DECUS PROGRAM LIBRARY

DECUS NO.	8-192
TITLE	T.A.L.C.: Taylor's Algebraic Linear Calculator
AUTHOR	Bruce J. Taylor Submitted by: Theodore Green
COMPANY	The Taft School Waterbury, Connecticut
DATE	May 30, 1969
SOURCE LANGUAGE	PAL III

TALC, TAYLOR'S ALGEBRAIC LINEAR CALCULATOR

DECUS Program Library Write-up DECUS No. 8-192

ABSTRACT

TALC (Taylor's Algebraic Linear Calculator) is a general-purpose calculator designed to evaluate a general algebraic equation, given all quantities involved in the equation. In effect, TALC turns any of the PDP-8 family computers into a powerful desk calculator capable of evaluating complex algebraic, trigonometric, and logarithmic functions. In addition, TALC utilizes the concept of "idiot-proofing" to virtually eliminate the possibility of an operator error invalidating the equation. TALC is easy to use and presents unlimited possibilities in any field where fast, accurate calculations are required.

REQUIREMENTS

Storage

TALC requires a minimum of 4K core storage. It resides in locations $0000-2377_8$ and from $4577-7577_8$.

Subprograms and/or Subroutines

TALC relies upon the Digital 8-5-S (C) Floating Point Package for all arithmetic processing. This auxiliary program is already incorporated in the binary tape of TALC and will not have to be loaded on top of TALC. Therefore, making TALC a complete system, ready for use upon loading.

Equipment

TALC was written on and for a 4K PDP-8/S with a high-speed reader, and a DF32 disk file. However, with the following modifications, TALC may be modified to operate on any PDP-8 with an ASR-33 or ASR-35 Teletype. If the installation does not include a high-speed tape reader, the following changes should be made in core:

ADDRESS	OLD CONTENTS	NEW CONTENTS
2252	6011	6031
2254	6016	6036
2270	6011	6031
2272	6016	6036
1125	6014	6011

If the installation does not include a DF32 disk file, the following changes should be made in core:

ADDRESS	OLD CONTENTS	NEW CONTENTS
0154	7600	0200

USAGE

Loading

TALC is loaded with the Digital-2-U Binary Loader.

Startup and/or Entry

To start TALC, the operator places 0200_8 in the switch register and depresses the following keys in order: "STOP", "LOAD ADDRESS", and "START".

Errors in Usage

TALC has been written with the specific aim of eliminating many of the errors to which even the best operator is prone. For those errors which are unavoidable, the diagnostic procedure has been simplified. If the user requests either an invalid function name or a mathematical contradiction (i.e., the square root of a negative number, division by zero, etc.) TALC will print out a diagnostic message of the following format:

```
ERROR XXXX
```

where "XXXX" represents a four digit number referring to the table of errors.

Recovery from Such Errors

When TALC prints the error message, it enters a wait state and waits for the operator to enter a character through the keyboard. At this point, the operator has two choices. One, if he wishes to cancel further evaluation of the equation, he has only to type in a "Rubout", "Control P", or a "Backarrow" (←). Upon receipt of any of these characters, TALC will return to the input mode and wait for the next equation. If TALC is being executed upon a computer equipped with a DF32 Disk file with a monitor head residing a 7600_8, then the operator may type in a "Control C", and TALC will return control to the resident monitor. If TALC has been properly modified for the absence of a DF32 Disk File, a "Control C" will have the same effect as a "Rubout" or any similar terminating character. If, however, the operator wishes to continue evaluation of the equation despite the error, he has only to enter any character but the four previously mentioned terminating characters. Upon receipt of this character, TALC will correct the error internally.

DESCRIPTION

TALC was written primarily to simplify user programming of the calculator. The format for entering an equation into the user types in upon entering the input mode is a "Control T", TALC will punch a tape of the current equation in a pseudo-binary format. This tape may be stored for later use. When the user wishes to restore the equation to TALC, he places the tape in the high-speed reader and types in a "Control R" as the first character in the input mode. Upon receipt of the "Control R", TALC will read in the tape and commence calculation. This option provides a facility for the storage of often-used equations for fast use.

The main feature of TALC that facilitates use is "idiot-proofing". Under the control of "idiot-proofing", TALC will disregard any invalid character in the equation. This feature does much to eliminate the chance of an operator error invalidating an equation. If the user tries to enter the following invalid equation:

```
     2+/3=;                         /NOTE DOUBLE OPERATORS
```

TALC will accept and echo the equation up to and including the "+". However, when the operator attempts to input the "/", TALC will refuse to echo the character and will, instead, return and wait for the operator to input a valid character (in this case, a numerical quantity, a variable, a unary "+" or "-", or a "(", the beginning of a parenthentical expression). As soon as the operator types a valid character, TALC will proceed with the compilation of the equation as though no error had occured. Therefore, TALC removes the chief cause of equation failure, human error.

TABLE OF FUNCTION NAMES AND ACTIONS

Mnemonic	Value Returned Upon Execution
SQT	Square root of argument
ABS	Absolute value of argument
SIN	Sine of argument in radians
TAN	Tangent of argument in radians
COS	Cosine of argument in radians
LOG	Common logarithm of argument
EXP	e raised to the argument (antilog of argument)
ATN	Arctangent of argument in radians
REM	Final value of last equation*
TFS	-1 if argument is negative 1 if argument is positive Ø if argument is zero

*this function is used as a single numerical quantity.

Examples and/or Applications

The following set of examples has been provided to clarify the points expressed in the preceding discussion. These examples are direct output from a TALC run, with appropriate comments along the right margin.

```
1+2=;                         /BASIC EQUATION FORMAT-NOTE SEMICOLON
+0.3000000E+01                /TERMINATOR


1+2-3*4/5↑6=;                 /EQUATION USING ALL BASIC BINARY OPERATORS
+0.2999231E+01


(1+2)*(3/4-5)=;               /ALTERING OF HIERARCHY WITH PARENTHESES
-0.1274999E+02


SQT 2=;                       /FUNCTION FORMAT-NOTE NO PARENTHESES NEEDED
+0.1414213E+01


SQT(2+2)=;                    /FUNCTION EVALUATION OF COMBINED ARGUMENT
+0.2000000E+01


ABS (5-7)=;
+0.2000000E+01


A+B=;                         /BASIC VARIABLE FORMAT
A: 2                          /VARIABLE CALLING FORMAT
B: 3
+0.5000000E+01

A: 134                        /REPEAT OF SAME EQUATION WITH DIFFERENT DATA
B: 123
+0.2570000E+03

A:←                           /EXIT FROM LOOP VIA "RUBOUT"


1243E23*3=;                   /NOTE INPUT IN SCIENTIFIC NOTATION
+0.3728994E+27


A*A↑2+B*A+C=;                 /POLYNOMIAL EVALUATION
A: 1                          /NOTE THAT VARIABLES ARE CALLED IN ORDER
A: 1                          /OF THEIR APPEARANCE IN EQUATION
B: 2
C: 1
+0.4000000E+01

A: 1
A: 1
B: 0
C: 0
+0.1000000E+01

A:←
```

```
SQT (-1)=;                   /MATHEMATICAL CONTRADICTION CALLED FOR

ERROR 1756                   /ERROR DIAGNOSTIC (SPACE ENTERED AFTER TYPEOUT)
+Ø.1ØØØØØØE+Ø1               /T.A.L.C. CORRECTS [SQT(ABS(-1))=]


SQT (-1)=;

ERROR 1756                   /AT THIS POINT <RUBOUT> TYPED-T.A.L.C.
                             /RETURNS TO INPUT MODE


LOG SQT 1ØØ=;                /NESTED FUNCTIONS
+Ø.1ØØØØØØE+Ø1


(((A+B)*8)=;                 /PARENTHESIS IMBALANCE

ERROR 12Ø4                   /SPACE TYPED-T.A.L.C. BALANCES INTERNALLY
A: 1
B: 2
+Ø.24ØØØØØE+Ø2

A:←
```

METHODS

Discussion

TALC is composed of: equation compiler, interpreter, and the mathematical routines. The last function is performed by the Digital Floating Point Package (See Digital 8-5(C) for further discussion).

The compiler section of TALC is built around the concept of "idiot-proofing", which is based upon two facts that are true for every valid equation and are violated by every invalid equation. The assumptions are as follows:

1. In a valid equation, no two numerical quantities will appear without an intervening binary operator.

2. In a valid equation, no two binary operators will appear without an intervening numerical quantity.

For purposes of this discussion, the following operators will be considered to be binary operators: "↑", "*", "/", "+", ")", "=", and ";". Similarly, the following operators will be assumed to represent unary operators: "(", "+", "-", and functions. TALC implements "idiot-proofing" by relying upon an internal flag, labeled MASTER in the flowcharts and listing, which indicates the nature of the last operator or quantity input. If the last operator was a binary operator, MASTER is found in a positive state. If the preceding quantity was a unary operator or a numerical quantity, MASTER is zero. TALC uses this flag to decide whether the current character should be a unary or binary operator. If MASTER is initially set to zero, TALC will recognize nothing but a binary operator. Upon recognition of the binary operator, TALC resets MASTER to a positive condition. Respectively, if MASTER is initially set to a positive state, TALC will recognize only a unary operator or a numeric quantity. After recognition of this quantity, TALC resets MASTER to the zero state. Therefore, by the interaction of these two states of MASTER, almost all possible equation conditions can be defined for comparison. While these two states define almost all possible conditions, there are three special cases in the equation which require special programming. The first of these is the equal sign ("="). It will be recalled that the equal sign treated as a binary operator and that it may appear between a unary operator and the succeeding binary operator. The equal sign violates the primary tenet of "idiot-proofing" that is, no two binary operators may appear in juxtaposition. This special case may be easily dealt with by the simple expedient of recognizing the equal sign and, when it is input to process it in precisely the same way as a normal binary operator except that MASTER is not reset to one, but is left in the zero state in anticipation of the next binary operator.

The special case in the binary operators is the right parenthesis (")"). In all parenthetical expressions, the parenthesis is treated as a binary operator, since it follows a numerical quantity. However, the ")" is always succeeded by a binary operator, this violates the same rule as the "=" and like the "=", the ")" is treated by recognizing it as a special case and failing to reset MASTER to one.

The third and last special case in "idiot-proofing" is the case of the "REM" function. This function has no arguments, but represents a numeric quantity in itself, and therefore, the proper state of MASTER upon recognition of the REM function should be zero. The normal function recognition routine leaves MASTER in the positive state upon exit from the routine. This case is handled by recognizing REM as a special case and resetting MASTER to zero as in the case of the "=" and the ")". These three special cases with the addition of the two general rules previously mentioned, comprise a general algorithm for the comparison of validity of all equations.

TALC sets up the equation in a special buffer in an internal code when compiling. Numerical input is stored in core in a buffer beginning at location 4000_8 and the addresses are entered in the buffer in lieu of the number itself. All other symbols are given a code number and this code number is entered in the buffer in lieu of the actual character. A list of codes appears at the end of this section. As an example, consider the compilation of the following equation:

(3+5)/SQT 2=;

After the equation has been entered and compiled, the equation buffer will contain the following code:

ADDRESS:	2400	2401	2402	2403	2404	2405	2406	2407	2410	2411
CONTENTS:	1000	4000	0005	4003	0003	0006	0110	4006	0002	0001
SYMBOL:	(	3	+	5	)	/	SQT	2	=	;

The locations 4000_8-4002_8 will contain the binary representation of the number 3, the locations 4003_8-4005_8 will contain the number 5, and finally, the locations 4006_8-4010_8 will contain the number 2. When the equation has been set up in this form, TALC is ready to begin the evaluation.

The method of interpretation used in TALC is from the article "Sequential Formula Translation". TALC sets up two lists in core knows as "cellers". One of these cellars is designed to hold the numerical quantities of the equation and is called "number cellar". The other cellar is to handle the operators of the program and is known as the "operator celler" or the "operator stack".

TALc begins the evaluation by examining the first symbol in the equation buffer. If the character is a numerical quantity (indicated by a value of 4000_8 or greater), the value is placed into the number cellar in the next free location. TALC then returns to examine the next character on the same basis. If the character is not a numeric quantity, but an operator, the procedure becomes more complicated. The internal codes of each operator were chosen to correspond to a universal operator hierarchy which determines the order in which the equation is evaluated (i.e., exponentiation occurs before division, etc.). This assignment allows one to make a general rule for the evaluation: if the operator stack is empty, the current character is stored immediately; if the operator stack is not empty, the current character is compared against the previous operator in the stack. If the new operator has a hierarchial value greater than that of the previous operator in the stack, the new operator is placed in the next sequential position in the stack. If the current operator has a hierarchial value less than the value of the previous operator, the routine indicated by the previous operator is executed. After the execution, the arguments of the old operation are deleted from the number cellar and the results of the execution are put in their place. After all this has been carried out, the current operator is compared against the operator immediately before the one which was just executed and the cycle repeats itself until either the current operator is compared against an operator which has a lower hierarchial value than its own or until the stack is exhausted; in either case, the current operator is immediately stored in the next sequential position in the operator cellar. Using this rule, all nonparenthetical equations may be evaluated quickly and efficiently. For equations with parentheses, the procedure remains precisely the same, but the routines invoked by the parentheses actually manipulate the operator stack. Consider the follwoing equation and its internal representation.

(1+2)=;

```
ADDRESS:    2400 2401 2402 2403 2404 2405 2406
CONTENTS:   1000 4000 0005 4003 0003 0002 0001
SYMBOL:      (    1    +    2    )    =    ;
```

During the evaluation of this equation, the "+" will eventually be compared against the "(" and at this point, the rule for execution will hold and the "(" routine will be executed. The "(" routine consists of a series of commands which store and "+" in exactly the same manner as if it had reached an operator of lower hierarchial value. Therefore, the "+" operator is allowed to trigger no further evaluation and the interpreter returns to pick up the next character from the equation buffer. When the ")" code comes up for examination, the stack will contain only two operators, a "(" followed immediately by a ")". At this point, the "=" code comes up for inspection and follows the rule stated and the ")" routine is executed. Both the "(" routine and the ")" routine are stack manipulators. When the ")" routine is invoked, it deletes the ")" code and the preceding "(" code from the operator stack and returns for further evaluation. The combination of the "(" and ")" routines form the basis of the system for evaluating subquantities: the "(" routine eliminates the possibility of evaluating proceeding beyond the parentheses until the "(" code is deleted by the execution of the ")" routine.

Accuracy

The mathematical functions and accuracy of TALC are dependent upon the Floating Point Package, Digital 8-5-S(C).

ERROR TABLE FOR FUNCTIONS

Diagnostic	Description	Recovery
0414	Invalid function mnemonic called	No recovery possible, return to input mode
1204	Unbalanced parentheses	TALC balances parentheses internally
1730	Logarithm of a negative number	Logarithm of absolute value taken
1737	Logarithm of zero	Returns smallest number representable in TALC
1756	Square root of a negative number	Square root of absolute value taken
2033	Division by zero	Returns largest number representable in TALC
2054	Negative number raised to power	Absolute value of number raised to same power
2301	Checksum error in equation tape	No recovery possible return to input mode

INTERNAL CODES

Operator	Code	Operator	Code
(	1000		0010
unary-	0777	*	0007
REM	0111	/	0006
SQT	0110	+	0005
LOG	0107	-	0004
EXP	0106	)	0003
ATN	0105	=	0002
SIN	0104	;	0001
COS	0103		
TAN	0102		
ABS	0101		
TFS	0100		

FORMAT

Input data

Data input for TALC are: numbers in any form or format the user desires, integer form (8), floating point (8.1), or exponential form (.81E+1). The "E" indicates a factor of 10 and the following number indicates to what power the factor of 10 is raised. Therefore, the number 0.81E+1 is equivalent to 0.81 x 10^1 or 8.1. Any number input into TALC is immediately terminated by a character that is not a digit, decimal point, or an "E". If the user is entering a number and he wishes to cancel just that number, he should enter a "rubout" and then enter the correct number in its place. If during the input of a number the operator wishes to return to the start of the input mode he must enter a "Control P" or a "Backarrow". Twice, once to terminate the current number and once to initiate the return.

Output Data

All output from TALC is in the exponential format:

0.XXXXXXXE±XX

where the X's represent digits. This output format represents the same theory on number representation.

EXECUTION TIME

Maximum

The time required for the completion of an evaluation is a function of the length and complexity of the equation. The only execution time the user need concern himself with is the time needed to input a number. Due to the internal conversion routine of TALC, the program needs more time to input a large number than a small number, particularly a number expressed in exponential format. The user should be sure that TALC is through the input and conversion routines before he attempts to enter the next operator. Normally, this will present no problem since the routine is fast enough to keep up with user's typing. When using large numbers the user should check to see that the JMP and IOT lights are on before typing in the next character.

CORE MAP

Positions	Contents and uses
0000-0177	Local and global variables; links to routines
0200-0577	Main body of Compiler
0600-1177	Subroutines of Compiler
1200-1533	Interpreter
1534-2177	Mathematical Invocation routines
2200-2377	Equation tape Input/Output
2400-2777	Equation buffer
3000-3177	Operator cellar
3200-3377	Number cellar
3400-3777	Unused
4000-4576	Number storage
4577-7577	Floating Point Package