Creative Programming for Young Minds

. . . on the TI-99/4A ™

Volume VI

CREATIVE
PROGRAMMING INCORPORATED
A SUBSIDIARY OF R.V. WEATHERFORD CO.

® A registered trade mark of Texas Instruments, Inc.

CREATIVE Programming for Young Minds

CREATIVE Programming for Young Minds didn't just happen. It represents the harvested fruit of an idea planted several years ago by Dr. Henry A. Taitt. He saw the pressing need for an enrichment program for young children that would help prepare them for the future they would be instrumental in shaping.

It was cultivated by Marilyn Buxton, whose deep interest in early childhood learning enabled her to find ways to teach primary children to program microcomputers.

It was fertilized by Devin Brown, with his lively wit and creative writing style. He gave it the nutrients it needed to appear in printed form to be shared. His shadow is cast over most of the later authors who patterned their style and examples after his original writings.

It was cared for by Howard Smith, Charles Miller, George Kolopanis, Alverta Darding, Lea Ann Hummel, Robin Koch and others, who worked with it in the lab helping to remove the bugs that would stunt its growth.

It was harvested by Nancy Taitt, Marilyn Hoots, Wayne Owens, Diane ZuHone, and others who typed and phoned and talked with people to spread the word and create a market for the final fruit.

And most important of all were the CHILDREN who tried and tested the materials that were produced. They shared their likes and dislikes, and made certain that everything that was included could be done by young minds.

These books were not created by a publisher to be sold to schools, where they would be used on children. They were instead, created from the successes of children, edited by the concerns of parents, and then offered to anyone that wishes to enrich the minds of young children.

If you elect to use these materials, then you assume the responsibility to encourage independent thought, reward creativity, enhance reasoning and logic, and above all, be forever open to alternate ways to solve problems.

If you do this, your own rewards will be found in the faces of the children you serve.

CREATIVE
PROGRAMMING INCORPORATED
A SUBSIDIARY OF R.V. WEATHERFORD CO.

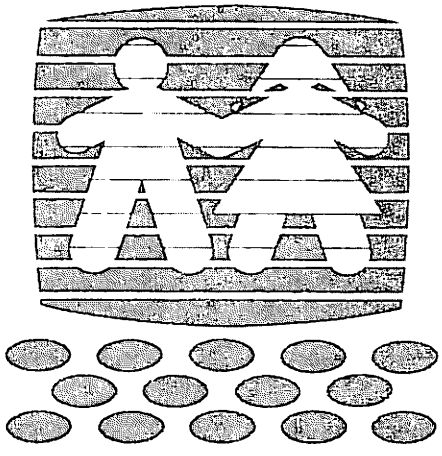
(217) 348-1451

Creative Programming for Young Minds

. . . on the TI99/4A ™

Volume VI

by Leonard Storm



CREATIVE PROGRAMMING FOR YOUNG MINDS

...ON THE TI-99/4A

VOLUME VI

T A B L E O F C O N T E N T S

LESSON #21: STRINGS

The ability of a microcomputer to handle strings of letters, numbers and symbols, gives it great flexibility and power.

In an earlier lesson, you learned that a string constant is a string of characters or spaces enclosed in quotes. For example,

"ABCDEFG", "123", "@#12"

are all valid string constants. You also learned that string constants could not be used in arithmetic operations since they do not have a numerical value. Thus, "1" + "2" makes no sense to the computer, while 1 + 2 does.

There is, however, a TI command that enables you to convert a string constant to a numeric constant. This command is the VAL command which stands for VALUE.

Type in the following program. It illustrates the use of the VAL command.

```
10 CALL CLEAR
20 REM A$ IS A STRING VARIABLE
30 A$="123.5"
40 B=VAL(A$)
50 PRINT "A$=";A$
60 PRINT "B=";B
```

RUN the program.

Statement 40 converts the string constant "123.5" to the numeric constant 123.5. This value is assigned to the variable B.

Notice the differences in the way statements 50 and 60 print onto the screen. Statement 60 leaves a space for the sign of the number between the equal sign and the number, while statement 50 does not leave such a space.

Now add these two lines:

```
70 PRINT "2*B=";2*B
80 PRINT "2*A$=";2*A$
```

Now RUN it. Did you get an error?

Statement 70 shows a valid numeric operation. Statement 80 shows an invalid operation between a numeric constant and a string constant.

Now let's experiment a little further. What happens if you enter the following program? Type NEW first to clear the memory.

```
10 A$="NO NUMBERS HERE"
20 PRINT VAL(A$)
```

Now RUN.

What did the computer do?

Since the string is not a valid representation of a number, the VAL function cannot be used to convert the string to a number.

TI BASIC also offers a reverse procedure. Numbers may be converted into strings. This is accomplished by using the STR$ command.

Type in NEW and enter the following program:

```
10 A=-123.4
20 B$=STR$(A)
30 PRINT "B$= ";B$
40 PRINT "2*A= ";2*A
50 PRINT "2*B$= ";2*B$
```

Now RUN the program.

Statement 20 converts the number represented by the symbol A into a string which is assigned to the string variable, B$. Thus, B$="-123.4" after statement 20 is executed. Statement 50 shows once more that string constants cannot be used in numeric operations.

Try to figure what the following command will do. Then type it into the computer to test your logic.

```
PRINT STR$(VAL("123"))
```

Another string command that is sometimes useful is illustrated in the following program. Type the program into the computer and then RUN it.

```
10 A$=CHR$(72)&CHR$(69)&CHR$(76)&CHR$(76)&CHR$(79)
20 B$=CHR$(77)&CHR$(89)
30 C$=CHR$(78)&CHR$(65)&CHR$(77)&CHR$(69)
```

Keep going. ——————————→

```
40 D$=CHR$(73)&CHR$(83)
50 E$=CHR$(84)&CHR$(69)&CHR$(88)
60 PRINT A$
70 PRINT B$;" ";C$;" ";D$
80 PRINT E$
90 GOTO 90
```

The CHR$ command converts a character code number to its corresponding character. For example, 72 is the character code for the letter H. Thus if Z$=CHR$(72), then Z$="H".

In statement 10 of the program, A$ is set equal to "H" & "E" & "L" & "L" & "O". Therefore, A$="HELLO".

The following program illustrates how easy it is to print out the alphabet by using the CHR$ command. Type the program into the computer and then RUN it.

```
10 FOR I=65 TO 90
20 PRINT CHR$(I);
30 NEXT I
40 GOTO 40
```

By the way, the character codes are also known as ASCII character codes. ASCII is pronounced "ask'-ee". The letters of ASCII stand for American Standard Code for Information Interchange.

Create a program that will display for you the codes from 32 to 127.

The CHR$ command can also be used to display characters you define. This is illustrated by the following program. Type it into the computer and then RUN it.

```
10 CALL CLEAR
20 CALL CHAR(100,"10387CFE7F3E1C08")
30 FOR I=1 TO 768
40 PRINT CHR$(100);
50 NEXT I
60 GOTO 60
```

Create a character of your own.

Now type the following command into the computer:

```
PRINT ASC("A")
```

How did the computer respond?

Notice that the ASC command converts a character to its corresponding ASCII code.

Now type the following command into the computer:

```
PRINT ASC("AB")
```

Notice that the ASC command converts only the <u>first</u> character in the string to its ASCII code. The other characters in the string produce no effect.

The following program converts keyboard characters to their corresponding ASCII codes. Type the program into the computer and RUN it. Input any character you wish.

```
10 CALL CLEAR
20 A$="THE ASCII VALUE OF "
30 PRINT "INPUT A KEYBOARD CHARACTER"
40 INPUT B$
50 PRINT
60 PRINT A$;B$;" IS";ASC(B$);"."
70 GOTO 30
```

Now, suppose you wanted to find the length of a string (number of characters), or you wanted to pick out a portion of the string and use that portion for something, or perhaps you want to create a new string which has the same characters as the old string but in the reverse order.

There are string commands which will let you accomplish these tasks.

The LEN command can be used to find the number of characters in a string. LEN stands for LENgth. Type in the following commands. Write the computer's response on the lines provided.

COMMAND	RESPONSE
PRINT LEN("")	_____
PRINT LEN(" ")	_____
PRINT LEN("1234")	_____

Keep going. ——————→

COMMAND	RESPONSE
PRINT LEN("ABCDEF")	_____
A$="1234567890"	
PRINT LEN(A$)	_____
B$="BBB"	
L=LEN(B$)	
PRINT L	_____
L=LEN(A$&B$)	
PRINT L	_____

As you can see, the LEN command finds the length of the string. Spaces are counted in determining the length of a string. The null string ("") has zero length. Notice that the length of a string may be assigned to a variable:

```
L=LEN("ABC")=3
```

The SEG$ command can be used to obtain a portion of a string. The SEG$ command has three parts as shown below:

```
SEG$(A$,P,L)
```

This SEG$ command will give you a portion of string A$. The portion (or substring) begins at position P in A$ and is L characters long.

For example,

```
B$=SEG$("HOWABOUTTHAT",4,5)
```

would set B$ equal to "ABOUT" since the string segment "ABOUT" begins at position 4 of the string "HOWABOUTTHAT" and is 5 characters long.

Type the following program into the computer and RUN it. Then fill in the table.

```
10 INPUT "INPUT A$,P,L ":A$,P,L
20 B$=SEG$(A$,P,L)
30 PRINT B$::
40 GOTO 10
```

Use the following inputs:

A$	P	L	RESPONSE
HOW ARE YOU	1	5	_____
HOW ARE YOU	3	7	_____
HOW ARE YOU	5	0	_____
HOW ARE YOU	5	15	_____
HOW ARE YOU	13	3	_____
HOW ARE YOU	0	3	_____
HOW ARE YOU	5	-1	_____

The following program shows how the string commands can be used to invert the character order in a string. Type the program into the computer and RUN it. Input any strings you want.

```
5 CALL CLEAR
10 INPUT "INPUT A STRING ":A$
20 L=LEN(A$)
30 FOR I=L TO 1 STEP -1
40 B$=B$&SEG$(A$,I,1)
50 NEXT I
60 PRINT B$
70 B$=""
80 GOTO 10
```

Here's how the program works:

Suppose you input ABC at line statement 10. Then statement 20 will set L equal to 3. Statements 30 through 50 build up the inverse (backwards) string one character at a time.

When statement 40 is first executed, I=3. Therefore, B$ equals SEG$(A$,3,1) which is "C".

When statement 40 is executed a second time, I=2, so that SEG$(A$,2,1)="B". Now B$="CB".

When statement 40 is executed the third time, I=1, SEG$(A$,1,1)="A" and B$="CBA". The string has now been inverted.

The string commands may also be used to convert from one number system to another. Let's see how a number in BINARY can be converted to a decimal number. But first, let's talk about the binary number system.

The binary number system contains only two digits (0 and 1). It is the system used by all digital computers. To understand how the binary system works, let's compare it with the decimal number system which has ten digits.

A decimal number such as 1732 can be rewritten as:

$$(1 \times 1000) + (7 \times 100) + (3 \times 10) + (2 \times 1)$$

That is, each place toward the left has a place value 10 times larger than the preceding place.

In the binary number system, each place toward the left has a place value two times as large as the preceding place.

For example, the binary number 11011 can be rewritten as:

$$(1 \times 16) + (1 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1)$$

Thus 11011 in binary is the same as $(1 \times 16) + (1 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1) = 27$ in decimal.

The following program converts a binary number into its decimal equivalent. Type the program into the computer and RUN it. Input the data listed below the program.

```
5 CALL CLEAR
10 V=0
20 INPUT "INPUT A BINARY #: ":B$
30 L=LEN(B$)
40 FOR I=L TO 1 STEP -1
50 DIGIT$=SEG$(B$,I,1)
60 IF (DIGIT$<>"0")*(DIGIT$<>"1") THEN 10
70 V=V+VAL(DIGIT$)*2 ^ (L-I)
80 NEXT I
90 PRINT "DECIMAL EQUIVALENT=";V
100 PRINT:::
110 GOTO 10
```

BINARY DATA	COMPUTER RESPONSE
0	_____
1	_____
10	_____
11	_____
100	_____
101	_____
11111111	_____
1111111111111111	_____
ABC	_____
123	_____

Here's how the program works:

Statement 30 finds the length of the binary string that you have entered as input in statement 20. Statement 50 looks at one character of the string at a time beginning from the right. Statement 60 checks to see that each character is either 0 or 1 (a valid binary digit). If it isn't, the program jumps back to statement 10 and starts over. Statement 70 computes the decimal number. It adds the current digit to the sum of the previous digits. The value of each digit is calculated by 2^(L-I).

EXERCISE 21-1

Now it's your turn. Create a program that will convert a base 10 (decimal) number into binary. For example, if you input 111, the computer will display a 7.

The last string function in TI BASIC that we will study is the POS (or position) command. This command allows one to find the position of one string within another. The form of the POS function is:

POS(mainstring,substring,n)

The POS function searches for the string called substring in the string called mainstring beginning at the nth position in mainstring. Examples are given below. Type in the following program and RUN it.

```
10 A$="ABCDEFABCDEF"
20 PRINT POS(A$,"ABC",1)
30 PRINT POS(A$,"ABC",2)
40 PRINT POS(A$,"F",9)
50 PRINT POS(A$,"AC",1)
```

Record the computer responses on the line below.

Note that if the POS function cannot find substring in the mainstring, then it returns to a value of zero.

Statement 20 finds that substring "ABC" is located in mainstring A$ beginning at position one. Thus statement 20 prints a one.

In statement 30, the POS function begins searching for "ABC" at position 2 in A$ and therefore misses the first occurence of "ABC". But the POS function locates another "ABC" beginning at position 7 in A$. Therefore statement 30 prints the number 7.

Now type in the following program:

```
10 INPUT "MAINSTRING ":M$
20 INPUT "SUBSTRING ":S$
30 I=0
40 N=1
50 N=POS(M$,S$,N)+1
60 IF N=1 THEN 90
70 I=I+1
80 GOTO 50
90 PRINT S$;" OCCURS ";I;"TIMES IN ";M$
100 PRINT:::
110 GOTO 10
```

Input the following strings when you RUN the program. Record the computer's responses.

M$	S$	RESPONSE
"AB AA"	"A"	_____
"ABBAAB"	"AB"	_____
"AAAAAAA"	"A"	_____
"TUVWZ"	"X"	_____

To see how the program works, let's use the following examples:

M$="ABA" S$="A"

After statement 40 is executed, N=1. Then statement 50 finds the position of "A" in "ABA" beginning at the position N=1. Therefore, the POS command returns a value

Keep going. ——————→

of 1 since the first occurrence of "A" is in the first position of "ABA". Then statement 50 sets N equal to one more than this position or N=2. Statement 70 adds 1 to the variable I, to indicate that there has been an occurrence of "A" in "ABA". Next, statement 80 returns the program to statement 50 to find the next position of "A" in "ABA". This time the POS function would return a value of 3 since the function begins its search for "A" at position 2 and doesn't find "A" until position 3 in string "ABA". Statement 50 then sets N equal to 4 (one more than the POS function value). Next, statement 70 adds another one to I to indicate that two "A"'s have been found in "ABA". Again, statement 80 causes a jump back to statement 50. This time, however, the POS function returns a value of zero since it cannot find an "A" in "ABA" beginning at position N=4. Therefore, statement 50 sets N equal to 1. Since N=1, statement 60 causes a jump to statement 90 to print out the number of occurrences of S$ in M$.

Try to figure out what would happen if statement 50 were:

```
50 N=POS(M$,S$,N)
```

Check your answer by RUNning the program with statement 50 changed.

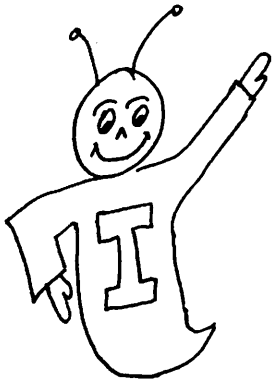
EXERCISE 21-2

Write a program that will take a message in English and convert it to a "secret" encoded message where all characters in the original message are replaced by their ASCII character codes. The program should end by printing the coded message on the screen. Use ASC$, SEG$, and LEN functions in your program. Write your program below.

EXERCISE 21-3

Write a program that will take the ASCII code generated by the previous program (EXERCISE 21-2) and convert it back to English. Store the complete message under a single variable name, say A$. Then have the program end by printing the message A$ on the screen. Write the completed program on the lines below.

LESSON #22: ON GOTO, ON GOSUB



THE ON GOTO STATEMENT ALLOWS A PROGRAM TO JUMP TO ONE OF SEVERAL PROGRAM LINES DEPENDING ON THE VALUE OF THE VARIABLE IN THE ON GOTO STATEMENT.

An example of an ON GOTO statement is shown below:

```
10 CALL CLEAR
20 PRINT "SELECT AN INTEGER FROM 1 TO 4.":
25 INPUT N
30 ON N GOTO 50,60,70,80
40 GOTO 20
50 PRINT "N=1"
55 GOTO 20
60 PRINT "N=2"
65 GOTO 20
70 PRINT "N=3"
75 GOTO 20
80 PRINT "N=4"
85 GOTO 20
```

Type the program into the computer and then RUN it.

Enter the following numbers for N:

N	COMPUTER'S RESPONSE
2	_____
3	_____

Keep going. ——————→

N	COMPUTER'S RESPONSE
4	_____
1	_____
1.1	_____
3.8	_____
5	_____

Whenever a non-integer value of N is entered, the ON GOTO statement first rounds N to an integer. If the value of N is greater than the number of line numbers in the GOTO list, then the program stops running and prints "BAD VALUE IN *line number*."

A variation of ON GOTO is ON GOSUB. The ON GOSUB statement is used when you want the computer to jump to any one of several subroutines based upon the value of a numeric expression in the ON GOSUB statement.

An ON GOSUB statement is illustrated in the next program. Type the program into the computer.

```
5 CALL CLEAR
10 INPUT X
20 ON X GOSUB 40,60,80
30 GOTO 10
40 PRINT "X=1"
50 RETURN
60 PRINT "X=2"
70 RETURN
80 PRINT "X=3"
```

Keep going. ——————→

```
90 RETURN
```

RUN the program. Input the following values for X.

X	COMPUTER'S RESPONSE
1	_____
1.3	_____
1.7	_____
2	_____
3	_____
3.3	_____
4	_____

When statement 20 is executed, the computer first determines the value of the numeric expression X, and rounds it if necessary to obtain an integer. This integer tells the program which line number in the ON GOSUB list to transfer to.

Thus far, anytime information was to be input through the keyboard, the computer had to wait on you. The computer printed a question mark and then waited for you to type in the data and then press ENTER. There are times when it would be great if the computer would not have to stop and wait for the data and a pressed ENTER key. TI BASIC has a subprogram which will allow for this possibility. This subprogram is accessed by using:

CALL KEY(mode, *key*, status)

The value of mode may be 0, 1, 2, 3, 4, or 5.

If mode = 0, then the console keyboard is the input device for the computer, the particular mode the same as was previously specified by CALL KEY.

If mode = 1, then the left side of the keyboard is the input device (or remote control unit 1). If mode = 2, then the right side of the keyboard is the input device (or remote control unit 2.)

Modes 3, 4, and 5 will not be discussed here.

The second part of the CALL KEY command, *key*, will be determined by the key you touch. The computer automatically assigns a value to *key* based upon which key you press. If the console is in mode 0, then one of the normal ASCII codes will be assigned to *key* each time one of the keyboard keys is pressed. For example, if the A key is pressed, then the value assigned to the second part of the CALL KEY command will be 65. If mode 1 or 2 is selected, then the character codes take on values from 0 through 19. (More about this later.)

The status variable is a numeric variable. If a value of 1 is returned for status, that means that a new key was pressed since the last use of the CALL KEY subprogram.

If status = -1, then the same key was pressed during this execution of CALL KEY as was pressed during the last execution of CALL KEY.

If status = 0, then no key was pressed.

Notice that values for key and status are not assigned by you, the programmer, but are assigned by the computer according to what has happened at the input unit (keyboard or remote control).

Now type in the following program which illustrates the use of the CALL KEY subprogram.

```
10 CALL CLEAR
20 CALL KEY(0,KEY,STAT)
30 IF STAT=0 THEN 80
40 IF STAT=-1 THEN 100
50 PRINT "A NEW KEY HAS BEEN PRESSED":"STAT=1"
60 PRINT "KEY=";KEY
70 GOTO 20
80 PRINT "NO KEY WAS PRESSED"
90 GOTO 20
100 PRINT "THE SAME KEY WAS PRESSED"
110 GOTO 20
```

RUN the program. Notice that the program prints "NO KEY WAS PRESSED" over and over.

Now press the A key and hold it down. First the computer should print:

```
A NEW KEY HAS BEEN PRESSED
STAT=1
KEY=65
```

Thereafter the computer will print:

```
THE SAME KEY WAS PRESSED
```

If you now stop holding the A key, the computer will again print:

NO KEY WAS PRESSED

What does KEY equal when you press the following keys:

SYMBOL	KEY=	SYMBOL	KEY=	SYMBOL	KEY=
1		A		N	
2		B		O	
3		C		P	
4		D		Q	
5		E		R	
6		F		S	
7		G		T	
8		H		U	
9		I		V	
0		J		W	
space		K		X	
!		L		Y	
"		M		Z	
#		'		+	
$		(		,	
%		)		-	
&		*			

Now type the following program into the computer:

```
10 CALL CLEAR
20 CALL KEY(0,KEY,STAT)
30 IF STAT=0 THEN 20
40 NOTE=KEY-48
50 ON NOTE GOTO 60,80,100,120,140,160,180,200
60 NOTE=220
70 GOTO 210
80 NOTE=247
90 GOTO 210
100 NOTE=262
110 GOTO 210
120 NOTE=294
130 GOTO 210
140 NOTE=330
150 GOTO 210
160 NOTE=349
170 GOTO 210
180 NOTE=392
190 GOTO 210
200 NOTE=440
210 CALL SOUND(-200,NOTE,0)
220 GOTO 20
```

Finally, RUN the program.

By pressing the number keys from 1 through 8, you can play a tune.

This is how the program works:

If no key is pressed when statement 20 is executed, then STAT will be set to zero. Statement 30 then will cause the computer to loop back to statement 20.

Now, suppose the number 2 key is pressed. Statement 20 will set STAT equal to 1 and KEY equal to 50 (ASCII code for 2). Statement 40 sets NOTE equal to 2. Statement 50 causes a jump to statement 80 which sets NOTE equal to 247. Statement 90 causes a jump to statement 210 which plays the specified note. The computer then jumps back to statement 20 while the note is still being played.

EXERCISE 22-1

Create a program of your own that will play a tune by spelling a word on the keyboard.

Now let's investigate modes 1 and 2. Type the following program into the computer:

```
10 CALL CLEAR
20 CALL KEY(1,A,B)
30 IF B<=0 THEN 20
40 PRINT "A=";A
50 GOTO 20
```

RUN the program. Use it to see what number the computer assigns to A for each key that you press. Record your results below.

KEY PRESSED	A	KEY PRESSED	A
1		A	
2		S	
3		D	
4		F	
5		G	
Q		Z	
W		X	
E		C	
R		V	
T		B	
P		6	
O		7	
I		8	
U		9	
Y		0	

Now change statement 20 to:

```
20 CALL KEY(2,A,B)
```

Press the following keys and record the computer's response.

KEY PRESSED	A	KEY PRESSED	A
6		1	
7		2	
8		3	
9		4	
0		5	
Y		Q	
U		W	
I		E	
O		R	
P		T	
/		;	
H		N	
J		M	
K		,	
L		.	

EXERCISE 22-2

Write a music program which allows 2 notes to be input at the same time so that the computer plays the 2 notes simultaneously. HINT: Use two CALL KEY commands,

```
CALL KEY(1,A,B)
CALL KEY(2,C,D)
```

and use a 2 note CALL SOUND statement.

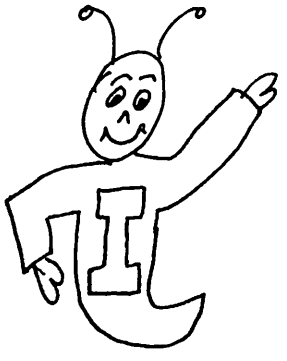
If one unit finds that none of its keys are being pressed, then the program should cause a silent note to be played for that unit.

Write your completed program on the lines below.

EXERCISE 22-3

This time you are to write a program which will use mode 0 of CALL KEY and the ON GOSUB command. Whenever one of the number keys is pressed on the console keyboard, the program should cause that same number to be drawn on the screen such that the number fills up most of the screen. Use HCHAR or VCHAR to draw the number in block form. Whenever a different number is pressed, the previous number should be erased before the new number is drawn.

LESSON #23: DATA STORAGE



THE DATA STATEMENT ALLOWS YOU TO STORE DATA (NUMBERS OR STRINGS) INSIDE YOUR PROGRAM. THE STORED DATA IS ACCESSED BY USING THE READ COMMAND.

The following program illustrates the form and use of DATA statements. Type this program into the computer and RUN it.

```
5 CALL CLEAR
10 PRINT "READ SOME DATA"
20 READ A,B
30 PRINT "A=";A,"B=";B
40 FOR I=1 TO 1000
50 NEXT I
60 DATA 3,9,27
70 GOTO 10
80 DATA 81,243,729
```

Statement 20 causes the numeric variables A and B to be given values from the DATA list beginning with the first DATA list: statement 60 and beginning from the left. Thus A=3 and B=9 are the first assignments made by the READ statement. Note that the positions of the DATA statements within a program are not important, only their relative order.

When statement 20 is next executed, A is set to 27 and B is set to 81, the next numbers (in order) in the data list.

What message does the computer give when the READ statement runs out of data? _____

Type in the following program and RUN it.

```
5 CALL CLEAR
10 DATA 1,A,B,,
20 READ X$
30 PRINT X$
40 READ X$
50 PRINT X$
60 READ Z
70 PRINT Z
80 READ Z$
90 PRINT "Z$=";Z$
```

Why does the program give an error message? _____

_____

Add one character to the appropriate statements to correct the problem. Show the corrections below.

_____

NOTE: TWO ADJACENT COMMAS IN THE DATA LIST REPRESENT THE NULL STRING (NO CHARACTERS IN THE STRING).

The RESTORE command can be used to repsoition the DATA list pointer so that a READ statement can access the same data more than once. This is illustrated in the following program:

```
5 CALL CLEAR
10 DATA 2,4,6,8
20 FOR I=1 TO 6
30 READ A
40 PRINT A;
50 NEXT I
60 RESTORE
70 READ A
80 READ B
90 READ C
100 PRINT A;B;C
110 DATA 10,12,14
```

Type the program into the computer and RUN it.

Statements 20 through 50 cause 6 numbers in the DATA list to be read and printed. Statement 60 repositions the data list pointer to the first element of the first DATA statement. Then, statements 70, 80, and 90 set A=2, B=4, and C=6.

Now make the following program change:

```
60 RESTORE 110
```

RUN the program again. Notice that statement 60 causes the DATA list pointer to be positioned to the first element of statement 10.

Now type the following program into your computer:

```
5 CALL CLEAR
10 DATA 1,1,1,2,1,3,1,4,1,5,0,0
20 DATA 1,3,2,3,3,3,4,3,5,3,6,3,0,0
30 CALL COLOR(2,2,2)
40 ROW=9
50 COL=11
60 GOSUB 160
70 GOSUB 160
80 RESTORE
90 COL=17
100 GOSUB 160
110 GOSUB 160
120 RESTORE
130 ROW=14
140 GOSUB 160
150 GOTO 150
160 READ A,B
170 IF A=0 THEN 200
180 CALL HCHAR(ROW+A-1,COL+B-1,40)
190 GOTO 160
200 RETURN
```

Can you figure out what the program does?

RUN the program to see.

Data may be stored in another way, not in the computer's memory, but on tape by using a cassette recorder. By doing this, one may store data indefinitely.

But how does one go about "filing" away data on cassette tape? In TI BASIC, one has to OPEN a file before the data can be stored. (This is like opening a bank account before one deposits money.)

The form of a typical OPEN statement is shown below:

```
100 OPEN #5:"CS1",INTERNAL,OUTPUT,FIXED
```

The #5 is a file number. When opening a file, any number from 1 to 255 may be used as the file number as long as no other currently open file has that number.

"CS1" tells where the file is to be located. "CS1" stands for cassette #1.

The computer handles file data in one of two formats, either INTERNAL or DISPLAY. If DISPLAY format is specified, the data will be stored in ASCII code. INTERNAL-type data is recorded in a machine language format which is efficiently read by a computer but not by people.

The fourth part of the open statement tells the computer whether data is to be written <u>to</u> the file (OUTPUT) or read <u>from</u> the file (INPUT).

The last part specifies record-type. A record is the group of data transferred between computer and device during one transaction.

FIXED means that the record length is fixed. If the (FIXED) record length is not specified, a record size of 64 characters is assumed by the computer (when a cassette recorder is the device being used).

For cassette tape records, the maximum length that you may specify is 192 characters (FIXED 192). All cassette records must be of FIXED length.

Now let's try to store some numbers on tape. Type the following program into the computer. Then RUN it. Follow the instructions given by the computer.

```
10 CALL CLEAR
20 OPEN #8:"CS1",INTERNAL,OUTPUT,FIXED
30 DATA 1.3,4.906E2,4,15,8213,99,0.1,8,25
40 FOR I=1 TO 3
50 READ A,B,C
60 PRINT #8:A,B,C
70 NEXT I
80 CLOSE #8
```

Statement 50 reads 3 numbers at a time from the data list. Statement 60 then prints these numbers in file #8. Each print causes one record to be written to the file. Therefore three records are written to file #8.

The CLOSE #8 statement in line 80 is the opposite of the OPEN statement. After the CLOSE statement is executed, the file is no longer available to your program.

Now let's retrieve the data from cassette tape. Type the following program into the computer and RUN it.

```
10 CALL CLEAR
20 OPEN #17:"CS1",INTERNAL,INPUT,FIXED
30 FOR I=1 TO 3
40 INPUT #17:A,B,C
50 PRINT A;B;C
60 NEXT I
70 CLOSE #17
```

Notice that the file number need not be the same for INPUT as for OUTPUT. However, the data format must be the same: INTERNAL, in this case. And the record-type must be the same: FIXED, in this case.

Now type this program into the computer.

```
10 CALL CLEAR
20 OPEN #2:"CS1",INTERNAL,INPUT,FIXED
30 INPUT #2: A,B,C
40 INPUT #2: A,B,C,D
50 INPUT #2: A,B
60 CLOSE #2
```

RUN the program. Did you get an error? Do you know why?

Notice that an error results because statement 40 tries to read 4 numeric variables from the second record of the file. However, only 3 variables were written to that record.

When a number is written in INTERNAL format, the computer uses 9 characters to represent the number. Therefore, 9 x 3 or 27 characters were needed per record to represent the numbers that were stored on tape. But we previously stated that FIXED length puts 64 characters per record. The computer automatically "pads" the rest of each record with zeroes.

Now type in the following program. Then RUN it.

```
10 CALL CLEAR
20 OPEN #1: "CS1",INTERNAL,OUTPUT,FIXED
30 PRINT #1: 1,2,3,4,5,6
40 PRINT #1: 1,2,3,4,5,6,7
50 PRINT #1: 1,2,3,4,5,6,7,8
60 CLOSE #1
```

Statement 50 causes an error since 8 numbers would take 9 x 8 or 72 characters for representation. However, a record with FIXED specification allows only 64 characters.

Change statement 20 to:

```
20 OPEN #1:"CS1",INTERNAL,OUTPUT,FIXED 128
```

Then RUN the program again.

This time the program works because the record length of 128 characters is greater than the 72 needed to represent the 8 numbers in statement 50.

EXERCISE 23-1

Write a short program that will read the data back from tape and print it on the screen. Check the program to see that it works properly.

INTERNAL-type string data requires one position in the record for every character in the string plus one position which designates the length of the string. The computer "knows" the length of the string by reading the length indicator which is located at the beginning of the string. For example, "HELLO, MY NAME IS TEX." would require 23 positions in a record.

Now type the following program into your computer.

```
10 CALL CLEAR
20 OPEN #1:"CS1",INTERNAL,OUTPUT,FIXED
30 INPUT "INPUT A$,B":A$,B
40 PRINT #1:A$,B
50 IF (A$="99")*(B=99) THEN 70
60 GOTO 30
70 CLOSE #1
```

RUN the program. Input some data. Try to find the largest string that statement 40 will accept. How many characters are in this string? _____

Remember a number in INTERNAL format takes the same 9 spaces no matter whether it is 0 or 1.234E19. Therefore, a maximum of 64-9-1 spaces is allowed for the string, A$.

To terminate the program and to close the file, type in 99,99 as your last INPUT.

EXERCISE 23-2

Use the program from page 284 to store some data on tape. Then write a program to retrieve the data. Show your working program below.

All DISPLAY-type data is stored in a file in ASCII code with one character in the data taking up one position in the record.

Therefore, numeric data does not gave a fixed length in DISPLAY format as it does in INTERNAL format. For example, the number -2.632E-13 would take 11 positions in DISPLAY format (including a position for sign, decimal point, exponent, trailing space, and digits).

Now enter the following program into your computer:

```
100 CALL CLEAR
110 OPEN #1:"CS1",DISPLAY,OUTPUT,FIXED 64
120 A$="123"
130 B$="QRS"
140 C=123
150 PRINT #1: A$;B$
160 PRINT #1: A$;",";B$
170 PRINT #1: C;C;C
180 PRINT #1: C;",";C;",";C
190 CLOSE #1
200 OPEN #1:"CS1",DISPLAY,INPUT,FIXED
210 INPUT #1: X$
220 PRINT X$
230 INPUT #1: X$,Y$
240 PRINT X$:Y$
250 INPUT #1: X$
260 PRINT X$
```

Keep going.——————————→

```
270 INPUT #1: A,B,C
280 PRINT A:B:C
290 CLOSE #1
```

RUN the program. You should obtain the following output on the screen:

```
123QRS
123
QRS
123  123  123
123
123
123
```

Here's how the program works:

Statement 150 prints the strings A$="123" and B$="QRS" consecutively on tape (123QRS). The semi-colon (;) in statement 150 does <u>not</u> get recorded on tape. Statement 150 has caused one record to be recorded.
Later, statement 210 inputs one string called X$ from the first record of the tape. Notice that X$="123QRS" (as shown in statement 220). Instead of finding two strings in the first record, statement 210 finds only one. This is because there is no separator between the strings "123" and "QRS".

Statement 160 records a second record on tape. This record contains the following:

123,QRS

Since this record contains a separating comma, statement 230 is able to read two different strings from the second record of the tape, namely, "123" and "QRS". Statement 240 prints these two strings on separate lines of the screen.

Statement 170 prints the same number, 123, three times on the third record. The semi-colons specify that the numbers are to be written one right after the other. Statement 250 reads the third record. It detects only a single string in the third record since no separators were recorded in this record. Notice that there are two spaces between each 123. Remember that every number always includes a place for a sign and a trailing space.

If C$="123" had been printed 3 times on the record, no such spaces between digits would have occurred.

Finally, statement 180 records a comma separator between each 123 so that statement 270 is able to discern 3 numeric constants in the fourth record. Statement 280 prints the three numbers on three separate lines.

Now replace the semi-colons in lines 150 through 180 with commas and RUN the program again. Notice the effect of the commas; items are separated into two columns. That is, commas cause additional spaces to be recorded on tape.

EXERCISE 23-3

Write a program that allows you to type text as if you were using a regular typewriter. The text should appear on the screen as it would on a typewritten page. Design the program so that pressing ENTER at the end of each line causes that line to be stored in a one dimensional string array called TEXT$(I), where I is the Ith line of text. Whenever TEXT$(I) is equal to the null string ("") on any input, use an IF-THEN statement to cause the program to jump to a list of options. One option should allow the user to store all of the text on cassette tape using DISPLAY format. Another option should allow the text stored on tape to be retrieved and placed back in array TEXT$(I). Another option should allow more text to be entered into the array from the keyboard starting with any line number I. Try to think of additional options that would make this program very useful. Write your completed program below.

THE COLORED PAGES

At the end of this manual, you will find several colored pages. These are projects that test your ability to use what you have learned. There are no right or wrong answers. If your program does what is asked, then it is quite acceptable. You are free to express your creativity. Be proud of what you do. Do not worry whether your solution is like anyone else's.

Some of these projects may seem easy. . .but do not be deceived into thinking that you can skip them. After all, if they are easy for you, then it will not take long to do them.

Good luck!

Henry A. Taitt

Henry A. Taitt
Director

INDIGO PROJECT 1

Create a graphics program that uses CHR$(x) to draw a skyscraper. Have the values of x stored as DATA.

INDIGO PROJECT 2

Using ON...GOTO or/and ON...GOSUB create a ten question multiple choice test. Include a method for determining the number correct.

INDIGO PROJECT 3

Create a program that will display all of the special characters that may be printed with CHR$(x). You may have to do a little outside reading.

INDIGO PROJECT 4

Create a program that moves a dot around on the screen when you touch the four arrow keys. Use CALL KEY so you won't have to press ENTER.

INDIGO PROJECT 5

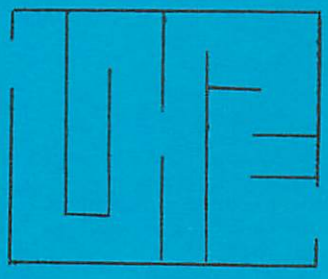
CHR$(x) can be used in a lot of ways. Write a program that will cause the following message to appear in the middle of the screen.

Look, I'm using quotes around "Hello"!

INDIGO PROJECT 6

Using DATA, READ, ON...GOSUB, CHR$ and CALL KEY plus other commands, produce a program that draws the simple maze shown below, and allows you to guide a symbol through the maze using keys of your choice for directions.

Record your solution on tape and send it to us for your Programmer VI card.



Send to:

Henry A. Taitt
CREATIVE Programming Inc.
604 Sixth Street
Charleston, IL 61920

TI-99/4A

Your name _____

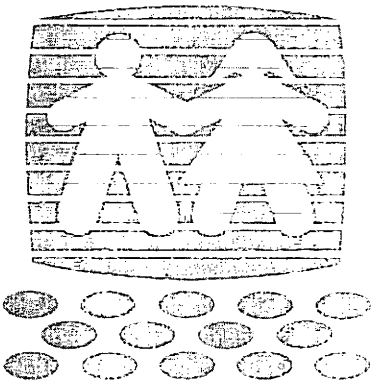
Phone # _____

Address _____

City, State _____

Zip _____ Birthdate _____

Don't forget to enclose a self-addressed stamped envelope.



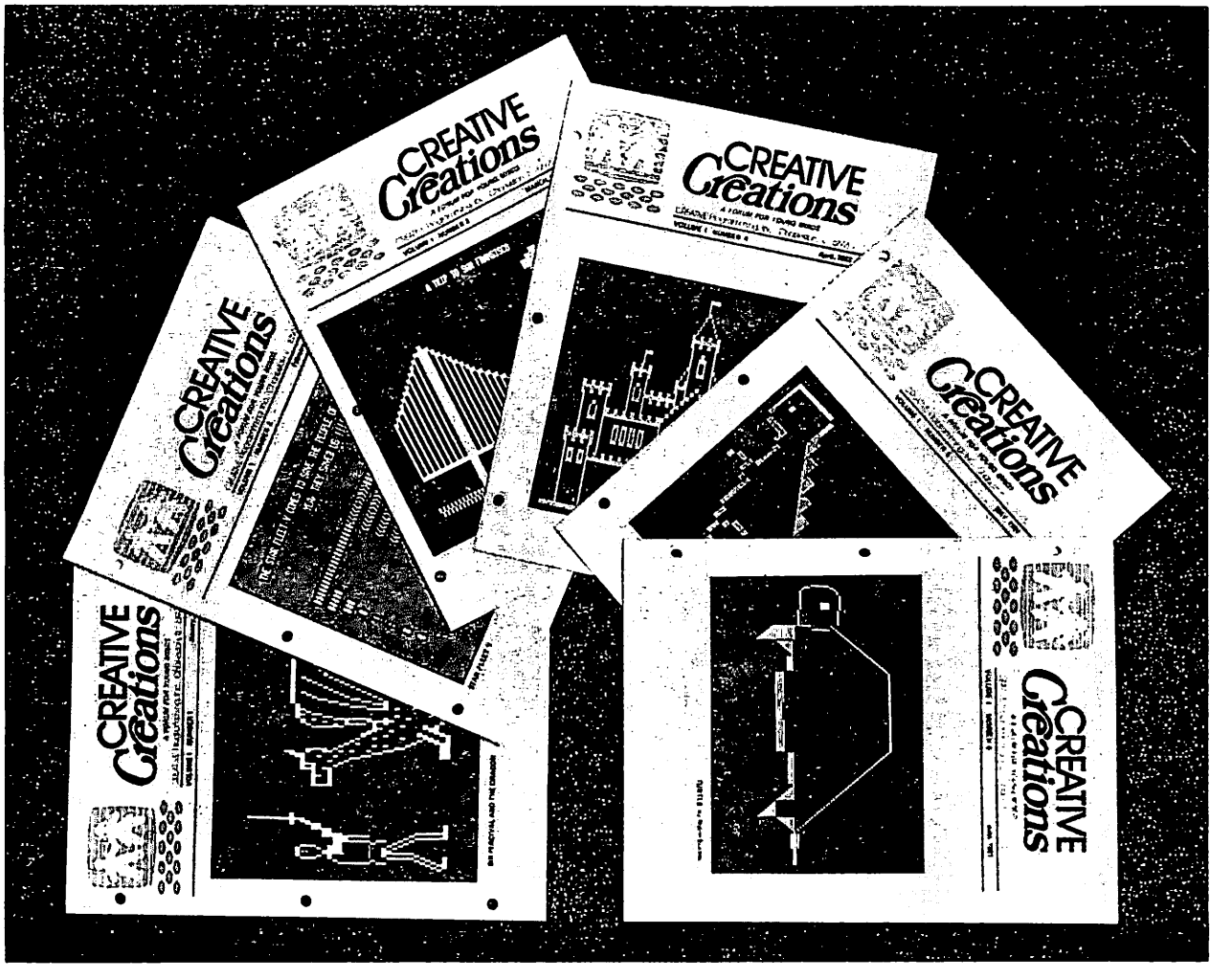
CREATIVE Creations

A FORUM FOR YOUNG MINDS

CREATIVE Programming, Inc., Charleston, IL 61920

A newsletter published 12 times a year. The articles are for young programmers, about young programmers and often written by young programmers.

Each month a graphics program created by a student is selected for the cover. It could be yours! Contests, mind bending challenges, computer game reviews, new creations, programs, even an X-rated column for parents and teachers who are running programs in their areas.



Name_____

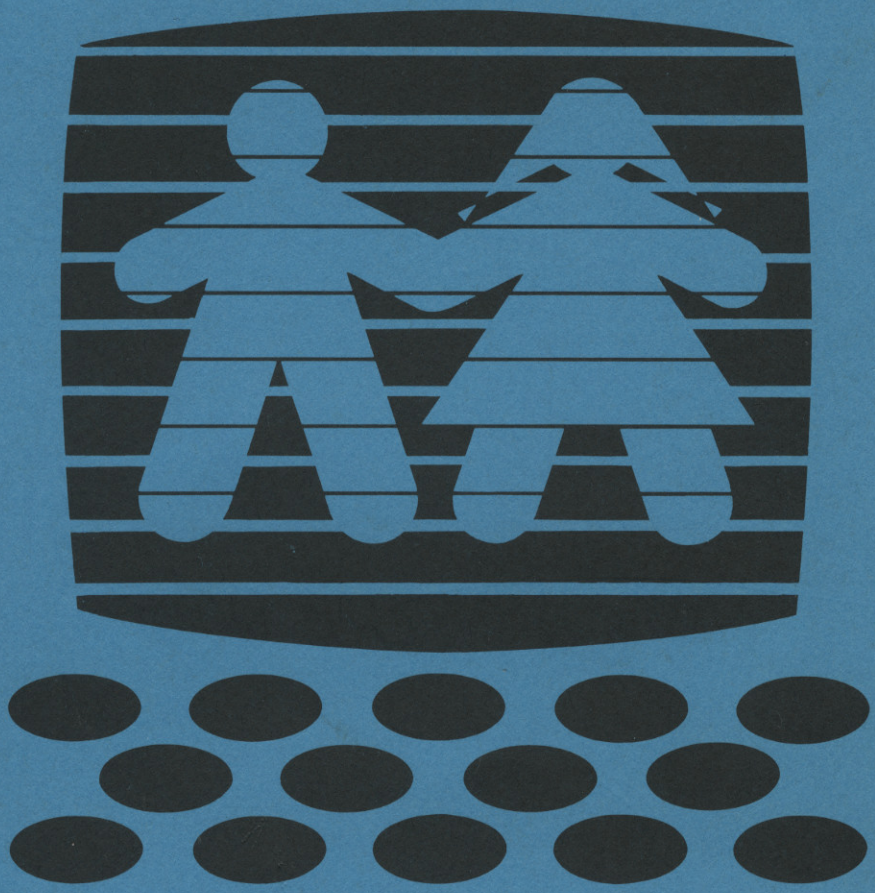
Address _____

City _____ State _____ Zip _____

Please make checks payable to: CREATIVE Creations
604 Sixth Street
Charleston, IL 61920

Only $18 a year ($32 for two years) brings all twelve issues to your door. Join us today in sharing in the excitement of CREATIVE Programming through CREATIVE Creations.

☐ one year ($18.00) ☐ two years ($32.00)



CREATIVE PROGRAMMING INCORPORATED

A SUBSIDIARY OF R.V. WEATHERFORD CO.

604 6th St., Charleston, IL 61920
(217) 348-1451