Program your minicomputer in FOCAL.

This conversational language conserves memory and is adaptable to a wide variety of applications.

Small, general-purpose, digital computers are being used increasingly. But what about an easy-to-use computer language for machines with small core memories? FOCAL* (FOrmula CALculator) was developed to meet this need. No programming experience is necessary to understand it, and it permits a minicomputer to perform many operations formerly restricted to a remote terminal on a time-shared computer.

A high-level language for small computers should have two basic design objectives: It must operate effectively within 4000 words of core memory and it should have complete editing capability and interpretive execution. It should also have a high degree of user orientation that is compatible with engineering needs.

Commands described to save memory

It is desirable for clarity that commands (Fig. 1) be spelled out completely to avoid ambiguity. However, abbreviations are often desired for speed and economy of core-memory space, since all text is resident in core. The FOCAL compromise is to choose commands so that each has full mnemonic value, while beginning with a different letter. By ignoring extra letters, the language accepts either abbreviations or expanded forms.

Many of the user-oriented features of FOCAL come from the concepts of RAND Corp's JOSS language. One of these is the line-number, group-number structure, which allows use of a set of lines as a subroutine for repeat computations. It also is possible to execute a single line as a subroutine. These subroutines are even recursive —they permit computation involving successive approximations.

The language has a natural format and readable structure with a minimum of arbitrary formats and few cryptic character codes. Finally, the program is a left-right, top-down interpreter, which makes it easily understood. This, too, makes it adaptable to re-entrant time-sharing.

The diagnostics are flexible

FOCAL has been provided with a large set of precise error diagnostics that describe and locate programming mistakes (Fig. 2). They provide immediate and accurate indication of errors. The error diagnostic is accompanied by the line number in which the error occurs. If further debugging facilities are required, a trace feature is provided that allows the user to pinpoint his error to the offending character. A question mark in the command text causes succeeding letters to be typed until an error or another question mark is encountered.

Most important is the ability to make corrections within seconds. Recognizing that many users will not be skilled typists, the developers of FOCAL have let the user correct single and multiple character errors within a line by using the "modify" command. This is done by giving positioning information, then inserting or deleting at a given spot within a text string.

Another feature is the large number of operations and functions available, (Fig. 1). And there are input/output facilities that include real-time data acquisition and analysis, large array storage and update functions, complex scope control functions, an analog-to-digital converter control and a Calcomp plotter function.

FOCAL also has a wide numerical range and good accuracy. It normally has a seven-digit input and six-digit output with roundoff. There is also a 10-digit version. With a floating point format, there is an exponential range of 10 to the ±616 —greater than that conveniently available on nearly any other computer.

The input/output features provide a great deal of flexibility and utility in creating meaningful output. For example, a print-plot can be made on the teletype with a single interactive statement (Fig. 3):

```
FOR Y=0, 5, 15; TYPE!; FOR X=0,
FSIN (Y)+12; TYPE"*".
```

Histograms, line functions, multiple axes and non-monotonic values can be plotted. Plotting on the Teletype gives qualitative as well as quantita-

* TM Digital Equipment Corp.

Richard Merrill, Senior Programmer, Digital Equipment Corp., Maynard, Mass.

Focal Commands

Command	Abbre-viation	Example of Form	Explanation
ASK	A	ASK X, Y, Z	FOCAL types a colon for each variable; the user types a value to define each variable.
COMMENT	C	COMMENT	If a line begins with the letter C, the remainder of the line will be ignored.
CONTINUE	C	C	Dummy lines.
DO	D	DO 4.1	Execute line 4.1; return to command following DO command.
		DO 4.0 DO ALL	Execute all group 4 lines. Return to command following DO command, or when a RETURN is encountered.
ERASE	E	ERASE ERASE 2.0 ERASE 2.1 ERASE ALL	Erases the symbol table. Erases all group 2 lines. Deletes line 2.1. Deletes all user input.
FOR	F	For i=x,y,z; (commands) FOR i=x,z; (commands)	Where the command following is executed at each new value. x=initial value of i y=value added to i until i is greater than z.
GO	G	GO	Starts indirect program at lowest numbered line number.
GO?	G?	GO?	Starts at lowest numbered line number and traces entire indirect program until another? is encountered, until an error is encountered, or until completion of program.
GOTO	G	GOTO 3.4	Starts indirect program (transfers control to line 3.4) Must have argument.

IF	I	IF (X) Ln, Ln, Ln
		IF (X) Ln, Ln; (commands)
		IF (X) Ln; (commands)
			Where X is a defined identifier, a value, or an expression, followed by one to three line numbers.
			If X is less than zero, control is transferred to the first line number, if X is equal to zero, control is to the second line number.
			If X is greater than zero, control is to the third line number.

LIBRARY CALL	L C	LIBRARY CALL name	Calls stored program from the disk.
LIBRARY DELETE	L D	LIBRARY DELETE name	Removes program from the disk.
LIBRARY LIST	L L	LIBRARY LIST	Types directory of stored program names.
LIBRARY SAVE	L S	LIBRARY SAVE name	Saves program on the disk.
LINK	L	L	For disk monitor system; FOCAL types 4 locations indicating start and end of text area, end of variable list and bottom of push-down list.
LOCATIONS	L	L	For paper-tape system; types same locations as LINK.
MODIFY	M	MODIFY 1.15	Enables editing of any character on line 1.15 (see below).
QUIT	Q	QUIT	Returns control to the user.
RETURN	R	RETURN	Terminates DO subroutines, returning to the original sequence.
SET	S	SET A = 5/B*C;	Defines identifiers in the symbol table.
TYPE	T	TYPE A + B – C;	Evaluates expression and types out = and result in current output format.
		TYPE A – B, C/E;	Computes and types each expression separated by commas.
		TYPE "TEXT STRING"	Types test. May be followed by ! to generate carriage return-line feed, or # to generate carriage return.
WRITE	W	WRITE WRITE ALL	FOCAL types out the entire indirect program.
		WRITE 1.0	FOCAL types out all group 1 lines.
		WRITE 1.1	FOCAL types out line 1.1.

a

Focal's Functions

FSOT( )	Square Root
FABS( )	Absolute Value
FSGN( )	Sign Part of the Expression
FITR( )	Integer Part of the Expression
FRAN( )	A noise Generator (.5 – .9)
FEXP( )	Natural Base to the Power
FSIN( )	Sine
FCOS( )	Cosine
FATN( )	Arctangent
FLOG( )	Naperian Log
FDIS( )	Scope Functions
FADC( )	Analog to Digital Input Function
FNEW( )	User Function
FCOM( )	Storage Function

b

Focal Operations and Their Symbols

Mathematical operators:

↑	Exponentiation
*	Multiplication
/	Division
+	Addition
–	Subtraction

Control Characters:

%	Output format delimiter	
!	Carriage return and line feed	
#	Carriage return	
$	Type symbol table contents	
( )	Parentheses	(mathematics)
[ ]	Square brackets	(mathematics)
< >	Angle brackets	(mathematics)
" "	Quotation marks	(text string)
? ?	Question marks	(trace feature)
*	Asterisk	(high-speed reader input)

Terminators:

SPACE key (names)
RETURN key (lines) (nonprinting)
ALT MODE key (with ASK statement)
Comma (expressions)
Semicolon (compounds and statements)

c

1. **This is the FOCAL language,** which includes commands (a), functions (b) and operations (c). With these instructions, a complete program can be written.

tive information. This is possible because one can cause the carriage to return without a line feed and can give a carriage return/line feed wherever desired.

FOCAL is handy in the laboratory

Among other I/O capabilities are some real-time interactions. Thus FOCAL can be used in the laboratory with devices controlled through an output bus by means of FOCAL functions. Where suitable FOCAL functions do not exist, the user can write his own in machine language and access them via a special FOCAL function that he sets up. This function can be used as any other. It can be imbedded within a set of computations, have any number of arguments and allow complex control functions to be performed in conversational language. The computer system is fast enough to allow interaction with devices that need servicing 10 times a second.

FOCAL is one of the first standard programs for the small computer to run asynchronously. It buffers data to and from the Teletype—a distinct advantage when running FOCAL on slower computers, because it allows data to be fed into the computer at a higher speed than the computational cycle. An enormous decrease in the response time of the program is possible and operation of the keyboard is smoother.

Another advantage of the interrupt system is that it permits the ending of program loops—for example, from the keyboard, by typing "Control-C." The recovery routine will then go into reset mode, type out the message code "?01.00" and return to command mode. Manual restart via the console switches, as well as all error diagnostics, also go to the recovery routine.

Error printing is witheld until prior printing is complete. Otherwise an error message could be printed prematurely, and the result might be misleading when attempting to trace specific errors within a character string.

Data formats are varied

There are several powerful output formats: floating point, fixed point and automatic right-shifting of the decimal point if numbers are larger than the allowed integer field. Any desired format can be specified by giving the total number of digits in the field and the number of digits to be allocated initially to the decimal field. Many input formats are acceptable: leading signs, leading blanks, E format, decimal format. Any reasonable specification of an input number is accepted by the machine. The program can accept alphanumeric strings, which it promptly compresses into a single code number. Thus an interactive program may accept the answers "YES," or "NO," or "MAYBE," etc. Such responses are recognized by comparison with "numbers" that begin with the digit zero—for example: IF (REPLY - 0YES) 2.1. In this example REPLY is a variable name and 0YES is a constant.

The compromise used to overcome the size limitation and still gain power is a sacrifice of speed and compatibility with other languages. The objective was to achieve maximum utility within the space allowed and still have enough user storage left to do fairly complex jobs.

User determines memory configuration

Since even the best intentioned design doesn't satisfy everyone, a good language lets the user establish the limits of the system. When FOCAL is first loaded into core from the tape on which it is supplied, it goes through an initial dialogue, which is actually an interactive FOCAL program whose variables are assessed by machine language. The dialogue requires no extra space, since it occupies the initially "blank" program tape area; it enhances user compatibility by causing changes in the structure of the program itself. In this way FOCAL asks the user whether he wishes to use certain of the extended functions, such as arctangent, logarithm and expo-

Focal's Error Diagnostics

Code	Meaning
?00.00	Manual start given from console.
?01.00	Interrupt from keyboard via CTRL/C.
?01.40	Illegal step or line number used.
?01.78	Group number is too large.
?01.96	Double periods found in a line number.
?01.:5	Line number is too large.
?01.;4	Group zero is an illegal line number.
?02.32	Nonexistent group referenced by 'DO'.
?02.52	Nonexistent line referenced by 'DO'.
?02.79	Storage was filled by push-down-list.
?03.05	Nonexistent line used after 'GOTO' or 'IF'.
?03.28	Illegal command used.
?04.39	Left of " = " in error in 'FOR' or 'SET'.
?04.52	Excess right terminators encountered.
?04.60	Illegal terminator in 'FOR' command.
?04.:3	Missing argument in display command.
?05.48	Bad argument to 'MODIFY'.
?06.06	Illegal use of function or number.
?06.54	Storage is filled by variables.
?07.22	Operator missing in expression or double 'E'.
?07.38	No operator used before parenthesis.
?07.:9	No argument given after function call.
?07.;6	Illegal function name or double operators.
?08.47	Parentheses do not match.
?09.11	Bad argument in 'ERASE'.
?10.:5	Storage was filled by text.
?11.35	Input buffer has overflowed.
?20.34	Logarithm of zero requested.
?23.36	Literal number is too large.
?26.99	Exponent is too large or negative.
?28.73	Division by zero requested.
?30.05	Imaginary square roots required.
?31.<7	Illegal character, unavailable command, or unavailable function used.

2. **Error diagnostics are of great help** in debugging a program, and FOCAL's diagnostics are extensive, in spite of its small core requirements.

nential. Users who do not need these can recover the unused memory space, create larger arrays or use the space as program text storage.

Size limitations place serious burdens on the core layout and on implementation of the program. However, the program is easily expandable to 8000 words. The 4000-word version accommodates a typical program of 20 lines and 40 variables, or about 1000 working cells. A dynamic allocation of resources is supplied in addition to the selection of configuration.

A user is not restricted to 20 lines, 40 variables, a fixed depth of subroutine calls or a fixed depth of nested expressions. The 1000 words of working space can be allocated to whatever purpose is required in a program: a short program with many variables or a long program with few variable assignments.

Punctation conserves memory

A special character can terminate a command string. Thus by using a semicolon to connect two commands, a user saves the data overhead associated with a line pointer, line number and carriage return; a semicolon saves three locations in the text buffer. This convention also fits naturally with the "FOR" command format:

FOR A = b,c,d; . . . This reads: "For the variable A, equal initially to the value of the expression b, incremented by c until A exceeds d, do the command string that follows the semicolon."

The command string can contain other commands and semicolons giving the language a powerful and easily understood iterative statement implemented in minimum core.

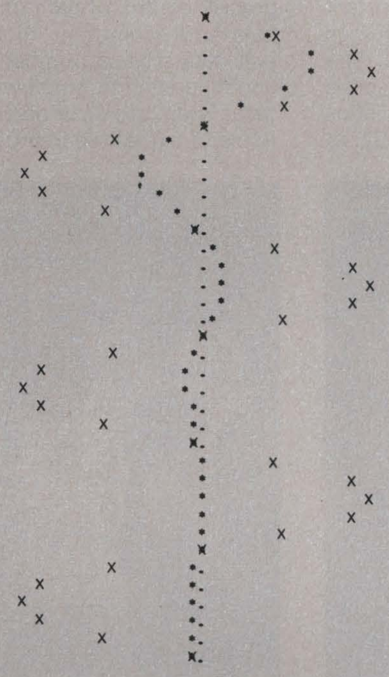
The IF command also was improved by the semicolon. The normal form, similar to that of FORTRAN is:

IF (expression) a, b, c; where a, b, c, are line numbers to go to if the result of the expression in parentheses is negative, zero, or positive, respectively.

The semicolon can shorten the IF command, and execution continues along the same line:

IF (exp) 1.45, 3.2; TYPE "GREATER THAN ZERO."

There is one other unique feature of FOCAL. It is possible to convert the computer operation into a desk calculator mode. Simple arithmetic calculations can be made, and the utility of the computer is enhanced for those who have no need for more elaborate programming. ■■



(a)

```
01.10 ASK "A1",A1,"OMEGA",W,"T0",T0,"DAMPING FACTOR",DAMPINGFACTOR
01.20 ASK "HOW MANY PERIODS DO YOU WANT TO PLOT ?",PERIOD
01.25 ASK "WHAT INCREMENT DO YOU WANT TO PLOT IN ?",INCREMENT
01.30 SET PI=3.14156
01.40 FOR T=0,INCREMENT*PI/180,PERIOD*2*PI;DO 2.0
01.50 QUIT
02.10 SET Y1=A1*FSIN(W*T-T0)
02.20 SET YD=A1*FEXP(-DAMPINGFACTOR*T)*FSIN(W*T-T0)
02.30 FOR Y=0,29;TYPE " "
02.40 TYPE ".",#
02.50 FOR Y=0,29+(Y1);TYPE " "
02.60 TYPE "X",#
02.70 FOR Y=0,29+(YD);TYPE " "
02.80 TYPE "*",!
02.90 RETURN
*GO
A1:20
OMEGA:1
T0:0
DAMPING FACTOR:.25
HOW MANY PERIODS DO YOU WANT TO PLOT ?:3
WHAT INCREMENT DO YOU WANT TO PLOT IN ?:30
```

(b)

3. **This simultaneous print-plot** of a damped and undamped sinusoid (a) results from the program (b) shown, and is a typical example of FOCAL problem-solving. By contrast, FORTRAN requires many more statements for the same results.

Test your retention

Here are questions based on the main points of this article. Their purpose is to help you make sure you have not overlooked any important ideas. You'll find the answers in the article.

1. What are the requirements for a language to be used with small computers?

2. Is asynchronous operation desirable in a conversational language?

3. Why are abbreviations desirable?

4. What is the trace feature?