISSIPPI STATE
COMPUTER SCIENCE DEPARTMENT

This paper discusses a programing language designed as a result of a group term project in an undergraduate Senior level Computer Science course. The course emphasized the study of language concepts such as syntax, semantics, data structures, and language structures rather than language implementation. The task for the project was to design a small special-purpose, application language which would utilize some of the language concepts discussed in the course.

A Business oriented language was chosen as opposed to a scientific language because of the group's background in business related courses and knowledge of business related programming problems. A need was seen for a programming language which could handle the programming problems of banking more efficiently than the general purpose programming languages used by the Banking Industries. The language to be discussed can be used as a general purpose language but has special features designed to solve the various programming problems of Banking. Because of its special features in the Banking area, the name Electronic Funds Transfer Language was chosen. The EFTL structure is Independent Routine structure. The Data structures are arrays and scalars and statements are symbolic in format.

GENERAL INFORMATION

Variable names in EFTL have a maximum length of nine characters and can contain data of flexible lengths. File names and subroutine names have a maximum length of five characters. All labeled statements must have a sequence number in columns one through five in chronological order. Comment statements may be inserted in the source program for reference purposes by use of the REFERENCE function. These comments are to be printed out, but will have no affect on the object program. Its format is:

```
***REF.character string
```

The three asteriks are in columns one through three, REF. is in columns four through seven and the character string starts in column eight and may end in column eighty. In EFTL, instructures fall into two categories, general instructions and special instruction types.

GENERAL INSTRUCTION TYPES

General instruction types are instructions that are not unique to EFTL only. Instructions included in this type are arithmetic instructions, comparison test instructions, transfer instructions and subroutine call instructions.

The Arithmetic instructions include the following operators:

+, addition
*, multiplication
-, subtraction
/, division
**, exponent

The format for the Arithmetic instructions is:
operator,operand,operand,...,operand: variable names

Comparison test instructions include the following operators:

=, equal
, less than
, greater than
=, less than or equal to
=, greater than or equal to

The format for the Comparison test operators is:
operator,identifier,identifier; options

Options include the arithmetic statements and TRANSFER statement. The TRANSFER verb is another general instruction type and has the following format:

TRANSFER sequence #

Its function is to permit a departure from the normal sequence of procedures by specifying a transfer of control to another point in the program.

A subroutine is invoked by the SUB verb, which has the format:

SUB Subroutine name(ARG$_1$, ARG2...)

SPECIAL INSTRUCTION TYPES

The special instruction types in EFTL are instructions which are unique to this language only. The instructions are distinguished from the general instructions by a reserved label name which causes that instruction to be executed according to its form. There are five form types which include FRM, IN, RES OUT and BAL label names which accomplish the special instructions of EFTL.

1	6	9	14	23	27	32	35	36	37 80
Seq. No.	FRM	File Name	variable name	Beg.	End	Dec. Pos.	Row	Col.	Comments

The first form is the FRM which describes an input or output file format. The file names entered here must also be entered on the IN or OUT form. All reserved label names begin in column six. Column thirty-two is requiered for numeric field names only.

1	6	9	14	17 80
Seq. No.	IN	File name	Device Name	OPTIONS

The second form is the IN form which makes available the next record from the specified input file and allows the execution of a specified imperative statement when the end of file is detected. Column seventeen includes the option LAST which transfer control to a sequence number when the last record on the file is read.

1	6	9	14 80
Seq. NO.	RES		CALCULATIONS

The specification form called RES causes one or more numeric data items to assume a new value derived from a named or literal data item or an arithmetic expression. It is used only in calculations concerning mixed operators. The order of priority in operators is: exponents, multiplication, division, addition, and subtraction.

1	6	9	14	17 80
Seq. No.	OUT	File Name	Device name	OPTIONS

The OUT form releases a unit record to an output and allows vertical positioning if the output medium is an on-line printer. If a # appears in column nine, the character string following the # will be printed in the program as a quote.

1	6	9	14	23	32	80
Seq. No.	BAL		Acc. NO.	Acc. NO.	AMOUNT	

An added form which will be used to handle monetary transactions by the computer is the BAL form. This special instruction will greatly reduce the time involved in the clearing of checks and will be useful in implementing a complete turnover of banking procedures to the computer.

SAMPLE PROGRAM

In this program, a master file is being updated. The accounts of persons who have made withdrawals are being adjusted according to amounts read into the computer. The master file is on tape and the transaction file comes in from the card-reader. After the master file has been updated, it will be printed out as the new master file.

Program:

```
10     FRM DEPOS ACCNO       2     11

       *          BALD       14    20

20     FRM CARD ACCN         2     11
       *        BAL          14    20

30     OUT #    This is the updated file

40     IN   CARD C           LAST 120

50     IN   DEPOST           LAST 120

60              =, ACCN, ACCNO, TRANSFER       90

70     OUT DEPOS P

80               TRANSFER    50

90               -, BALD, BAS:  BALD        2

100   OUT DEPOS P

110             TRANSFER     40

120  STOP
```

SUMMARY

The goal of this project was to demonstrate the design of a special purpose language. An Independent Routine structure was chosen for EFTL because it fits the need of this language better than Block structures. Arrays were chosen as the most complex data structure since most structures can be built from arrays.

EFTL applies common language concepts to define a language to serve the needs of a specified user group.