OPERATING MANUAL

FOR THE

COMPUCORP 445 STATISTICIAN
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>i</td>
</tr>
<tr>
<td>Chapter 1 -- The Granaries of Isis</td>
<td>1</td>
</tr>
<tr>
<td>A general introduction to keyboard operations</td>
<td></td>
</tr>
<tr>
<td>Chapter 2 -- More About Keys And Things</td>
<td>14</td>
</tr>
<tr>
<td>A detailed tour of the keyboard</td>
<td></td>
</tr>
<tr>
<td>Chapter 3 -- Greek Ships And Other Phenomena</td>
<td>91A</td>
</tr>
<tr>
<td>An introduction to programming</td>
<td></td>
</tr>
<tr>
<td>Chapter 4 -- More About Programming</td>
<td>125A</td>
</tr>
<tr>
<td>A sophisticated look at the programmable 445</td>
<td></td>
</tr>
<tr>
<td>Chapter 5 -- Examples And Problems</td>
<td>162</td>
</tr>
<tr>
<td>Examples and problems</td>
<td></td>
</tr>
<tr>
<td>Chapter 6 -- Magnetic Cards And Other Important Things</td>
<td>208</td>
</tr>
<tr>
<td>Magnetic cards; recording and entering programs</td>
<td></td>
</tr>
<tr>
<td>and data; accuracy of the 445; peripheral equipment</td>
<td></td>
</tr>
<tr>
<td>Key Function Glossary</td>
<td>215</td>
</tr>
<tr>
<td>What things do</td>
<td></td>
</tr>
<tr>
<td>Appendix</td>
<td>217</td>
</tr>
<tr>
<td>Functions and codes</td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td>222</td>
</tr>
<tr>
<td>Where things are</td>
<td></td>
</tr>
</tbody>
</table>
INTRODUCTION

Congratulations. You've just bought an extremely versatile piece of calculating equipment. Now all you have to do is learn how to use it. But -- as you're about to find out -- using the Compucorp 445 Statistician isn't much more difficult than using your pencil. (In some cases, it's easier.) And it's certainly a whole lot faster.

The reason it's so easy to use is simply because it was designed from the ground up to do things your way -- rather than the other way around. So if you understand a particular problem, you can probably do it on the 445 without much fooling around with mechanical intricacies. Just enter the information, tell the machine what to do with it and read the answer. Of course, there's a right way to go about doing this. And there are a few little tricks that save a lot of time and eliminate several steps on certain operations. So even though the 445 is set up to follow your normal sequences, it's still a good idea to read this manual first so you can get the most out of your machine.

And getting the most out of your machine means more than just getting fast answers. Because once you've mastered the technique of communicating with the 445 you'll discover another interesting aspect of our little machine. Besides being a lavishly powerful little bag of wires, it's a heck of a lot of fun!

-ii-
CHAPTER ONE

THE GRANARIES OF ISIS
Back in the ancient land of Egypt, the Pharaohs had great granaries built to store surplus wheat during the good years so that they might feed their people during the lean years. One day, the Pharaoh called upon the Keeper of the Granaries and asked him how things were going. He also wanted to know how much wheat was produced during the year, and how much would have to be stored. And he asked him a new question. One that had never occurred to a Pharaoh to ask before. "How much wheat do we usually get, and can we tell beforehand how much wheat we'll have to store so we can build enough grain houses in time to hold it?"

To the first question, the Keeper of the Granaries was able to give a relatively fast answer. He happened to have with him records of grain production for the previous seven years, so he simply took a mathematical average. He did it like this:

While the Keeper of the Granaries was occupied with these calculations, another idea occurred to the Pharaoh. "I notice, O Keeper of the
Granaries, that there is often quite some difference between one year and the next. Is there not some way we can tell how much faith to put in our answer?" The Keeper of the Granaries looked at the Pharaoh with injured eyes, but nonetheless promised to give the Pharaoh an objective measure of the accuracy of his answer. Accordingly, the Keeper of the Granaries then calculated the standard deviation. He did it like this:

On the 445, you'd do it like this:

<table>
<thead>
<tr>
<th>YEAR (now = 0)</th>
<th>AMOUNT OF GRAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>year -1</td>
<td>962,300 hekats</td>
</tr>
<tr>
<td>year -2</td>
<td>1,127,000 hekats</td>
</tr>
<tr>
<td>year -3</td>
<td>627,000 hekats</td>
</tr>
<tr>
<td>year -4</td>
<td>805,300 hekats</td>
</tr>
<tr>
<td>year -5</td>
<td>719,700 hekats</td>
</tr>
<tr>
<td>year -6</td>
<td>1,072,500 hekats</td>
</tr>
<tr>
<td>year -7</td>
<td>984,300 hekats</td>
</tr>
</tbody>
</table>
SET GROUP 1 tells the machine where to store the information.

-clears the areas where information is to be stored.

| 9 6 2 3 0 0 | \[ \sum_{n=1}^{x} x^2 \]
| 1 1 2 7 0 0 0 | \[ \sum_{n=1}^{y} y^2 \]
| 6 2 7 0 0 0 | \[ \sum_{n=1}^{z} z^2 \]
| 8 0 5 3 0 0 | \[ \sum_{n=1}^{a} a^2 \]
| 7 1 9 7 0 0 | \[ \sum_{n=1}^{b} b^2 \]
| 1 0 7 2 5 0 0 | \[ \sum_{n=1}^{c} c^2 \]
| 9 8 4 3 0 0 | \[ \sum_{n=1}^{d} d^2 \]

\[ \text{SD} \] does what it says (prints SD)

\[ \text{MEAN} \] prints mean

\[ \text{2ND FUNC} \] prints mean

\[ \sum 0.00 \]

\[ \sum 962,300.00 \]

\[ \sum 1,127,000.00 \]

\[ \sum 627,000.00 \]

\[ \sum 805,300.00 \]

\[ \sum 719,700.00 \]

\[ \sum 1,072,500.00 \]

\[ \sum 984,300.00 \]

\[ 186,314.11 \]

\[ 899,728.57 \]

On the basis of this information, and considering that the populace consumed around 700,000 hekats of grain each year, the Pharaoh then ordered that 200,000 hekats of grain be stored each year to accommodate the general yearly need.

When he got home, the Keeper of the Granaries began thinking of all
the grain the Pharaoh expected him to store. And he decided that there had to be a better way to calculate the amount of grain needed in any particular year. "There must be a way," he thought, "to know how much grain will be produced each year." So he sat down and studied the records that he had been keeping during all the twenty-three years he had been the Keeper of the Granaries. And he also studied the records his father had kept before him. And he discovered an interesting thing. Every year that the Nile rose very high, there was a lot of grain. And every year that the Nile didn't rise high, there wasn't much grain. The Keeper of the Granaries now had enough information to give the Pharaoh (and himself) a far better way of keeping track of the grain they would need. He wrote down the height of the Nile during each of the previous seven years, and next to each of those numbers he wrote down the amount of grain produced. Then he correlated those numbers. He did it like this:
<table>
<thead>
<tr>
<th>HEIGHT OF NILE</th>
<th>HEKATS OF GRAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.8 cubits</td>
<td>962,300</td>
</tr>
<tr>
<td>5.5 cubits</td>
<td>1,127,000</td>
</tr>
<tr>
<td>3.6 cubits</td>
<td>627,000</td>
</tr>
<tr>
<td>4.0 cubits</td>
<td>805,300</td>
</tr>
<tr>
<td>3.5 cubits</td>
<td>719,700</td>
</tr>
<tr>
<td>5.9 cubits</td>
<td>1,072,500</td>
</tr>
<tr>
<td>4.6 cubits</td>
<td>984,300</td>
</tr>
</tbody>
</table>

\[ \hat{F_n} \quad 0 \] clears areas where data will be stored.

\[ \text{ADV} \] separates CL from data (on tape)

\[ 4 \quad 8 \quad \text{XY} \]

\[ 9 \quad 6 \quad 2 \quad 3 \quad 0 \quad 0 \quad = \]

\[ 5 \quad 5 \quad \text{XY} \]

\[ 1 \quad 1 \quad 2 \quad 7 \quad 0 \quad 0 \quad 0 \quad = \]

\[ 3 \quad 6 \quad \text{XY} \]

\[ 6 \quad 2 \quad 7 \quad 0 \quad 0 \quad 0 \quad = \]

The XY key enters dependent data. The independent variable goes in with the XY key; then the dependent variable goes in with the equals key. The tape skips a space after each pair.
### CHAPTER 1

<table>
<thead>
<tr>
<th>4</th>
<th>XY</th>
<th>4.00</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>XY</td>
<td>3.50</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>XY</td>
<td>5.90</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>XY</td>
<td>4.60</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

**SET GROUP** 1 tells the machine where first group of data is:

**LIN REG** takes correlation coefficient between X and Y:

<table>
<thead>
<tr>
<th>0.94</th>
<th>LR</th>
<th>1</th>
</tr>
</thead>
</table>

**2ND FUNC** gives us the slope:

<table>
<thead>
<tr>
<th>190,121.91</th>
<th>F</th>
<th>2</th>
</tr>
</thead>
</table>

**F** gives us the intercept of X on Y:

| 33,315.83 | F  | 3 |
Therefore, the equation for estimating the amount of grain that could be expected each year would be

\[ \text{Hekats of Grain} = 190,219.91 \times (\text{ht. of Nile}) + 33,315.83 \]

With that information, he went to see the Pharaoh once again. "O great Pharaoh," he said, "what dummies we were. There is a great deal more to knowing how much grain to store than we thought. Let me tell you what I have discovered." And the Keeper of the Granaries then explained to the Pharaoh how whenever the Nile floods a lot they get a lot of grain, and whenever it floods just a little, they get just a little grain. And he showed him the table of numbers which would predict the amount of grain that will be produced for each height the Nile might reach.

"O Keeper of the Granaries," the Pharaoh said, after studying the tables for a very long time, "what a dummy you are. Nowhere in your great table of numbers do you account for the Omens of Isis. How are we to value your table of numbers if you leave out Isis?"

The Keeper of the Granaries, knowing of the great power of the Priest of Isis, promised that he would undertake to consider the Omens of Isis and bring the Pharaoh a new table by which to judge the production of grain. So, grumbling somewhat under his breath, the Keeper of the Granaries gathered up his table and went back to his home. And he sent for the Priest of Isis.
The Keeper obtained from the Priest of Isis a record of the Omens of Isis for the past seven years, and sent him away. Then he tried to correlate these with the data he had already collected. He did it like this:

On the 445, you do it like this:

<table>
<thead>
<tr>
<th>OMENS*</th>
<th>HEIGHT OF NILE (cubits)</th>
<th>GRAIN (hekats)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+4</td>
<td>4.8</td>
<td>962,300</td>
</tr>
<tr>
<td>+2</td>
<td>5.5</td>
<td>1,127,000</td>
</tr>
<tr>
<td>+7</td>
<td>3.6</td>
<td>627,000</td>
</tr>
<tr>
<td>-3</td>
<td>4.0</td>
<td>805,300</td>
</tr>
<tr>
<td>+1</td>
<td>3.5</td>
<td>719,700</td>
</tr>
<tr>
<td>+4</td>
<td>5.9</td>
<td>1,072,500</td>
</tr>
<tr>
<td>+3</td>
<td>4.6</td>
<td>984,300</td>
</tr>
</tbody>
</table>

(*Good agricultural omens minus bad agricultural omens for the six month period preceding the harvesting of the crop.)
clears areas where data will be stored.

separates CL from data

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>9</th>
<th>2</th>
<th>3</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>4.00</td>
<td>962,300.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.80</td>
<td>2.00</td>
<td>5.50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,127,000.00</td>
<td>7.00</td>
<td>627,000.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Three-variable dependent data are entered the same way as two-variable, using the XY key for the independent variables and the equals key for the dependent variable. As before, the tape skips a space after each group of data.
(*Since entry on the 445 is algebraic, the minus sign indicates subtraction rather than a negative value. To make a number negative, you have to use CHANGE SIGN.*)
correlates Nile and Grain, as before.

-0.04 LR 3

correlates Omens and Grain.

Noticing absolutely no correlation whatever between the Omens of Isis and the production of grain, the Keeper simply drew a symbol of the benevolence of Isis on his table and presented it again to the Pharaoh, saying, "O great Pharaoh, I have considered the Omens of Isis, and the table before you does not in any way contradict their portent."

The Pharaoh was very pleased, and ordered that just such an amount of grain as was predicted by the Keeper's table be stored each year according to the height to which the Nile rose, and with the grace of Isis.

After three years, when the Pharaoh noticed that the Keeper's table was, indeed, marvellously accurate, he sent for this Keeper of the Granaries and made him Wizard of the Nile. And so he remained until the end of his days.

If the Keeper of the Granaries had had a 445 (and the same data was in his machine as is presently in ours) he could have predicted the grain production for each year like this:
SET GROUP 4 indicates three-variable data

<table>
<thead>
<tr>
<th>4</th>
<th>9</th>
<th>first independent variable</th>
<th>4.00</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>9 second independent variable</td>
<td>5.10</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dependent variable automatically supplied because of above.</td>
<td>990,630.65</td>
<td>Z</td>
</tr>
</tbody>
</table>

With a machine like this, who knows, maybe he would have become the new Pharaoh.
CHAPTER TWO

MORE ABOUT KEYS AND THINGS
As you may suspect, there's much more to the 445 than you can immediately see on the keyboard. In addition to all the normal arithmetic operations, there's a whole bunch of statistical operations built right in. You can do simultaneous summations for three distinct groups of data. You can enter data into these three groups either singly or in "grouped" form, or as two- or three-variable dependent data. And you can delete data items from each of the three groups. Using the data in each group, you can calculate the mean, standard deviation (either n or n-1 method), standard error of the mean, z-statistic and normal probability associated with the z-statistic. Also, you can calculate independent and dependent t-statistics between any two of the groups. And you can do either two-variable or three-variable linear regression with the ability to "extend" the regression to calculate dependent variables from the independents. And in addition to all this, you can do chi-square goodness-of-fit and permutation/combination calculations.

Besides the capacities of the machine, there are many ways in which operation of the machine can be made much more sophisticated than the techniques we've been using. So before we get into programming -- which in itself is a very powerful story -- let's go over the keyboard in an organized kind of way and consider all the little nooks and crannies in our 445's storehouse of tricks.
CHAPTER 2 -16-

SETTING IT UP

Assuming you've plugged the machine into a power source, the first thing you have to face is the ON-STANDBY-OFF switch at the very left of the top row of buttons. In the ON position, the 445 will be completely operative. All keys will operate, and all programming functions will function. On STANDBY, no keys or programming functions will operate, but the machine will retain in memory any program instructions or data that have been fed into it while it was ON. When you switch the machine from STANDBY to ON, RESET is automatically executed -- which clears all signals which may have been lurking in any of the operation circuits or in the entry register. Then you can just continue using whatever data or program instructions you left in the machine before putting it on STANDBY.

When you put the machine on OFF, all power is cut off and whatever information may have been stored in any of the memory circuits is completely lost. When you switch the machine from OFF to ON, several significant things happen. All registers are cleared (we'll talk about registers a little later). All program memory is filled with NOOP codes. This means that there is nothing --- absolutely nothing --- in program memory. (You'll find out why this is important when we get to programming.) The decimal point is set to two places. Therefore, all numbers will be printed (when they're printed) with two digits to the right of the decimal point. Group 1 is set (more about Set Groups later). And RESET is executed.
(Besides clearing the operation circuits and the entry register, RESET will also clear an ERROR or OVERFLOW condition. If you press \textbf{RESET} in the middle of a multiple-key sequence, the machine will abandon the operation in progress and clear it all out so you can start over from the beginning. We'll talk more about ERROR and OVERFLOW later on.)

Now we have to consider the rest of that top row of switches. For keyboard operation you can ignore most of them. But there are three that have to be checked first. The fourth one from the very right-hand side, the RUN-STEP-LOAD switch, should always be on RUN when you're doing keyboard operations which do not involve programming. And if your machine has a DEGREE/GRAD switch, you have to decide which setting you want. With this switch in the DEGREE position, all trigonometric functions are calculated on a base of the 360° circle. With the switch in the GRAD position, these trigonometric functions are calculated on a base of the 400-grad circle -- with 100 minutes per grad, and 100 seconds per minute. Also, there are several choices to make and things to know about printing.

\textbf{PRINTING}

The 445 printer will print the mantissa of numbers with up to ten digits of significance, the two-digit exponent (when required), minus sign for both exponent and mantissa (when negative) and print symbols indicating the operation performed.

\textbf{Print Format}

When you plugged the machine in, the decimal point was set with two places to its right. From then on, you're on your own. To establish the position of the
decimal point, all you have to do is push \textbf{SET D. P.}, followed by a single digit (0-9). Digits 0-8 indicate the number of digits to the right of the decimal point. Digit 9 tells the machine to print everything exponentially.

Regardless of how you set the decimal point, however, the 445 will do a few convenient things all by itself. Fractional numbers are always printed with at least one digit of significance. If necessary, the decimal point will be shifted to the left automatically until this is achieved. If the decimal point has to be shifted beyond the tenth position in the mantissa to find a digit of significance, then printing will automatically shift to exponential. In exponential mode, the 445 will print the ten most significant mantissa digits, the two exponent digits and a sign for both mantissa and exponent (if negative). Also, if the number is too large to fit to the left of the decimal point as it is set, then the point will be shifted to the right until there are enough places to print the number. If the decimal point must be moved beyond the right-hand margin, then printing will automatically shift to exponential.

Commas will be printed where they belong to the left of the decimal point. And the minus sign of negative numbers will appear next to the left-most digit.

\textbf{Print Control}

With the RUN-STEP-LOAD switch in RUN position, printing is controlled
primarily by the position of the PRINT switch. It can be "ON" or "OFF". With the PRINT switch ON, most keyboard operations will print, providing an audit trail of what you did and how you did it.

There are, however, a few keyboard operations which will not print even with the PRINT switch ON. These are:

- CLEAR ENTRY
- SET DP
- FLAG*
- HALT*
- BRANCH*
- JUMP*
- RESUME*

With the PRINT switch OFF, no keyboard operations will print except a few functions which print in all cases. These "print always" functions are:

- PRINT ENTRY
- PRINT ANSWER
- IDENTIFIER*
- DOT PRINT*
- OVERFLOW
- ERROR

* These are programming operations, and will be covered in a later chapter.
PRINT ENTRY is simply a key that tells the machine to print whatever is in the entry register. It can be used to record selected entries with the PRINT switch OFF. Or, if you haven't entered a new number before pushing PRINT ENTRY, then the 445 will print the last number it has seen -- either the last number entered or the result of the last operation performed. PRINT does the same thing, except that whereas PRINT ENTRY will print the value with the number of places indicated by the decimal point setting without rounding off the final digit, PRINT ANSWER will print the value rounded off to the number of places indicated by the decimal setting. And PRINT will leave the rounded number in the entry register, whereas PRINT ENTRY will leave the full, unaltered original value in the entry register even though it may have caused a shorter version to be printed.

CLEAR ENTRY is a key that you use when you make a mistake while entering a number. Let's say you want to enter 345.67, and by mistake you hit 345.77. If you catch the mistake before it's printed, just push CLEAR ENTRY. The entry register will be cleared, and you can enter the number all over again. CLEAR ENTRY clears only the number being entered, it will not affect an operation in progress. CLEAR ENTRY can also be used to clear an ERROR or OVERFLOW condition.

ERROR condition results from an attempt to engage the 445 in an illegal operation. When that happens, "ERROR" will be printed, regardless of
the PRINT switch position. The calculator operation will halt. The keyboard will become inoperative. And the IDLE light will flash. (The IDLE light is that rectangular little light on the upper right-hand side.) Either RESET or CLEAR ENTRY will clear the ERROR condition. These are the illegal operations which cause ERROR:

- Divide by zero
- \(1/x\) of zero
- \(\sqrt{\text{a negative number}}\)
- Log of zero or negative number
- Entry of more than thirteen digits (or a decimal point followed by more than twelve digits)

- Calculating \(a^x\) with the \(a^x\) key

- Calculating \(0^0\) with the \(a^x\) key

- Calculating standard deviation \((n-1)\) with \(n \leq 1\)

- Calculating standard deviation \((n)\) with \(n \leq 0\)

- Factorial of numbers less than zero

- Factorial of non-integer numbers

- Pressing two or more keys simultaneously

- Exceeding the two-level key buffer*

- Calculating linear regression with \(n \leq 1\)

- Calculating \(z\) with \(n \leq 1\)

*See ENTERING NUMBERS, below.
OVERFLOW condition is caused by attempting to operate with numbers outside the range of the calculator. (The range of the 445 is $10^{-98}$ to $10^{+98}$.) When this happens, "OVERFLOW" is printed, regardless of the PRINT switch position. Calculator operation halts. The keyboard becomes inoperative. And the IDLE light flashes. RESET or CLEAR ENTRY will clear the OVERFLOW condition.

(The IDLE light will remain off while the machine is calculating. In most cases, you don't have to pay much attention to it, unless it's flashing. When it flashes, this means the machine has gone into either ERROR or OVERFLOW mode, and you have to clear it before you can do anything else.)

**Paper Advance**

If you want to advance the paper without printing, just touch the ADVANCE key next to the paper bail. Pressing will advance the paper one space. Holding the key down will cause repeat spacing until you let up.

OK. So we've plugged the machine in, turned it on and set PRINT to ON. Now what?

**ENTERING NUMBERS**

Entering numbers on the 445 is a simple matter of using the 0-9 numeral keys in the middle of the keyboard and the decimal point (plus the
CHAPTER 2 -23-

CHANGE SIGN and EXPONENT keys, as explained below. All numbers entered are assumed to be whole numbers unless you hit , which you'll find between and . The is a decimal point, which you have to use to enter fractional numbers. CHANGE SIGN, as you might have guessed, changes the sign. It may be pressed any time during entry of the mantissa (also directly before or after it) to make the mantissa negative. If there is a negative number in the entry register and you press , it will change this number to positive. If the entry register is positive, will make it negative. The CHANGE SIGN key may also be pressed during entry of an exponent to make the exponent negative.

You can enter a number exponentially by using the EXPONENT key near the bottom of the block of keys just left of the numeral keys. Simply enter the mantissa (with the decimal point in any position), press , and then enter the exponent (it can be either one or two digits). If you don't enter a mantissa, but just hit and enter an exponent, the 445 will assume a mantissa of 1. You can make the exponent negative by pressing after pressing .

Keyboard Buffering

Most keys are electrically interlocked to prevent multiple, simultaneous depression. That's why the 445 goes into ERROR mode when two or more of them are pressed at once. However, these keys are also
buffered to two levels. This means that during calculation initiated by a key, two more keys may be pressed without loss of operation continuity. As long as no two of the keys are pressed at the same time, the 445 will hold the three keys sequentially and perform the operations they indicate in the order in which the keys were pressed. If you press more than two more keys while the machine is working on something for you already, it'll print "ERROR" and stop everything.

A few keys operate independently of the interlocking and buffering. These are:

- RESET
- CLEAR ENTRY
- PAPER ADVANCE
- LIST PROGRAM

BASIC KEYBOARD ARITHMETIC

Arithmetic operations on the 445 are performed algebraically. You enter values and operations pretty much as you would write them in an equation. For example, suppose you wanted to add 4 and 6. You'd simply press 4 then + then 6 and = . The 445 will print both of the numbers entered and the result of the operation. The procedure is the same for subtraction, multiplication and division, with the substitution, of course, of - , \( \times \) or \( \div \) for the +.
CHAPTER 2 -25-

in our example. Multiplication and division are rounded in the thirteenth digit.

Another key which operates like the basic arithmetic keys is $\Box^x$, which you'll find just to the right of them. $\Box^x$ raises a number to a power. You do it like this:

Enter number (a).
Press $\Box^x$.
Enter power (x).
Press $\Box=$.

That's all there is to it. Both (a) and (x) may be positive or negative, integer or fraction. Except that a negative (a) with a noninteger (x) will cause ERROR. And there are also the following little facts to keep in mind:

\[
\begin{align*}
a^0 &= 1 \\
a^1 &= a \\
1^x &= 1 \\
0^x &= 0 \\
0^{-x} &= \text{ERROR} \\
0^0 &= \text{ERROR}
\end{align*}
\]

Just to the right of $\Box^x$ is $\Box^x$. This is the INVERT (or RECIPROCAL) key. It simply takes the reciprocal of the number in the entry register, prints it and leaves it in the entry register. All you do is
CHAPTER 2

press \( \frac{1}{x} \) and the number in the entry register gets the reciprocal treatment.

To the right of \( \frac{1}{x} \) is \( \sqrt{} \), which -- not surprisingly -- takes the square root of the number in the entry register. All you have to do is press \( \sqrt{} \) and the number in the entry register will be printed again and then the square root of that number will be printed and left in the entry register.

SCRATCH PAD REGISTERS

In the left-most block of keys, along with PRINT ENTRY, PRINT ANSWER, RESET, etc., there are two keys stacked vertically that get you into the ten scratch pad registers accessible from the keyboard. These keys are \( \text{ST}_n \) and \( \text{RCL}_n \). That is, STORE \( n \) and RECALL \( n \), where \( n \) stands for the name of the register indicated.

The scratch pad registers are named 0, 1, 2, 3, etc., up to 9. They are not affected by any keyboard operations except sum-square, XY entry, delete, chi-square and register arithmetic (all of which will come up again later on). To store a number in one of these ten registers, all you have to do is enter the number into the entry register, push \( \text{ST}_n \), followed by a single digit to identify the register in which you'd like this number stored. At any later time, you can recall this number from the scratch pad register into the entry register (and therefore onto the tape, if PRINT is ON) simply by pushing \( \text{RCL}_n \).
and the name of the register you put the number into before. It's important to remember that when you use \textit{STn} to put a number into a scratch pad register, the new number will replace anything that was there before. If, for instance, register 6 has 123.456 in it and you enter 32.45, press \textit{STn} and 6, register 6 will now have 32.45 in it. The 123.456 that was there before is completely lost.

\textbf{EXAMPLE:}

\begin{center}
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
0 & 0 & 0 & \textit{SET D. P.} & 3 & 0 & 0 \\
\hline
4 & 5 & 6 & 7 & 8 & \textit{STn} & 3 \\
\hline
1 & 2 & 3 & 4 & 5 & \textit{STn} & 3 \\
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
0 & 0 & 0 & 45678.000 & \downarrow & 3 & 0 & 0 \\
\hline
0 & 0 & 0 & 45678.000 & \uparrow & 3 & 0 & 0 \\
\hline
0 & 0 & 0 & 12345.000 & \downarrow & 3 & 0 & 0 \\
\hline
0 & 0 & 0 & 12345.000 & \uparrow & 3 & 0 & 0 \\
\hline
\end{tabular}
\end{center}

Notice that the 445 prints the number of the register you use as well as a symbol indicating the operation performed. \textit{\downarrow} means \textit{STORE} and \textit{\uparrow} means \textit{RECALL}.

It's also possible to \textbf{add} to a number stored in a scratch pad register.

And you can multiply it, divide it and subtract from it. That's the subject of our next section.
(To be perfectly honest, there aren't really ten scratch pad registers. There are eleven. The decimal point also names a register which has another use we'll find out about later on. But when it's not being used that way, you can use it exactly like any of the other ten scratch pad registers. Just press \( D \) instead of a numeral key in conjunction with the access keys described above and it's at your service.)

Register Arithmetic

You can perform all the basic arithmetic operations on the values stored in the scratch pad registers by using \( ST_n \) in conjunction with the specific arithmetic function key. For example, suppose you wanted to store 4442 in register 0 and then do things to it. You'd press:

\[
\begin{array}{c}
4 & 4 & 4 & 2 \\
\end{array} \quad ST_n \quad 0
\]

Now, if you wanted to add 2 to this number, all you have to do is press:

\[
\begin{array}{c}
2 \\
\end{array} \quad \text{which is the value you're adding}
\]

\[
\begin{array}{c}
\text{ST}_n \\
\end{array} \quad \text{which tells the machine you're going to send it to a scratch pad register}
\]

\[
\begin{array}{c}
+ \\
\end{array} \quad \text{which tells the machine what to do when it gets there}
\]

\[
\begin{array}{c}
0 \\
\end{array} \quad \text{which tells the machine which register to do it in.}
\]

You now have 4444 in register 0. Let's divide it in half. Just go through
the same steps as above, substituting $\div$ for $\times$. (This will cause the number in register 0 to be divided by the number in the entry register.) Now register 0 has 2222 in it. Shall we multiply it times four? OK. Do the same thing again, this time starting off by entering 4 into the entry register, then going through the same sequence, only substituting $\times$ for $\div$. Now press $\text{RCL}_n$ and 0. You should have 8888 in the entry register and on the tape. It's important to note that in every case $\text{ST}_n$ was pressed before the arithmetic function key. And the name of the register after the arithmetic function. This is the only way it will work.

Here's the tape which shows all the arithmetic we just did:

```
4 4 4 2 STn 0
2 STn + 0
RCLn 0
2 STn ÷ 0
RCLn 0
4 STn X 0
RCLn 0
```

```
4442.000 ↓ 0
2.000 + ↓ 0
4444.000 ↑ 0
2.000 ÷ ↓ 0
2222.000 ↑ 0
4.000 X ↓ 0
8888.000 ↑ 0
```
We just recalled 8888 from register 0 into the entry register. Do it again. You'll notice that you get 8888 again. Recalling a number from a register doesn't disturb the register.

We can use numbers stored in registers in another way. We just used a number in the entry register to add to, subtract from, divide or multiply a number in a scratch pad register. We can also do the same thing in reverse -- we can use a number in a scratch pad register to add to, subtract from, divide or multiply a number in the entry register. The procedure is the same, except that this time we start with $\text{RCL}_n$, then indicate the function, followed by the name of the scratch pad register. (In this case, $\text{RCL}_n \div$ will cause the number in the entry register to be divided by the number in the scratch pad register you recall.)

EXAMPLES:

We want to store 2 in register 7, and 4 in register 8.

So we do this:

\[
\begin{array}{c}
2 \text{ STn 7} \\
4 \text{ STn 8}
\end{array}
\]

\[
\begin{array}{c}
2.000 \downarrow 7 \\
4.000 \downarrow 8
\end{array}
\]

Now we can use the numbers stored in these registers to operate on numbers in the entry register.
Note the printed symbols on the right-hand side of the tape which tell you exactly what you did.

One Final Note About Register Arithmetic
When doing arithmetic operations into a scratch pad register, the results will be stored in that register. The entry register is unchanged and any algebraic operation in progress is unaffected. If the register being operated on overflows, its contents will be meaningless, and the machine will go into OVERFLOW condition. When doing arithmetic operations out of a register, the results will be stored in the entry register and may be used in an algebraic sequence. The scratch pad register is unchanged. If the entry register overflows, its contents will be meaningless, and the machine will go into OVERFLOW condition. (Remember that the 445 overflows when it tries to work with numbers outside the range of \(10^{-98}\) to \(10^{+98}\).)

MAIN DATA REGISTERS
In addition to the ten scratch pad registers we've been talking about, there are up to 512 main data storage registers (depending upon your specific machine) which are also directly accessible from the keyboard.
Each of these registers can be used in exactly the same way as a scratch pad register. But there are some differences in the procedure. Primarily because there are more main data registers to name individually. Each of the registers has a numeric name, from 00 through 511. And these numeric names are used to access each register separately. To the left of the top row of keys, there are two keys called \[ \text{ST} \text{nn} \] and \[ \text{RCL} \text{nn} \]. These work just like the \text{STORE} and \text{RECALL} keys we used for the scratch pad registers, except you'll notice that instead of "n", each of these keys has "nn". This is because main data storage registers are accessible only by two-digit numeric code names, instead of the single-digit names of the scratch pad registers. Main data registers 00 through 99 are accessible by pressing either of these two keys (depending upon whether you're storing or recalling) and the two-digit name of the register. But when you get to register 100, there aren't enough places to accommodate the third digit. So we have to use a hundreds code for registers 100 through 199. For this we use the DECIMAL POINT. So if we want to store a number in a register between 100 and 199, we press \[ \text{ST} \text{nn} \] then \[ . \] and then the last two digits of the numeric name. Similarly, for registers 200 through 299, we use \[ \text{CHG} \text{SIGN} \] to stand for two hundreds. And we press the CHANGE SIGN key directly after the \text{STORE} or \text{RECALL} key. For registers 300 through 399, we use \[ \text{EXP} \] to indicate three hundreds. Registers
400 through 499 use $\pi$ as the four hundreds code. Registers 500 through 511 are not directly accessible. That's really all there is to it. Full register arithmetic is available in each of the main data registers, and it is performed exactly the same as with the scratch pad registers. You just have to remember that the access keys require two-digit numeric codes. And you have to remember to use the $\chi$ for hundreds, $\text{CHG}$ for two hundreds, $\text{EXP}$ for three hundreds and $\tau$ for four hundreds. For example, if you wanted to store 25 in register 362 and then multiply it by 30, you'd do this:

\[
\begin{array}{c}
25.000 \\
30.000 \\
750.000
\end{array}
\]

Notice that the 445 prints the number of the register you've used. That is, it prints a two-digit numeral code. If the register number has three digits, only the last two digits will appear on the tape.

**Constant Multipliers and Dividends**

The first number entered in a multiplication or division operation can be used as a constant multiplier or dividend, respectively. Let's say you wanted to multiply 2 by a series of numbers. You'd just do this:
Or, if you wanted to divide 2 by a series of numbers (constant dividend),
you'd do this:
The trick is that when you enter a number and hit \( \text{=} \), the machine goes back and performs the previously entered operation -- using the new number as the second value. You can also get a constant divisor simply by using \( \frac{1}{x} \) (reciprocal). You do it like this:

\[
\begin{align*}
3 & \quad \frac{1}{x} \\
\times & \\
2 & \quad = \\
3 & \quad = \\
4 & \quad = \\
\end{align*}
\]

As you can see, this just takes the reciprocal and sets it up as a constant multiplier. Simple -- but effective.

To the right of the block of numeral keys we've been playing with, along with \( + - \) etc., are two more keys that come in very handy. These
are [ ] and ( ). Parentheses on the 445 can be used very much like those you would write in a formula. You can even have one set of parentheses inside another. When using parentheses on the keyboard, however, there are a few logical rules to keep in mind.

- If you close parentheses [ ] without opening them, the machine goes into ERROR mode.
- Close parentheses ) acts as = with regard to the operation inside the parentheses.
- Two-level nesting is allowed [ i.e., ( () ) ]. But if you try to open a third set of parentheses without closing the second, the machine goes into ERROR mode.

**Algebraic Chaining**

One of the major conveniences on the 445 is algebraic chaining. Any algebraic key may be used instead of equals to terminate a previous algebraic operation and begin a new one. In the course of lengthy operations, this eliminates quite a few steps. For example,

\[
2 + 3 = 5 \\
5 \div 4 = 1.25 \\
1.25 \times 6 = 7.50
\]

can become:
CHAPTER 2  -37-

\[ 2 + 3 \div 4 \times 6 = 7.50 \]

You can also combine algebraic chaining with parentheses. Here are a few examples to illustrate some typical situations involving these techniques.

\[
\frac{4}{(6 + 7)} \quad \frac{4}{(6 + 7)}
\]

\[
\begin{align*}
4 & \div \frac{4}{(6 + 7)} \\
& \frac{4}{(6 + 7)} \\
& \left( \frac{6}{(6 + 7)} \right) \\
& = \frac{4}{(6 + 7)}
\end{align*}
\]

\[
\begin{align*}
& \quad \left( \frac{6}{(6 + 7)} \right) \\
& \quad \left( \frac{5}{(6 + 7)} \right) \\
& \quad \left( \frac{4}{(6 + 7)} \right)
\end{align*}
\]

\[
\begin{align*}
4.000 & \quad \div \\
5.000 & \quad \div \\
9.000 & \quad ( \\
6.000 & \quad ( \\
7.000 & \quad ( \\
13.000 & \quad ) \\
13.000 & \quad = \\
0.692 & \quad *
\end{align*}
\]
Repeated Equals and Repeated Arithmetic

Repeated equals eliminates the necessity to re-enter values. If you depress \( = \) again after terminating an arithmetic operation, the 445 will repeat the operation, using the first-entered value as a constant and operating on the value then in the entry register. For example,
3 \times 3 = 9 \text{ Notice that here we didn't have to enter the 3 again, since it was already in the entry register.}

3 \times 3 = 27 \text{ When we hit } =\text{ the second time, 9 was in the entry register. So the machine just multiplied it by 3, the first-entered value.}

3 \times 3 = 81 \text{ Here, by pushing } =\text{ again, we caused the machine to multiply the 27 that was in the entry register after the last operation again by 3, the first-entered value.}

3 + 2 = 5

3 + 2 = 8 \text{ As above, the second time we pushed } =\text{ the machine went back and this time added the first-entered value to what was in the entry register.}

3 + 2 = 11 \text{ Obviously, we did the same thing again, only this time the entry register had our 8 in it.}

3 \times 2 = 6

3 \times 2 = 18 \text{ Tricky. Even though we entered an intermittent value, 2, the machine still went back when we hit the second } =\text{ and multiplied what was in the entry register by the first-entered value.}

3 \times 2 = 54 \text{ The same as above, except that now the entry register had 18 in it instead of 6.}

As you can see by the very first example, entering a number followed by \[X\] and \[=\] is a fast way to get the square of the number. Repeated equals only works for multiplication and addition. It has no
An operation similar to repeated equals is repeated arithmetic. In this case, we repeat arithmetic keys instead of \( = \). Here's how it works:

\[
\begin{align*}
3 \times &= 9 \quad \text{(Look familiar?)} \quad \text{This is equivalent to } 3^2. \\
3 \times \times &= 81 \quad \text{This is equivalent to } 3^4. \\
3 \times \times \times &= 6561 \quad \text{This is equivalent to } 3^8.
\end{align*}
\]

What's happening is this -- the first time (\( 3 \times = 9 \)), the single digit followed by \( \times \) and \( = \) gives the first-entered value an exponent of 2. Each additional \( \times \) doubles the exponent. Thus two \( \times \)'s equal \( 3^4 \), three \( \times \)'s equal \( 3^8 \), etc. That's with multiplication. With addition, rather than doubling the exponent, each additional \( + \) doubles the answer. Like this:

\[
\begin{align*}
3 + &= 6 \quad \text{ (= } 3 \times 2^1) \\
3 + + &= 12 \quad \text{ (= } 3 \times 2^2) \\
3 + + + &= 24 \quad \text{ (= } 3 \times 2^3)
\end{align*}
\]

Another way to look at this is simply to recognize that each time you hit \( + \) the number in the entry register is added to itself. Repeated subtraction and division yield 0 and 1, respectively, and therefore have no value. Try it.
CHAPTER 2 -41-

MULTIPLE FUNCTIONS

Second Function and Double-Function Keys

Many of the function keys on the 445 keyboard calculate two quantities. When you press a key that calculates two functions, one is printed and left in the entry register; the other is stored in a separate register. To get at the second function, you press 2ND FUNC. The SECOND FUNCTION key exchanges the contents of these two registers and prints the number now in the entry register. If you press 2ND FUNC again, the two registers will again be switched, and the quantities will be back in their original places. When you press a double-function key, the first function will go to the entry register, and the second function will wait in the second function storage register.

A good example of a double-function key is e, which you'll find second from the top in the vertical row of keys just to the left of the numeral keys. Pressing this key puts e (13 digits) into the entry register and puts e (13 digits) into the second function register. If you want to print e, push PRINT ENTRY. And if you want e, push 2ND FUNC. Then if you want e back again, just push 2ND FUNC again.

On the far-right side of the keyboard is a very useful pair of double-function keys -- Ln LOG and e^x 10^x. Ln LOG calculates both the base-e and base-ten logarithms of the number in the entry register. The number and the log_e will be printed. The log_{10} will go into the
CHAPTER 2 -42-

second function register and may be recalled by pressing \[2\text{ND} \text{FNC}\]. Similarly, \[e^x\] calculates the antilogarithm base-e and base-ten of the number in the entry register. The number and the base-e antilogarithm will print. The base-ten antilogarithm will go into the second function register, and may be recalled by pressing \[2\text{ND} \text{FNC}\].

**Third Function**

In addition to functions which calculate two quantities, there are also functions which calculate three quantities. In this case, one is printed and left in the entry register; the second is stored in the second function register and may be recalled by pressing \[2\text{ND} \text{FNC}\]; and the third quantity is stored in the third function register from which it may be recalled by \[\Phi_n \; 3\].

When you press \[\Phi_n \; 3\], the contents of the entry register will be exchanged with the contents of the third function register, and the new value in the entry register will be printed. If you press the two keys again, the entry register and the third function register will again exchange, the new contents of the entry register will be printed and the two numbers will be back where they were before. Note that the contents of the second function register are not affected by the operation of THIRD FUNCTION. However, if you initiate a function that calculates three quantities and then press \[2\text{ND} \text{FNC}\], the second function value will be in the entry register, and the first function value (which
was in the entry register) will be in the second function register. Now if you press \( F_n \) 3, the contents of the entry register (which is the second function value) will be exchanged with the contents of the third function register. The result of all this will be that the first function value will be in the second function register, the second function value will be in the third function register and the third function value will be in the entry register. All of the values may, of course, be returned to their respective registers by reversing the procedure.

**ADDITIONAL FUNCTIONS**

Besides the functions available with individual keys, there are ten additional functions available from the keyboard by means of \( F_n \). Each of these additional functions has a single-digit numeral name, and is initiated by pressing \( F_n \) and then the numeral name of the function you want (as we did for THIRD FUNCTION, above). You'll find a complete list of these additional functions and their numeral names in the strip just below the top row of keys. Each individual function will be described in detail as we get to the area of calculation where it is used.

Except for INTEGER/FRACTION, which we'll describe right here since there's no other logical place to do it. Pressing \( F_n \) 5 will separate the number in the entry register into its integer and fraction portions. The integer portion will be printed and left in the
entry register. The fraction will be put in the second function register and may be recalled with \( \text{2ND} \) \( \text{FUNC} \). Both the integer and the fraction will retain the sign of the original number.

STATISTICAL FUNCTIONS

Now we get to the statistical keys and the remaining Additional Functions -- which perform specialized statistical calculations faster than a speeding bullet. The statistical functions on the 445 are interrelated in many ways, and are used together to produce results tailored to specific needs.

Set Group and Data Storage

Data entered with the keyboard data summation keys are stored in the ten scratch pad registers we discussed earlier. Summation data for three independent or dependent groups may be kept in these registers simultaneously. \( \text{SET} \) \( \text{GROUP} \) is used to tell the machine which Group or Groups of independent data to use — either for accumulating data or calculating functions. Dependent data are always accumulated in the same registers regardless of the Group set.

A Group is set with \( \text{SET} \) \( \text{GROUP} \) followed by \( 1 \) \( 2 \) or \( 3 \). When a Group is set, you can accumulate and delete data (n-count, \( \Sigma X \), \( \Sigma X^2 \)) and calculate standard deviation, mean, standard error of the mean and z-statistic -- all with respect to that Group. Setting
CHAPTER 2

Group 1, 2 or 3 also determines which two Groups are used in the t-statistic and two-variable linear regression and line calculations.

**SET**

followed by the digit 4 causes linear regression and Line functions to operate on a three-variable basis rather than the normal two-variable basis. Other group-dependent functions initiated while Group 4 is set will automatically use the Group that was in operation just before Group 4 was set. Setting Group 1, 2 or 3 will remove the Group 4 setting and return linear regression and Line calculations to the normal two-variable basis.

We'll keep track of what data are in which register of which Group as we get to particular functions which use these Groups.

A NOTE TO BENNY:

Since Set Group data are accumulated in the ten scratch pad registers, you can enter summed data directly into their appropriate registers just as well as accumulating them with the keyboard summation keys. The function keys don't care how the data got there, just so long as they're in the right places. Then you can perform all the calculations you like on that data -- just as though they had been accumulated by the machine. (Except for three-variable data, where there's a little trick you have to use if you enter summed data. See the note following Three-Variable Linear Regression.)

When you do accumulate data with the keyboard summation keys, it's a good idea to make sure the scratch pad registers are clear before
you begin. This is done easily with \( \Phi_n \) 0 which clears scratch pad registers 0-9 all at once.

**Sum-Square**

Using the number in the entry register as \( X \), this key accumulates \( n \), \( \sum X \) and \( \sum X^2 \). Depending upon which Group you’ve got set, \( \sum_{n=1}^{X} x^2 \) puts the data in these registers:

<table>
<thead>
<tr>
<th>Group</th>
<th>Register Usage</th>
<th>( n )</th>
<th>( \sum X )</th>
<th>( \sum X^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Like this:

\[ \text{SET D. P.} \ 2 \]

These operations do not print.

\[ \text{SET GROUP} \ 1 \]

\[ \Phi_n \ 0 \]

Clears scratch pad registers 0-9

\[ \text{ADV} \]

Separates CL from data

\[ 2 \ . \ 1 \ \sum_{n} x^2 \]

\[ 2 \ . \ 5 \ \sum_{n} x^2 \]

\[ 3 \ . \ 1 \ \sum_{n} x^2 \]
Note the print symbols on the right-hand side of the tape. SET DP and SET GROUP do not print. CL means we've cleared the scratch pad registers. The \( \sum \) means that a summation was performed on the entered value. The 1 next to the \( \sum \) means that the summation went into Set Group 1. Therefore, our sums should be in registers 1, 2 and 3, according to the chart. Let's see if they are.

<table>
<thead>
<tr>
<th>( \text{RCL}_n )</th>
<th>1</th>
<th>number of items (n)</th>
<th>3.00</th>
<th>( \uparrow )</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{RCL}_n )</td>
<td>2</td>
<td>( \sum X )</td>
<td>7.70</td>
<td>( \uparrow )</td>
<td>2</td>
</tr>
<tr>
<td>( \text{RCL}_n )</td>
<td>3</td>
<td>( \sum X^2 )</td>
<td>20.27</td>
<td>( \uparrow )</td>
<td>3</td>
</tr>
</tbody>
</table>

As you can see, our summations are in their proper places. Now let's do another group of summations, this time into Set Group 2.

| \( \sum_{n} x^2 \) | 7 | 7.00 | \( \sum \) | 2 |
| \( \sum_{n} x^2 \) | 8 | 8.10 | \( \sum \) | 2 |
| \( \sum_{n} x^2 \) | 9 | 9.90 | \( \sum \) | 2 |

And, just to keep the machine honest, let's recall the registers of Set Group 2 and see what's in them.
CHAPTER 2

\[ \text{RCL}_n\ 4 \ \text{number of items (n)} \]
\[ \begin{array}{c|c}
3.00 & \uparrow 4 \\
\end{array} \]

\[ \text{RCL}_n\ 5 \ \sum x \]
\[ \begin{array}{c|c}
25.00 & \uparrow 5 \\
\end{array} \]

\[ \text{RCL}_n\ 6 \ \sum x^2 \]
\[ \begin{array}{c|c}
212.62 & \uparrow 6 \\
\end{array} \]

Everything is where it's supposed to be. Now let's do one final summation into Set Group 3.

\[ \text{SET GROUP}\ 3 \]
\[ \begin{array}{c|c|c}
1 & 3 & \sum_{n=2} x^2 \\
1 & 1 & \sum_{n=2} x^2 \\
1 & 3 & 5 \sum_{n=2} x^2 \\
1 & 4 & 1 \sum_{n=2} x^2 \\
\end{array} \]
\[ \begin{array}{c|c}
13.00 & \sum 3 \\
11.00 & \sum 3 \\
13.50 & \sum 3 \\
14.10 & \sum 3 \\
\end{array} \]

And, just for consistency's sake, let's see what's in the Set Group 3 registers.

\[ \text{RCL}_n\ 7 \ \text{number of items (n)} \]
\[ \begin{array}{c|c}
4.00 & \uparrow 7 \\
\end{array} \]

\[ \text{RCL}_n\ 8 \ \sum x \]
\[ \begin{array}{c|c}
51.60 & \uparrow 8 \\
\end{array} \]

\[ \text{RCL}_n\ 9 \ \sum x^2 \]
\[ \begin{array}{c|c}
671.06 & \uparrow 9 \\
\end{array} \]
"Grouped" Data

If you've got a whole group of identical items to enter, you don't have to enter them individually.

You can enter a group of data with this sequence:

```
enter X
press \[ \sum \]
press \[ \sum nxx^2 \]
enter frequency (f) of X
press \[ = \]
```

If \( f = 20 \), and the Set Group is 1, 20 will be added to register 1; \( 20X \) will be added to register 2; and \( 20X^2 \) will be added to register 3.

Let's enter a group.

```
SET
GROUP
2

Φn
0

ADV

5.4 is entered 8 times

8 =
```

<table>
<thead>
<tr>
<th>SET</th>
<th>GROUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.40</td>
<td>8.00</td>
</tr>
</tbody>
</table>

Clears scratch pad registers 0-9
Separates CL from data
As you can see, the tape symbol for the frequency of \( X \) is \( n \) (i.e., the number of times \( X \) is entered). Let's see what got put in the Set Group 2 registers.

\[
\begin{align*}
\text{RCL}_n & 4 \quad \text{number of items (n)} & 8.00 & \uparrow 4 \\
\text{RCL}_n & 5 \quad \sum X \ (5.4 \times 8) & 43.20 & \uparrow 5 \\
\text{RCL}_n & 6 \quad \sum X^2 \ (5.4^2 \times 8) & 233.28 & \uparrow 6
\end{align*}
\]

Let's add another group on top of that one.

\[
\begin{align*}
\begin{array}{c}
5 \quad 6 \quad \sum_{n=1}^{3} n \times x^2 \\
3 \quad =
\end{array} & 5.6 \text{ is entered 3 times} & 5.60 & \sum 2 \\
& & 3.00 & n
\end{align*}
\]

Now let's see what's in those registers.

\[
\begin{align*}
\text{RCL}_n & 4 \quad \text{number of items} \ (8 + 3) & 11.00 & \uparrow 4 \\
\text{RCL}_n & 5 \quad \sum X \ (5.4 \times 8) + (5.6 \times 3) & 60.00 & \uparrow 5 \\
\text{RCL}_n & 6 \quad \sum X^2 \ (5.4^2 \times 8) + (5.6^2 \times 3) & 327.36 & \uparrow 6
\end{align*}
\]

Our second group of data was tidily added to the first group.

**Deleting Data**

Also, you can remove data from a summation by using the DELETE key.

Just enter the item you want removed from the summation along with
DELETE, and the \( n \), \( X \) and \( X^2 \) will be removed from their respective registers. You can press DELETE before or after entering \( X \), just so long as you've pressed DELETE before you press \( \sum_{nxx^2} \).

To remove a group of identical data, use the same sequence as you would use for entering the data, but press DELETE before you get to =.

Here's an example of a Sum-Square sequence using the keys we've been talking about:

\[
\begin{align*}
\text{SET} & \quad 3 \\
\text{GROUP} & \\
\Phi \ n & \quad 0 \\
\text{ADV} &
\end{align*}
\]

separates CL from data

\[
\begin{align*}
2 \ . \ 1 & \quad \sum_{nxx^2} \\
2 \ . \ 3 & \quad \sum_{nxx^2} \\
2 \ . \ 3 \ . \ 4 & \quad \sum_{nxx^2} \\
4 \ . \ = & \\
2 \ . \ 6 & \quad \sum_{nxx^2} \\
8 \ . \ 6 & \quad \sum_{nxx^2}
\end{align*}
\]

\[
\begin{align*}
0.00 & \quad \text{CL} \\
2.10 & \quad \sum 3 \\
2.30 & \quad \sum 3 \\
2.34 & \quad \sum 3 \\
4.00 & \quad n \\
2.60 & \quad \sum 3 \\
8.60 & \quad \sum 3
\end{align*}
\]
2.25 is entered 17 times

8.6 is removed from summation

11 of the 2.25's we entered before are removed here.

Note the print symbols for summation and deletion. When you use \( \sum \) to add data, the print symbol is \( \sum \). When you use \( \text{DELETE} \), the print symbol is \( \overline{\sum} \). Also, the tape will advance a space automatically when you enter or delete a group of data.

Now let's take a look into these Set Group 3 registers.

If you add up the items we entered, and remove the ones we removed, you'll discover that our little machine is keeping it all quite orderly.
Standard Deviation, Mean and Standard Error Of The Mean (n-1 version)

The SD/MEAN key calculates the standard deviation, mean and standard error of the mean for the Set Group which is active. This function uses the following formulas:

$$SD = \sqrt{\frac{\sum x^2 - (\sum x)^2}{n - 1}}$$

(Note that the denominator under the radical is n-1, not n, so this key calculates the estimate of the standard deviation for this Group.)

$$\bar{X} = \frac{\sum X}{n}$$

Standard error = \frac{SD}{\sqrt{n}}

The standard deviation is printed; the mean is put into the second function register and may be recalled with \[2ND\] \[FUNC\]; and the standard error of the mean is put in the third function register and may be recalled with \[3\]. No data is destroyed while performing these calculations.

Here's an example:

SET D. P. \[3\]

SET GROUP \[3\]
CHAPTER 2

\[
\bar{X}_{n} = 0
\]

\[
\begin{align*}
1 &\ \ 4.5 \ \ \sum nxx^2 \\
1 &\ \ 4.7 \ \ \sum nxx^2 \\
1 &\ \ 5.0 \ \ \sum nxx^2 \\
1 &\ \ 4.4 \ \ \sum nxx^2 \\
1 &\ \ 4.7 \ \ \text{DELETE} \ \ \sum nxx^2 \\
1 &\ \ 4.8 \ \ \sum nxx^2 \\
\end{align*}
\]

\[\text{ADV}\]

\[\begin{align*}
\text{SD} &\ \ \text{MEAN} \quad \text{takes standard deviation (estimate)} \\
\text{2ND} &\ \ \text{FUNC} \quad \text{gives us the mean} \\
\end{align*}\]

\[
\bar{X}_{n} = 3
\]

gives us the standard error of the mean

Note the print symbol (SDn-1) that indicates which standard deviation you've taken.

Standard Deviation, Mean and Standard Error Of The Mean (n version)

Pressing \[\bar{X}_{n} \ 4\] will get you the standard deviation, mean and standard error of the mean, just like \[\text{SD MEAN}\], above, except that here the formula for standard deviation is:
CHAPTER 2

\[ SD = \sqrt{\frac{\sum X^2 - \left( \frac{\sum X}{n} \right)^2}{n}} \]

(Since the denominator under the radical is \( n \) (not \( n-1 \)) this function is calculating the sample standard deviation.)

Using the data we've already entered in the example above, let's take the sample standard deviation, mean and standard error:

\[ \text{SD} \quad 4 \]
\[ \text{takes standard deviation (of sample)} \quad 0.233 \quad \text{SDn} \quad 3 \]
\[ \text{2ND} \quad \text{FUNC} \]
\[ \text{gives us the mean} \quad 14.692 \quad \text{F} \quad 2 \]
\[ \text{SD} \quad 3 \]
\[ \text{gives us the standard error of the mean} \quad 0.116 \quad \text{F} \quad 3 \]

Note the print symbol (SDn) that indicates which standard deviation you've taken.

z-Statistic

The function \[ \text{SD} \quad 1 \] calculates the z-statistic of the current Set Group. Using the number in the entry register as \( X \), here's the formula:

\[ z = \frac{X - \bar{X}}{SD} \]

Each time you press \[ \text{SD} \quad 1 \], the SD and \( \bar{X} \) are calculated all over again for the above formula. Here, the sample SD (not the estimate)
is used. You needn't have done a SD or \( \overline{x} \) calculation beforehand. But you do have to be sure that the Group on which you're taking a z-statistic has data in it.

The sequence you use to take a z-statistic is as follows:

1. enter X
2. press \( \overline{x} \)
3. 1

That's all there is to it. Both X and z are printed; z is left in the entry register and may be used directly as input to the normal distribution function.

Since we've already got the right kind of data in our Group 3, and our machine is already set to that group, to get a z, all we have to do is this:

\[
1 4 . 9 8 \overline{x} 1
\]

our X 14.980 X
our z 1.231 Z

Let's do it again with a new X:

\[
1 4 . 9 \overline{x} 1
\]

our X 14.900 X
our z 0.889 Z

Normal Distribution \( \Phi \) calculates the area under the normal distribution curve.

Using the number in the entry register as z, the total area from -∞
to \( z \) (one-tailed probability) is printed and kept in the entry register. The area from \(-z\) to \( +z \) (two-tailed probability) is put into the second function register and may be recalled with \( \text{2ND FUNC} \). The formula used for normal probability is

\[
y = \frac{1}{\sqrt{2 \pi}} (e^{-z^2/2})
\]

If \( z < 0 \), the area from \(-z\) to \( +z \) is negative. The area in all cases is calculated to six fractional digits -- remaining digits are zeroed.

Again using the data we entered into Set Group 3 in the example under Standard Deviation (n-1) above, let's take a \( z \) and calculate the normal probability.

Here, we used the \( z \) which \( \Phi \) \( \sqrt{ } \) 1 put into the entry register as our entry for the normal probability calculation. But you don't have to do a \( z \) calculation first. You can get all the normal curve information with the entry of any \( z \) you like. Like this:

You automatically get one-tailed probability

Now you get two-tailed probability
Multiple Variable Data Entry

The XY key is used to accumulate both two-variable and three-variable data.

TWO-VARIABLE

For two-variable data, the sequence is as follows:

1. enter X
2. press \( \text{XY} \)
3. enter Y
4. press \( \text{=} \)

With this sequence, data is accumulated in these scratch pad registers:

<table>
<thead>
<tr>
<th>Register</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \Sigma XY )</td>
</tr>
<tr>
<td>1</td>
<td>( n )</td>
</tr>
<tr>
<td>2</td>
<td>( \Sigma X )</td>
</tr>
<tr>
<td>3</td>
<td>( \Sigma X^2 )</td>
</tr>
<tr>
<td>4</td>
<td>( n )</td>
</tr>
<tr>
<td>5</td>
<td>( \Sigma Y )</td>
</tr>
<tr>
<td>6</td>
<td>( \Sigma Y^2 )</td>
</tr>
</tbody>
</table>

Note that X data is stored in registers 1, 2 and 3 and Y data is stored in registers 4, 5 and 6. Two-variable data entry with \( \text{XY} \) always puts the first variable in Group 1 and the second variable in Group 2. So you can do standard deviation, z-statistic, etc. with either the X
or the Y data accumulated this way. (That's why \( n \) is stored in both register 1 and register 4.) Let's do a two-variable summation:

\[
\begin{array}{c}
\text{ADV} \\
2 \text{ XY} \\
3 \text{ } = \\
4 \text{ XY} \\
5 \text{ } = \\
6 \text{ } = \\
\end{array}
\]

Note the X and Y print symbols which identify each data item entered.

Also notice that the tape skips a space after each XY group. Now let's take a look at what's in those registers:
The DELETE key is used to remove data from an \( \sum_{i} XY \) summation just like it was used before to remove data from a \( \sum_{i} n \cdot x \cdot x^2 \) summation.

Let's try removing some data from this summation:

\[
\begin{array}{c}
4 \quad XY \\
6 \quad \text{DELETE} \quad =
\end{array}
\]

(Note the - after Y on the tape, indicating that these data have been deleted.)

And let's look at the registers again:
The bad data are gone, and the good data live on.

THREE-VARIABLE

For three-variable data accumulation, the sequence is as follows:

- enter X
- press \[XY\]
- enter Y
- press \[XY\] again
- enter Z
- press \[=\]

With this sequence, data are accumulated in this format:
### Register Data

<table>
<thead>
<tr>
<th>Register</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\Sigma XY$</td>
</tr>
<tr>
<td>1</td>
<td>n</td>
</tr>
<tr>
<td>2</td>
<td>$\Sigma X$</td>
</tr>
<tr>
<td>3</td>
<td>$\Sigma X^2$</td>
</tr>
<tr>
<td>4</td>
<td>$\Sigma YZ$</td>
</tr>
<tr>
<td>5</td>
<td>$\Sigma Y$</td>
</tr>
<tr>
<td>6</td>
<td>$\Sigma Y^2$</td>
</tr>
<tr>
<td>7</td>
<td>$\Sigma XZ$</td>
</tr>
<tr>
<td>8</td>
<td>$\Sigma Z$</td>
</tr>
<tr>
<td>9</td>
<td>$\Sigma Z^2$</td>
</tr>
</tbody>
</table>

Here's an example of three-variable data entry:

1. **$\Phi_n$ 0** clears scratch pad registers 0-9
2. **ADV** Separates CL from data
3. **2 XY**
4. **3 XY**
5. **4 =**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.000</td>
<td>CL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.000</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.000</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.000</td>
<td>Z</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Note that the tape skips a space after each group of three data items, just as it skips a space after each set of two items with the two-variable data sequence. And now the machine prints an X, a Y and a Z on the tape to make it easy to identify each of the data items individually.

Now to take a peek at those registers:

\[
\begin{array}{c}
\text{RCL}_n \quad 0 \quad \Sigma XY \\
\text{RCL}_n \quad 1 \quad n \\
\text{RCL}_n \quad 2 \quad \Sigma X \\
\text{RCL}_n \quad 3 \quad \Sigma X^2 \\
\text{RCL}_n \quad 4 \quad \Sigma YZ
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{IRCL} & 10 & \text{IRCL} \\
66.000 & \uparrow & 0 \\
3.000 & \uparrow & 1 \\
10.000 & \uparrow & 2 \\
54.000 & \uparrow & 3 \\
112.000 & \uparrow & 4 \\
\end{array}
\]
We can, of course, remove a group of data from our summation, like this:

And now the registers will look like this:
Note that when you accumulate data for three dependent variables, n is stored only in register 1 (rather than register 1 for Group 1, register 4 for Group 2 and register 7 for Group 3). For this reason, accumulation of data by the three-variable sequence sets an internal flag which tells the machine to get n from register 1 when doing SD, z and linear regression no matter which Group is set. So you can safely calculate SD and z for each of the 3 variables, even though you put the data in with the XY key and n is not in registers 4 and 7 for Set Groups 2 and 3. This flag is reset by entering data in the two-variable sequence described above, by turning the machine off, or by

\[ \Phi_n \ 0 \]

**Linear Regression**

\[ \text{LIN} \ \text{REG} \]

performs both two-variable and three-variable linear regression.

Setting the Group to 1, 2 or 3 indicates two-variable linear regression.
CHAPTER 2

Setting Group 4 indicates three-variable regression. Data for two-variable linear regression are entered as described above, using $\Sigma_{X}Y$ and the two-variable data sequence. Data for three-variable linear regression are entered as described above, too, using $\Sigma_{X}Y$ and the three-variable data sequence. Since $\Sigma_{X}X^2$ does not accumulate cross-products ($Z_{XY}, Z_{YZ}$, etc.) linear regression cannot be calculated for data entered with this key.

TWO-VARIABLE

Two-variable linear regression may be performed between any two of the three summation groups. The Group set may be 1, 2 or 3. Here's a chart that defines which two groups are used with each setting and where the data come from. (Don't forget that when you enter two-variable data with the XY key, X goes into Set Group 1 and Y goes into Set Group 2 automatically, no matter what Set Group is current. So when you do a regression on that data, Group 1 must be set.)

<table>
<thead>
<tr>
<th>SET GROUP</th>
<th>GROUPS USED</th>
<th>REGISTER USAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Indep. Var. (X)</td>
<td>Dep. Var. (Y)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
When [LIN REG] is pressed, the coefficients $r$, $m$, and $i$ are calculated -- where $Y = mX + i$ is the least-square regression equation of $Y$ on $X$; $r$ is the correlation coefficient; $m$ is the slope; and $i$ is the intercept on the $Y$ axis. These coefficients are calculated as follows:

$$r = \frac{\sum XY - \frac{\sum X \sum Y}{n}}{\sqrt{\left(\frac{\sum X^2 - (\sum X)^2}{n}\right) \left(\frac{\sum Y^2 - (\sum Y)^2}{n}\right)}}$$

$$m = \frac{\sum XY - \frac{\sum X \sum Y}{n}}{\sum X^2 - (\sum X)^2}$$

$$i = \frac{\sum X - m \sum X}{n}$$

When you set the Group to 1, 2 or 3 and press [LIN REG], the coefficient is printed and left in the entry register. The slope is put in the second function register and may be recalled with [2ND FUNC]. And the intercept is put in the third function register and may be recalled with [\text{Fn}] 3.

Here's an example:

$\text{Fn} 0$

separates CL from data
CHAPTER 2

1  XY
3  =
1  XY
4  =
2  XY
5  =
1 .5 XY
6  =

1.000  X
3.000  Y
1.000  X
4.000  Y
2.000  X
5.000  Y
1.500  X
6.000  Y

SET GROUP 1

this guarantees that computation will use Set Groups 1 & 2

LIN REG

calculates correlation coefficient

0.674 LR 1

2ND FUNC

gives us the slope

1.818 F 2

3

gives us the intercept

2.000 F 3
THREE-VARIABLE

Three-variable linear regression is performed by setting Group 4 and then pressing \( \text{LIN} \) \( \text{REG} \). Assuming, of course, that you've entered data in the three-variable sequence, as described under Multiple Variable Data Entry, above. The coefficients \( i, m_1 \) and \( m_2 \) are calculated where

\[ Z = i_1 + m_1 X + m_2 Y \]

is the least-square regression equation of \( Z \) on \( X \) and \( Y \). Here, \( i, m_1 \) and \( m_2 \) are determined by simultaneous solution of the following equations:

\[
\begin{align*}
\Sigma Z &= i_n + m_1 \Sigma X + m_2 \Sigma Y \\
\Sigma ZX &= i ZX + m_1 \Sigma X^2 + m_2 \Sigma XY \\
\Sigma ZY &= i ZY + m_1 \Sigma XY + m_2 \Sigma Y^2
\end{align*}
\]

After these calculations are performed, \( i \) is printed and left in the entry register; \( m_1 \) is put into the second function register and may be recalled with \( \text{2ND} \) \( \text{FUNC} \); and \( m_2 \) is put into the third function register, from which it may be recalled with \( \text{n} \) \( 3 \).

Let's do a three-variable linear regression to illustrate these tricks.

\[
\begin{align*}
\text{n} &= 0 \\
\text{ADV} & \text{ separates CL from data}
\end{align*}
\]
<table>
<thead>
<tr>
<th>SET</th>
<th>GROUP</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
<td>1.000</td>
<td>2.000</td>
<td>3.000</td>
<td>4.000</td>
<td>5.000</td>
<td>6.000</td>
<td>7.000</td>
<td>8.000</td>
<td>9.000</td>
<td>1.000</td>
<td>4.000</td>
<td>7.000</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td></td>
</tr>
</tbody>
</table>

Indicates 3-variable situation
CHAPTER 2

\[ Z = i + m_1 X + m_2 Y \]

\[ r_{xy} \] prints (r between Groups 1 and 2)  
\[ r_{yz} \] prints (r between Groups 2 and 3)

We can also get the correlation coefficient, slope and intercept between any two of the three variables for which we've collected data.

Like this:
(r = correlation coefficient)

ANOTHER NOTE TO BENNY:
As you may recall, the three-variable data entry sequence sets an internal flag to tell standard deviation, z and linear regression where to get n. If, however, you've stored the summed data directly into their respective registers, this flag never got set. Therefore the machine doesn't know where to get n for these calculations. So you have to set the flag yourself by setting Group 4 before you set the actual Group you want set for calculating.

LINE
Using the data collected for Linear Regression, this function calculates the dependent variable of either the two- or three-variable regression equation. With Group 1, 2 or 3 set, \( \sum n \) will calculate and
print $Y_{est}$ where $Y_{est} = mX+i$. This assumes that $X$ is in the entry register; $m$ and $i$ are as described under Linear Regression, above.

You don't have to use LIN REG before using $\Phi_n$ because the Line function automatically does a linear regression to determine $m$ and $i$.

Here's an example:

```
$\Phi_n$ 0
```

ADV separates CL from data

```
1 XY
3 =
```

```
1 XY
4 =
```

```
2 XY
5 =
```

```
1. 5 XY
6 =
```

<table>
<thead>
<tr>
<th></th>
<th>0.000</th>
<th>CL</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>3.000</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>1.000</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>4.000</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>2.000</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>5.000</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>1.500</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>6.000</td>
<td></td>
</tr>
</tbody>
</table>
You give it $X$
It gives you $Y_{est}$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.000</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3.818</td>
<td>Y</td>
<td>1</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.500</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4.727</td>
<td>Y</td>
<td>1</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2.500</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6.545</td>
<td>Y</td>
<td>1</td>
</tr>
</tbody>
</table>

With Group 4 set, the LINE function will do a three-variable Line calculation where $Z_{est} = m_1X + m_2Y + i$. Here, $m_1$, $m_2$ and $i$ are the same as described under three-variable Linear Regression, above.

$X$ and $Y$ are entered like this:

enter $X$

press $\overline{9}$

enter $Y$

press $\overline{9}$

$Z_{est}$ is calculated, printed and left in the entry register.

Like this:
<table>
<thead>
<tr>
<th>CL</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.500</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.600</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.500</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.600</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.800</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

 separat CL from data

\[ \begin{array}{c}
\text{ADV} \\
1 \ XY \\
2 \ XY \\
3 \ = \\
7 \ XY \\
8 \ XY \\
9 \ = \\
1 \ 5 \ XY \\
8 \ 6 \ XY \\
3 \ 1 \ = \\
7 \ 5 \ XY \\
4 \ 6 \ XY \\
1 \ 8 \ = 
\end{array} \]
CHAPTER 2

SET GROUP 4 indicates 3-variable situation

1  X  1.000  X

2  Y  2.000  Y

Z  →

7  X  7.000  Z

8  Y  8.000  Y

Z  →

2  X  2.000  X

4  Y  4.000  Y

Z  →

2.823  Z  4

Independent and Dependent t

This key calculates the dependent t-statistic and the independent t-statistic according to the following formulas:

\[
t_{dep} = \frac{\bar{X} - \bar{Y}}{\sqrt{\frac{SD_X^2 + SD_Y^2 - 2r SD_X SD_Y}{n}}}\]
CHAPTER 2 -77-

\[ t_{\text{ind}} = \frac{\bar{X} - \bar{Y}}{\sqrt{\frac{(n_X-1) \text{SD}_X^2 + (n_Y-1) \text{SD}_Y^2}{n_X + n_Y - 2} \left[ \frac{1}{n_X} + \frac{1}{n_Y} \right]}} \]

Where

\[ \bar{X} = \frac{\sum X}{n_X} \]

\[ \bar{Y} = \frac{\sum Y}{n_Y} \]

\[ \text{SD}_X = \text{standard deviation of } X \ (n-1) \]

\[ \text{SD}_Y = \text{standard deviation of } Y \ (n-1) \]

\[ r = \text{coefficient (see Linear Regression, above.)} \]

Dependent variables should be entered with \[ \sum XY \]. Independent variables should be entered with \[ \sum nxx^2 \]. The Set Group and register usage format looks like this:

<table>
<thead>
<tr>
<th>SET GROUP</th>
<th>GROUPS USED</th>
<th>( n_X )</th>
<th>( \sum X )</th>
<th>( \sum x^2 )</th>
<th>( n_Y )</th>
<th>( \sum Y )</th>
<th>( \sum y^2 )</th>
<th>( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 &amp; 2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2 &amp; 3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3 &amp; 1</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
As you can see, the summations and n's used are dependent upon the Group set.

With the data stored, when you press \( t_{\text{dep}} \) the dependent t-statistic is printed and left in the entry register; and the independent t-statistic is put into the second function register and may be recalled with \( 2\text{ND} \text{FUNC} \).

Let's do a dependent t-statistic:

- \( \Phi_n \) 0
- ADV
- separates CL from data
- 1 \( \text{XY} \)
- 3 =
- paired data entry
- 1 \( \text{XY} \)
- 4 =
- 2 \( \text{XY} \)
- 5 =
As you can see, we've entered the data with \( \text{XY} \) because the data are dependent. Note the delightful little \( t \) on the tape which identifies the dependent t-statistic. The digit before the little \( t \) indicates the Groups compared. Now let's do an independent t:

\[
\begin{align*}
1.500 & \quad X \\
6.000 & \quad Y \\
\text{SET} & \quad \text{GROUP} \quad 1 \\
\text{t}^{\text{dep}} & \quad \text{dependent t-statistic} \\
& \quad \text{printed} \\
\text{F} & \quad 2 \\
\text{2ND} & \quad \text{FUNC} \\
& \quad \text{gives you independent t-statistic} \quad \text{-- may be of same value}
\end{align*}
\]
CHAPTER 2

<table>
<thead>
<tr>
<th>SET GROUP</th>
<th>2</th>
<th>ADV</th>
</tr>
</thead>
</table>

set new Group, advance tape for clarity

| 4 | 5 | \( \sum_{nxx} \) |
| 4 | 8 | \( \sum_{nxx} \) |
| 4 | 9 | \( \sum_{nxx} \) |
| 4 | 1 | \( \sum_{nxx} \) |
| 4 | 4 | \( \sum_{nxx} \) |
| 5 | 2 | \( \sum_{nxx} \) |
| 5 | 1 | \( \sum_{nxx} \) |

This time, we've entered the data with \( \sum_{nxx} \) instead of \( XY \) because we have independent data. And we have to set Group 1 because our independent data are in Groups 1 and 2, and the machine will

\[
\begin{align*}
4.500 & \quad \Sigma \ 2 \\
4.800 & \quad \Sigma \ 2 \\
4.910 & \quad \Sigma \ 2 \\
4.190 & \quad \Sigma \ 2 \\
4.400 & \quad \Sigma \ 2 \\
5.200 & \quad \Sigma \ 2 \\
5.120 & \quad \Sigma \ 2
\end{align*}
\]

\[
\begin{align*}
t_{dep} & = -0.963 & 1 t \\
t_{ind} & = -12.929 & F \ 2
\end{align*}
\]
automatically use the data in the Group set and the next group. We could have put the data in Groups 2 and 3 just as well. Like this:

\[
\begin{array}{cccc}
\Phi & 0 & \text{SET} & 2 \\
2 & 1 & \sum \frac{nxx^2}{nxx} & \\
2 & 2 & \sum \frac{nxx^2}{nxx} & \\
2 & 2 & 5 & \sum \frac{nxx^2}{nxx} \\
2 & 6 & \sum \frac{nxx^2}{nxx} & \\
2 & 4 & 2 & \sum \frac{nxx^2}{nxx} \\
\text{SET} & 3 & \text{ADV} \\
4 & 5 & \sum \frac{nxx^2}{nxx} & \\
4 & 8 & \sum \frac{nxx^2}{nxx} & \\
4 & 9 & 1 & \sum \frac{nxx^2}{nxx} \\
4 & 1 & 9 & \sum \frac{nxx^2}{nxx} \\
4 & 4 & \sum \frac{nxx^2}{nxx} & \\
5 & 2 & \sum \frac{nxx^2}{nxx} & \\
5 & 1 & 2 & \sum \frac{nxx^2}{nxx}
\end{array}
\]

<table>
<thead>
<tr>
<th>Data</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>CL</td>
</tr>
<tr>
<td>2.100</td>
<td>\Sigma 2</td>
</tr>
<tr>
<td>2.200</td>
<td>\Sigma 2</td>
</tr>
<tr>
<td>2.250</td>
<td>\Sigma 2</td>
</tr>
<tr>
<td>2.600</td>
<td>\Sigma 2</td>
</tr>
<tr>
<td>2.420</td>
<td>\Sigma 2</td>
</tr>
<tr>
<td>4.500</td>
<td>\Sigma 3</td>
</tr>
<tr>
<td>4.800</td>
<td>\Sigma 3</td>
</tr>
<tr>
<td>4.910</td>
<td>\Sigma 3</td>
</tr>
<tr>
<td>4.190</td>
<td>\Sigma 3</td>
</tr>
<tr>
<td>4.400</td>
<td>\Sigma 3</td>
</tr>
<tr>
<td>5.200</td>
<td>\Sigma 3</td>
</tr>
<tr>
<td>5.120</td>
<td>\Sigma 3</td>
</tr>
</tbody>
</table>
You can do an independent t between Groups 3 and 1, too. Like this:

\[ \Phi_n \quad 0 \quad \text{SET GROUP} \quad 3 \]

\[ \begin{align*}
2 & \cdot 1 \quad \sum_{nxx} \quad 2 \\
2 & \cdot 2 \quad \sum_{nxx} \quad 2 \\
2 & \cdot 2 \cdot 5 \quad \sum_{nxx} \quad 2 \\
2 & \cdot 6 \quad \sum_{nxx} \quad 2 \\
2 & \cdot 4 \cdot 2 \quad \sum_{nxx} \quad 2 
\end{align*} \]

\[ \begin{array}{c}
0.000 \quad \text{CL} \\
2.100 \quad \sum 3 \\
2.200 \quad \sum 3 \\
2.250 \quad \sum 3 \\
2.600 \quad \sum 3 \\
2.420 \quad \sum 3 
\end{array} \]
If you've entered three-variable dependent data, you can take t-statistics between any two of the variables. Here's how:

- **SET GROUP 1 ADV**
- **SET GROUP 2 ADV**
- **SET GROUP 3 ADV**

**don't need a dummy number since register 1 has something in it**

**again meaningless here**

**that's our number**

If you've entered three-variable dependent data, you can take t-statistics between any two of the variables. Here's how:

- **tdep**
- **tind**
- **2ND FUNC**

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Cl</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.500</td>
<td>≤ 1</td>
<td></td>
</tr>
<tr>
<td>4.800</td>
<td>≤ 1</td>
<td></td>
</tr>
<tr>
<td>4.910</td>
<td>≤ 1</td>
<td></td>
</tr>
<tr>
<td>4.190</td>
<td>≤ 1</td>
<td></td>
</tr>
<tr>
<td>4.400</td>
<td>≤ 1</td>
<td></td>
</tr>
<tr>
<td>5.200</td>
<td>≤ 1</td>
<td></td>
</tr>
<tr>
<td>5.120</td>
<td>≤ 1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>t-statistic</th>
<th>Cl</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.194</td>
<td>3 t</td>
</tr>
<tr>
<td>-12.929</td>
<td>2 F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F-value</th>
<th>Cl</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>CL</td>
</tr>
</tbody>
</table>

CHAPTER 2

1. XY
2. XY
3. =
7. XY
8. XY
9. =

{ data

1.5.5 XY
8.6 XY
3.1 =
7.5 XY
4.6 XY
1.8 =

1.000 X
2.000 Y
3.000 Z
7.000 X
8.000 Y
9.000 Z
1.500 X
8.600 Y
3.100 Z
7.500 X
4.600 Y
1.800 Z
Chi-Square

The chi-square statistic is calculated with

\[ \sum \frac{(f_o - f_e)^2}{f_e} \]

The sequence is as follows:

- enter observed frequency \((f_o)\)
- press \(\sum\)
- enter expected frequency \((f_e)\)
- press \(=\)
Observed and expected values are printed; the expected value remains in the entry register. The chi-square statistic,

\[ \chi^2 = \frac{\sum (f_o - f_e)^2}{f_e} \]

is accumulated in register 9 and the number of pairs is kept in register 7. (The machine also uses register 8 during calculation of the formula, so don't rely on scratch pad register 8 while you're using the chi-square function.)

Here's an example of chi-square:

- \( \Phi_n \) 0
- ADV
- 2 \( \Phi_n \) 8
- 4 =
- 3 \( \Phi_n \) 8
- 7 =
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>DELETE</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.000   o
8.000   e
5.000   o
12.000  e
66.000  o
13.000  e
66.000  o
13.000  e
6.000   o
13.000  e

Note that the tape skips a space after each set of observed and expected data. And the print symbols o and e make it pretty easy to see which is which.
CHAPTER 2

Now let's see what's in those registers:

- \[ n\text{-count} \]
  - \( RCL_n 7 \)  
  - 5.000  
  - \( \uparrow \)  
  - 7

- \( \chi\text{-square statistic} \)
  - \( RCL_n 9 \)  
  - 13.138  
  - \( \uparrow \)  
  - 9

**Permutations and Combinations**

\[ \binom{n}{r} \]

Calculates the number of permutations and combinations of \( n \) items taken \( r \) at a time. The sequence is:

1. Enter \( n \)
2. Press \( \binom{}{} \) 7
3. Enter \( r \)
4. Press \( \_ \)  

The values \( n \), \( r \) and the number of permutations are printed. The number of combinations is put into the second function register and may be recalled with \( \text{2ND} \) \( \binom{}{} \)  

The number of permutations (P) and the number of combinations (C) are calculated according to the following formulas:

- \( P = \frac{n!}{(n-r)!} \)
- \( C = \frac{n!}{r!(n-r)!} \)
CHAPTER 2

Let's do some:

8 \( \binom{n}{7} \) 8 items 8.000 n

1 \( \binom{n}{7} \) 1 at a time 1.000 r

permutations 8.000 P

2ND FUNC

combinations 8.000 F 2

ADV

8 \( \binom{n}{7} \) 8 items 8.000 n

2 \( \binom{n}{7} \) 2 at a time 2.000 r

permutations 56.000 P

2ND FUNC

combinations 28.000 F 2

ADV

8 \( \binom{n}{7} \) 8 items 8.000 n

5 \( \binom{n}{7} \) 5 at a time 5.000 r

permutations 6,720.000 P

2ND FUNC

combinations 56.000 F 2
CHAPTER 2

ADV

8 [Π] n 7

8 items

8.000 n

8 =

8 at a time

8.000 r

perms

40,320.000 P

2ND
FUNC

combinations

1.000 F 2

FACTORIAL

One last Additional Function is [Π] n 6, which calculates the factorial (!) of the number in the entry register. Both the number and its factorial will print. If the number is non-integer or negative, ERROR will occur. Factorials of numbers greater than 69 are beyond the capacity of the machine \(10^{98}\), and you'll get OVERFLOW if you attempt one. The factorial of zero will come up as 1.

OPTIONAL KEYS

The 445 keyboard is fitted with spaces for three optional keys. If you like, you can have them installed on your keyboard with appropriate labels.

The purpose of these keys is to provide direct keyboard access to specific locations in program memory. Then if you store a program
to perform a certain operation at one of these locations you can use it as part of your keyboard capability. These optional keys will be explained thoroughly in Chapter 4.
CHAPTER THREE

GREEK SHIPS AND OTHER PHENOMENA
One day a Phoenician admiral and part-time statistician was gazing out over the horizon watching his ships disappear on their way to war exercises in the Mediterranean when suddenly a staggering thought struck him. He had been speculating on how long it took for the ships to drop below the horizon and disappear. And then this thought wandered to the curvature of the earth, which, although well known to people who depended upon their knowledge of the sea, was still a matter of almost total ignorance to the land-bound populace. In any case, he was ruminating on the possibility that there might be a connection between the curvature of the earth and the length of time it took ships to disappear, when another possibility captured his imagination. Perhaps there was a relationship between the distance from the ship to the observer when the ship disappears over the horizon and the curvature of the earth! And, indeed, there was. Since the radius would be the same where the ship was as it is where you are, if you knew the height of the ship you could calculate the distance from the tip of the mast to the center of the earth. Therefore, you know two sides of a triangle with a right angle in it. Now, didn't that Pythagoras fellow say if you knew all that you could find the length of the remaining side of the triangle? Yes, he did. So the admiral drew this drawing:
and then he did this figuring:

\[ R^2 + d^2 = (R + h)^2 \]
\[ R^2 + h^2 = R^2 + 2Rh + h^2 \]
\[ d^2 = 2Rh + h^2 \]
\[ d = \sqrt{2Rh + h^2} \]

If \( h = 35.2 \text{ cubits} \) and \( R = 13.95 \text{ megacubits} \), then
\[ d = \sqrt{2 \times 13.95 \times 10^6 \times 35.2 + (35.2)^2} \]
And, after two nights and three days of hand calculation, our ship-commanding friend finally arrived at the distance of 31.34 kilocubits. By this time, of course, the ships in question had already conquered their imaginary enemy and were on their way back to port.

If our admiral had had a 445, he could have solved the problem like this:

\[ d = \sqrt{2 \times 13.95 \times 10^6 \times 35.2 + (35.2)^2} \]
Now, the admiral has a fleet of ships with a whole bunch of different mast heights. And he wants to know how fast they're going on the way to battle. He knows when each one disappears by looking at his watch. And he can calculate the distances as above. But the statistician in him craves to know the average speed of all the ships in the fleet and the standard deviation of that speed. With a pencil and paper, this will take the admiral some time. But you can do it quite easily on the 445.

You don't really want to calculate each ship separately from beginning to end. After all, you've got a workable formula for the distance, the speed is simply the distance divided by the time and you know what the mast heights and times are. There has to be an easier way to deal with this information. And there is. You can organize your solution into a logical sequence of steps and program the 445 to do all the calculation for you. All you have to give the machine for each ship's calculation is the height of the ship from the water line to the tip of the mast, and the time it took to disappear over the horizon (we'll use $2\frac{1}{2}$ hours). Such a series might look like this:

\[
\begin{align*}
2 & \\
X & \\
1 & \\
3 & \\
. & \\
9 & \\
5 & \\
\text{EXP} & \\
6 & \\
\end{align*}
\]

Now, if you followed this list of steps sequentially on the keyboard, you'd wind up with the right answer (12,535.26 cubits per hour).

But to get the machine to do it all without any help, you have to put in a few more steps, and
a little more information about what to do with certain pieces of data as it goes along.

Don't forget that the programmable part of the 445 is exactly like a computer. (In fact it is a computer.) And computers can't think for themselves. You have to tell them exactly what to do at every step of the way.

So, now let's go back and organize our sequence into a series of steps for programming the 445.

It's good to write down our program steps in the order in which we want to load them into the machine. The height of the mast will be in the entry register when the program starts (because we will have put it there), so the first thing we want the machine to do is print that number. Therefore, the first instruction is PRINT ENTRY. Now, the program doesn't really need this number at this point, so we'll store it away in a register where it will be available later. Therefore, the next two instructions will be $ST_n$ and 0. This will put the number in register 0. Now we have to enter the time. So we program a HALT where we can enter it. Then PRINT ENTRY to print the time. And $ST_n$ and 1
to store the time away for later use, as we did with the mast height. From here on, the program steps are the same as our sequential list, except that instead of 35.2 (the ship height in our example) we want whatever number we've got stored away in register 0, and for time we'll use the contents of register 1. The whole list of program steps, so far, looks like this:

```
PRINT ENTRY
STn
0
HALT
PRINT ENTRY
STn
1
2
X
1
3
.
9
5
EXP
6
```
At this point in the program, our answer will be in the entry register. So now we want the answer printed. Our next instruction, then, is PRINT ANSWER. Then we're finished with that operation. But we know that we're going to be doing a whole series of these calculations.
We've got a HALT to enter the time. But when do we enter the next ship's height? Right now. Our next instruction, then, is HALT -- which simply tells the machine to stop and let us put something in or do something else before it goes on to the next instruction. So when we're running the program later, we'll put the next ship's height in there and it will be recorded in the entry register. But wait -- our first instruction is PRINT ENTRY. This means that if we run off a whole string of ships' heights, the heights and answers will be printed one after another on the tape without any spaces between them. That could be difficult to read. So let's put in a paper advance instruction and then a HALT instruction. Then we have to tell the machine what to do next. What we want to do is go back to the beginning of our program so we can do the calculation starting with the new number that will be in the entry register. To get there, we have to know the address where the program begins. Actually, we can start loading the program anywhere. But unless there's a reason to do it differently, programs are usually started at the beginning of program memory. So let's put ours there. The first address in program memory is 0000. To tell the machine we want to go to this address we use one of the two keys on the left side of the upper row, the JUMP or BRANCH keys, followed by 0 0 . (The difference between BRANCH and JUMP will be explored thoroughly in Chapter 4.) Where we put our program is called the program address -- since the program literally lives there.
We know our program will start at 0000, so we can end our program by sending the machine back there. Our last three instructions, then, will be \texttt{JUMP}_{nn}, 0 and 0.

Now let's load our program and see if it actually works. First we have to get to the address where we're going to load it. So press \texttt{JUMP}_{nn} 0 0. Then we have to tell the machine to load what we give it. So put the RUN-STEP-LOAD switch on LOAD. With this switch in LOAD position, everything you enter will be loaded into program memory in sequential order starting at the address where you start (in this case, 00). Let's put in our program.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Address</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRINT</td>
<td>0000</td>
<td>060</td>
</tr>
<tr>
<td>ENTRY</td>
<td>0001</td>
<td>110</td>
</tr>
<tr>
<td>( S T_n )</td>
<td>0002</td>
<td>000</td>
</tr>
<tr>
<td>0</td>
<td>0003</td>
<td>056</td>
</tr>
<tr>
<td>HALT</td>
<td>0004</td>
<td>060</td>
</tr>
<tr>
<td>PRINT</td>
<td>0005</td>
<td>110</td>
</tr>
<tr>
<td>ENTRY</td>
<td>0006</td>
<td>001</td>
</tr>
<tr>
<td>( S T_n )</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Now put the RUN-STEP-LOAD switch back on RUN.

As you probably noticed, the 445 printed out a symbol in the right-hand column for just about everything you programmed into it. In the middle column are the numeric codes of each key you used. And the left-hand column contains the STEP number of each instruction in your program. So you can look at the tape and review your program to see if you put everything in that you wanted to. Just compare the tape to your program list. Everything in your program except PRINT ENTRY, EXP, ADV and HALT will have a symbol in the very right-hand column to identify it. If everything is the same, you're ready to run the program. But first you have to put the PRINT switch on OFF. Otherwise, the 445 will print everything it does while it's working on the program. And we really aren't interested in all of that. We just want to know what numbers we put in and what the final answer is.

We've already put the RUN-STEP-LOAD switch on RUN so we don't have to worry about that. (Remember, everything you do while the RUN-STEP-LOAD switch is on LOAD is recorded in program memory.)

Let's enter our old ship's height with a time of $2\frac{1}{2}$ hours, and see if we
get the same answer we got before. Enter 35.2. Press RESUME.

Enter 2.5. Then all you have to do to initiate your program is press RESUME again.

3 5 2 RESUME

2 5 RESUME

35.20
2.50

12,535.26 A

After printing out the answer, it'll stop. That's because we put a HALT there. Now we can enter the next ship's height -- say, 43.7 cubits. Then press RESUME and enter the time, say 3 hours.

Now press RESUME again and the 445 will take off and do everything you've programmed it to do all over again. This time we get 11,639.16 cubits per hour. And you can keep on entering numbers and pressing RESUME until you run out of ships or your forefinger falls off.

Now let's suppose you had a more difficult problem. Let's say you want to calculate the expansion,

\[ \sin X = X - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \ldots \]

Let's also say that all we need is the result of the first three terms in the expansion.
Here's how you do it.

First, of course, we have to tell the 445 where to put our program.

So we hit \textbf{JUMP} \[0 \ 0\]. Now we put the \textbf{RUN-STEP-LOAD}

switch on LOAD and begin entering our program.

\begin{verbatim}
PRINT ENTRY
STn \{ Store X
0
STn \} \{ Start collecting values for final answer
1
ax
3
\div
X^3 / 3!
3
Tn
6
=  \\
\end{verbatim}

\begin{tabular}{|l|}
\hline
0000 & 060 \\
0001 & 110 \downarrow \\
0002 & 000 \\
0003 & 110 \downarrow \\
0004 & 001 1 \\
0005 & 025 \ ax \\
0006 & 003 3 \\
0007 & 024 \div \\
0008 & 003 3 \\
0009 & 116 \deq \\
0010 & 006 6 \\
0011 & 020 = \\
\hline
\end{tabular}
\[ \text{ST}_n \] Subtract \( \frac{x^3}{3!} \) from answer

\[ \text{RCL}_n \]

\[ 0 \]

\[ a^n \]

\[ 5 \]

\[ \div \]

\[ 5 \]

\[ \Phi_n \]

\[ 6 \]

\[ = \]

\[ \text{ST}_n \] Add \( \frac{x^5}{5!} \) to answer

\[
\begin{array}{ccc}
0012 & 110 & \downarrow \\
0013 & 022 & - \\
0014 & 001 & 1 \\
0015 & 111 & \uparrow \\
0016 & 000 & 0 \\
0017 & 025 & a^n \\
0018 & 005 & 5 \\
0019 & 024 & \div \\
0020 & 005 & 5 \\
0021 & 116 & \Phi \\
0022 & 006 & 6 \\
0023 & 020 & = \\
0024 & 110 & \downarrow \\
0025 & 021 & + \\
0026 & 001 & 1 \\
\end{array}
\]
Now move the switch from LOAD back to RUN.

We put in PRINT ENTRY instead of PRINT ANSWER at the end because with the kind of numbers we're calculating, we really don't want a rounded answer. For the same reason, we'd better set the decimal point for five places before we do any calculating. Let's put in .1, .2 and .3 as our values for X (in radians) and see how our program works.
You can see if your program is working by calculating the values on the keyboard according to the formula.

Now, let's add a bit of complexity to the program. Let's suppose that you want to continue calculating terms in the expansion for \( n \) terms. Therefore, you have to have a way of entering \( n \), and of testing for \( n \). A good way to start putting together a program with this degree of interdependence is to chart the flow of operations the machine will have to perform. This allows you to figure out and see clearly exactly how each element in the program relates to the other elements, and in what order they have to take place. This technique is called flowcharting, and the chart itself is called a programming flowchart. There are all kinds of different symbols that computer programmers use to indicate every little detail of their system, but for our little machine we only need two flowcharting symbols. One is a simple rectangle, in which you put operations. The other is a diamond shape, in which you put decisions. That's all you need for a flowchart. Let's make one.
Flowcharts generally run from top to bottom, left to right.

As you can see, we've set up two arbitrary values, $m$ and $k$, which we use to increment $X$ and to alternate the sign of each new term. We've also set up a place where we store $n$. After doing the calculations we decrement $n$ and then test it to see if it's zero. If it is, then we've done all the terms we wanted to, and we can print the answer. If not, we just go back and calculate another term, decrement $n$ and test again.

Each block in the flowchart here has a number. You can see how we carry out each of these elements in the actual program by studying the steps next to the corresponding number. Note that we're now using a loop to calculate the terms instead of doing them sequentially.
OK. We've got a flowchart. Now let's take a look at the actual program as we load it into the machine. (Don't forget to hit \( \text{JUMP } \boxed{\text{nn}} \), and to put the machine in LOAD mode.)

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RESET</td>
</tr>
<tr>
<td>1</td>
<td>( \text{ST}_n ) () 9</td>
</tr>
<tr>
<td></td>
<td>Here, the zeroes which ( \text{RESET} ) puts in the entry register are stored in register 9. The answer will be accumulated here.</td>
</tr>
<tr>
<td>2</td>
<td>HALT</td>
</tr>
<tr>
<td></td>
<td>PRINT ENTRY</td>
</tr>
<tr>
<td></td>
<td>( \text{ST}_n ) () 8</td>
</tr>
<tr>
<td></td>
<td>HALT to enter ( X ), then print it and store it away.</td>
</tr>
<tr>
<td>3</td>
<td>HALT</td>
</tr>
<tr>
<td></td>
<td>PRINT ENTRY</td>
</tr>
<tr>
<td></td>
<td>( \text{ST}_n ) () 7</td>
</tr>
<tr>
<td></td>
<td>HALT to enter ( n ), then print it and store it away.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>062</td>
<td>( \wedge )</td>
</tr>
<tr>
<td>0001</td>
<td>110</td>
<td>( \downarrow )</td>
</tr>
<tr>
<td>0002</td>
<td>011</td>
<td>9</td>
</tr>
<tr>
<td>0003</td>
<td>056</td>
<td></td>
</tr>
<tr>
<td>0004</td>
<td>060</td>
<td></td>
</tr>
<tr>
<td>0005</td>
<td>110</td>
<td>( \downarrow )</td>
</tr>
<tr>
<td>0006</td>
<td>010</td>
<td>8</td>
</tr>
<tr>
<td>0007</td>
<td>056</td>
<td></td>
</tr>
<tr>
<td>0008</td>
<td>060</td>
<td></td>
</tr>
<tr>
<td>0009</td>
<td>110</td>
<td>( \downarrow )</td>
</tr>
<tr>
<td>0010</td>
<td>007</td>
<td>7</td>
</tr>
</tbody>
</table>
4. establish \( m = 1 \)

5. establish \( k = 1 \)

6. This establishes the symbolic address "$\sqrt{\ }$" for this point in the program.

7. Calculate \( \frac{x^m}{m!} \cdot x \cdot k \)

\[
\begin{array}{c}
0011 & 001 & 1 \\
0012 & 110 & \downarrow \\
0013 & 006 & 6 \\
0014 & 110 & \downarrow \\
0015 & 005 & 5 \\
0016 & 066 & \\
0017 & 055 & \sqrt{\ } \\
0018 & 111 & \uparrow \\
0019 & 010 & 8 \\
0020 & 025 & ax \\
0021 & 111 & \uparrow \\
0022 & 006 & 6 \\
0023 & 024 & \div \\
0024 & 111 & \uparrow \\
0025 & 006 & 6 \\
\end{array}
\]
Calculate
\[ \frac{x^m}{m!} \times k \]

(continued)

add it to answer

increment m by 2
CHAPTER 3

Multiply k by -1. (Note that -1 remains in the entry register.)

Decrement n by adding the -1 in the entry register to it.

Recall n for testing.

JUMP

if positive

to symbolic address

" \sqrt{ \) " Otherwise,
Recall answer, print it and advance paper.

Go back to "START."

(Don't forget to put your machine back in RUN now.)

There's a new trick here. We used \texttt{I} \texttt{SYM} \texttt{DB} to address a location in a program by means of a symbolic address instead of a numeric code address. We'll go into that in greater detail in Chapter 4. But now let's run our program.

First push \texttt{RESUME} to get to the first HALT. Here's where you enter X. We'll use 0.3 for X. Now push \texttt{RESUME} again to get to the second HALT instruction. Here's where you put in n. We'll use 1 for n in the first example. Now press \texttt{RESUME} again. The 445
will run the program with these values and print the result. Here's a sample tape of this program using the values indicated:

\[
\begin{array}{ccc}
\text{Term} & \text{Answer} \\
0.3 & 1.00000 \\
0.3 & 0.30000 \\
0.3 & 3.00000 \\
0.2 & 4.00000 \\
0.2 & 0.19866 \\
\end{array}
\]

Now let's go on and get into a further level of complexity. Here we shall set up a program that does the same calculation, but allows us to test for two different conditions, either of which will serve as a signal to stop calculating additional terms. We'll tell the machine to stop when either a) it has calculated \(n\) terms or b) the last term calculated divided by the answer falls below a certain value. And we will make both \(n\) and the value of \(\frac{\text{Term}}{\text{Answer}}\) variable.

Here's the flowchart:
START

1. answer = 0
2. enter X and store
3. enter limit \( \frac{T}{A} \) and store
4. enter n and store
5. \( m = 1 \)
6. \( k = 1 \)
7. (symbolic address "Ln/Log")
8. calculate
   \[
   \frac{X^n}{m!} \times k
   \]
   and add to answer
9. calculate relative size of last term
10. NO
11. limit?
12. add 2 to m
13. change sign of k
14. decrement n
15. recall n
16. NO
17. (symbolic address "e^x/10^x")
18. recall and print answer
19. go back to START

Register Usage Chart

<table>
<thead>
<tr>
<th>Variable</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>answer ((\sin x))</td>
<td>00</td>
</tr>
<tr>
<td>X</td>
<td>01</td>
</tr>
<tr>
<td>T/A</td>
<td>02</td>
</tr>
<tr>
<td>n</td>
<td>03</td>
</tr>
<tr>
<td>m</td>
<td>04</td>
</tr>
<tr>
<td>k</td>
<td>05</td>
</tr>
</tbody>
</table>
And here's the program as it would be entered:

1. **RESET**
   - The zeroes which RESET puts in the entry register are stored in register 00. This is where the answer will be accumulated.
   - Accumulate.

2. **ST nn**
   - Enter, print and store X.
   - Enter, print and store limit.

3. **PRINT ENTRY**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>062</td>
<td>^</td>
</tr>
<tr>
<td>0001</td>
<td>120</td>
<td>↓</td>
</tr>
<tr>
<td>0002</td>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>0003</td>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>0004</td>
<td>056</td>
<td></td>
</tr>
<tr>
<td>0005</td>
<td>060</td>
<td></td>
</tr>
<tr>
<td>0006</td>
<td>120</td>
<td>↓</td>
</tr>
<tr>
<td>0007</td>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>0008</td>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>0009</td>
<td>056</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>060</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>120</td>
<td>↓</td>
</tr>
<tr>
<td>0012</td>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>0013</td>
<td>002</td>
<td>2</td>
</tr>
</tbody>
</table>
Enter, print and store n.

establishes m = 1

establishes k = 1

Establishes symbolic address "Ln/LOG" for this point in the program.
CHAPTER 3

Calculate:

\[
\frac{x^m}{m!} \times k
\]
CHAPTER 3

Calculate
\[ \frac{x^m}{m!} \times k \]

(continued)

add above term to answer (note that this value will remain in the entry register.)

*These steps (ENTER CODE nnn and 045) enter directly the numeric code of a programming function not accessible from the keyboard. This function is absolute value, and is used here to avoid the necessity of dealing with the sign of the number.
CHAPTER 3

RCL
nn

0

2

= (continued)

JUMP
nn

if negative

S
I
Y
N
M
D
B

"e^x/10^x"

Otherwise,

ST
nn

+ increment

m by 2

0

4

0057 121 \uparrow
0058 000 0
0059 002 2
0060 020 =
0061 126 Ju
0062 022 -
0063 067
0064 051 lg^{-1}
0065 002 2
0066 120 \downarrow
0067 021 +
0068 000 0
0069 004 4
CHAPTER 3

CHG SIGN

1

ST nn

change sign of k

X

0

5

ST nn

decrement n by 1 (-1 was left in the entry register by the previous operation)

+ 0

3

RCL nn

recall n

0

3

0070 013 -

0071 001 1

0072 120 ↓

0073 023 X

0074 000 0

0075 005 5

0076 120 ↓

0077 021 +

0078 000 0

0079 003 3

0080 121 ↑

0081 000 0

0082 003 3
CHAPTER 3

JUMP nn

if zero

SYM DB

to symbolic address

"e^x/10x"

Otherwise,

JUMP nn

to symbolic address

"Ln/LOG"

define ending point

print answer and advance tape

0083 126  Ju

0084 020  =

0085 067

0086 051  1g^{-1}

0087 126  Ju

0088 067

0089 050  1g

0090 066

0091 051  1g^{-1}

0092 121  →

0093 000  0
Let's do a couple of computations on our program to see if it works.

First, let's compute:

\[ X = 1 \]

\[ \text{limit} = 1 \times 10^{-20} \]

\[ n = 3 \]

<table>
<thead>
<tr>
<th>Set D.P.</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESUME</td>
<td></td>
</tr>
</tbody>
</table>

(to get to the first `HALT`)

1. **RESUME**

2. **EXP** -2 0 **RESUME**

3. **RESUME**

| 0094  | 000  | 0 |
| 0095  | 060  |   |
| 0096  | 065  |   |
| 0097  | 126  | Ju|
| 0098  | 000  | 0 |
| 0099  | 000  | 0 |

| 1.000000000  | 00   |
| 1.000000000  | -20  |
| 3.000000000  | 00   |
| 8.416666666  | -01  |
CHAPTER 3

This time, the computation stopped because of the \( n \) test. Now let's do this one:

\[
X = 1 \\
\text{limit} = 1 \times 10^{-3} \\
n = 10
\]

This time, the computation stopped because of the term/answer limit.

If you lived through this last example, you should begin to get a feeling for how the 445 works. And you should begin to wonder about some of the curious tricks and subtleties of which the machine is capable. And if you really followed the last example, you should be ready to explore the further reaches of programming technique. In any case, the next chapter covers the intricate and the obvious with regard to programming the 445.
CHAPTER FOUR

MORE ABOUT PROGRAMMING
Programming is nothing more than automatic key pushing. It is not a mysterious branch of magic which allows some people with inside information to hold exclusive domain over inanimate objects. Therefore, if you know how to tell a computer (or computer-like machine) how to do something, it'll do it. No questions asked. The 445, as we've discovered, is a little computer. Therefore, if you know how to tell it, it'll do anything for you that it's capable of doing. And there are quite a number of things it can do. You can program it to do anything for you that you can do on the keyboard. And, equally important, you can tell it by programming to do some very powerful things that you can't do on the keyboard. And that's really where this chapter comes in very handy. We are now going to go through the programming part of the 445 in very much the same kind of way as we went through the keyboard in Chapter 2. The first thing we have to do is find out where things get put when we tell the 445 what to do.

MEMORY

There are three separate memories on the 445 that you can get to directly. There are the ten scratch pad registers, which is one kind of quick-access memory. And there are up to 512 main data registers which you can also access directly and use exactly like the scratch pad registers. And also, completely separate from the scratch pad
and main data registers, there is program memory. And that's what we'll be talking about in this chapter. Let's take a look at a map of the 445's memory setup.

As you can see from the memory map, the 445's memory is divided into four separate sections, called "pages". Page 1 is where you store your numbers when you use the 512 main data registers you can get to from the keyboard. Pages 2 and 3 make up the area where the 445 has its operational memory and where it does its own figuring. That's also where the scratch pad registers are. Page 0, as it says on the map, is for program storage. The 445 stores each program instruction as a three-digit numeric code. (Later we'll discuss how you can enter these codes directly if you want to.) Program memory contains up to 4096 instruction locations. Each of these locations can hold
one instruction. Therefore, you could conceivably write a program with up to 4096 instructions and load it into your 445. As in the other memories, each data location has an address. These addresses are numbered 0000 through 4095, and are accessible in a number of ways. OK. That's memory. Now let's talk about how we get there.

**DIRECT ADDRESSING**

Program instruction addresses (also called program steps) are numbered decimally from 0000 through 4095. There are two ways to get to these steps. One way is SYMBOLIC ADDRESSING, which we'll get to later on. The other way, which we'll talk about here, is DIRECT (ABSOLUTE) ADDRESSING. Every tenth program step (i.e. step 0000, 0010, 0020, 0030, etc.) is a branch point, and may be addressed directly with a BRANCH or JUMP instruction. Program steps between branch points may not be addressed directly with BRANCH and JUMP alone. Since this is true, the numeric codes of branch points are shortened for access by removing the last digit -- which is always zero anyway. In addition, the first digit of these program instruction addresses is dropped if it's a lowly zero, and replaced with a non-numeric code if it's 1, 2 or 3. What's left is a two-digit numeric code, which we do use for addressing. So program step 0000, which is the first branch point, becomes branch point 00. Program step 0050, which is the sixth branch point, is branch point 05. And
so on. Each BRANCH or JUMP instruction for direct addressing, then, must be followed by a two-digit address for branch points 000 through 099, and a non-numeric code plus a two-digit address for all other branch points. Using a non-numeric code after the BRANCH or JUMP instruction to indicate hundreds, two-hundreds and three-hundreds before entering a two-digit numeric code for the rest of the address is the same sneaky method we used for the 512 main data registers. In fact, we use the same sneaky non-numeric codes. For hundreds (branch points 100 to 199) we use the DECIMAL POINT. For two-hundreds (branch points 200 to 299) we use CHANGE SIGN. For three-hundreds (branch points 300 to 399) we use EXPONENT. Branch points beyond 399 may not be addressed by means of BRANCH and JUMP.

(Don't be confused by the numbering of branch points and program steps. Just remember that branch point numbers are program step numbers from which we have mercilessly removed the final zero and non-numericked the first digit.)

BRANCHING AND JUMPING

BRANCH and JUMP can be used interchangeably in many situations. But there is one major difference which must be considered. BRANCH will take you to where you designate and store in its own private place the address where you were when you branched. Then, if there's a
RESUME after the machine has done whatever you told it to do where you branched, it will return to the program step immediately following the BRANCH instruction. Therefore, BRANCH is used to enter subroutines, since you can come out of the subroutine and return directly to where you came from. JUMP, however, merely takes you to the address you indicate and drops you like a hot potato. It couldn't care less what you want to do next. If you keep this difference in mind, you shouldn't get into any trouble with BRANCH and JUMP. (There's more about subroutines later on.)

Here's a diagram of how BRANCH and JUMP work:
CHAPTER 4  -131-

PRINTING

The 445 will print everything you enter while it's in LOAD mode.
(And this printed record is a marvelous way of debugging and checking
your program before running it.) So it doesn't matter how the PRINT
switch is set while you're loading in a program. But generally you
don't want the machine to print out all the intermediate steps while
it's running a program. And if you have the PRINT switch ON while
running your program, the machine will print out every little thing
it does. So remember to include in your program PRINT ENTRY and
PRINT ANSWER instructions when you want to have a number printed
so you can leave the PRINT switch on OFF while running your program.
As you may remember from Chapter 3, PRINT ENTRY will print
whatever is in the entry register, without rounding off, while PRINT
ANSWER will round off what's in the entry register. This becomes
extremely important when you're dealing with very small values, where
you might want to use PRINT ENTRY to print the final answer rather
than PRINT ANSWER to preserve maximum accuracy out to the last
digit printed.

LOADING THE PROGRAM

As you know, to load in a program, you put the machine in LOAD
mode. That is, you put the RUN-STEP-LOAD switch on LOAD. When
you do this, the machine will store consecutively in program memory
every key pushed. And it'll print out a record of the program you've entered. Let's take a look at that printed record and find out just exactly what all those numbers mean. To do that, let's load in a simple program and then see how the tape corresponds with what we've done. So push \textbf{BRANCH} \[0 \ 0\] to get to the first branch point and put the machine on LOAD. Now enter this program:

\begin{verbatim}
PRINT ENTRY

X

3

= 

PRINT ANS

HALT

JUMP nn

0

0

0000 060

0001 023 X

0002 003 3

0003 020 =

0004 060

0005 056

0006 126 Ju

0007 000 0

0008 000 0
\end{verbatim}
Now take your machine off LOAD and put it on RUN. (It's a good idea to get into the habit of ending every program entry by taking the machine out of LOAD. That way you avoid inadvertently doing horrible things to your program while looking at the tape or talking to your secretary. You can always put it back if you need to.)

Now let's take a look at that tape. The left-hand column of numbers, the four-digit ones, is your program step number list. Each step in the program has an identifying number, starting with 0000. (This comes in handy when you're interrupted while loading a 396-step program and you forget where you were.) The next column, in the middle, contains the numeric codes of each step in the program. Every operation the 445 can perform has a three-digit code name. The code name of the numeral 2, for instance, is 002. The code name of = is 020. The last column, on the right-hand side, contains the symbol for every operation in your program that has a symbol. PRINT ENTRY and HALT, you'll notice, have no symbols. But JUMP, X and the numerals do. Most computational keys have symbols, and most purely clerical keys don't. There's a complete list in the Appendix. Now let's consider another way of getting to individual program addresses, this time using the symbols we've been talking about.

**SYMBOLIC ADDRESSING**

You can name any step in your program with a symbolic name. And
then you can JUMP or BRANCH directly to that location from any other location by means of the symbolic address. It makes no difference whether the numeric address of that program step is at a branch point or not. Symbolic addressing is entirely independent of branch points. You name the step by using the INDIRECT/SYMBOL key and any one of 95 symbols available. You can use as a symbol any function represented by octal code numbers 000 through 137, with the exception of 066, which is a stick-in-the-mud. (In case you've forgotten, octal numbers have no 8's or 9's.) Many of these numbers represent key codes, and may be put into your program just by pressing the key. Those which do not represent keys are entered by means of their code numbers and the ENTER CODE key. Let's say you want to give a place in your program the symbolic address "\( \sqrt{ } \). You do it by loading into the program \( I \) \( S \) \( N \) \( Y \) \( M \) \( D \) \( B \) and then \( \sqrt{ } \).

The point in the program which is defined by this symbol is the step immediately following the \( \sqrt{ } \). To make the program go there, all you have to do is load in \( \text{BRANCH} \) \( nn \) (or \( \text{JUMP} \) \( nn \)) then \( I \) \( S \) \( N \) \( Y \) \( M \) \( D \) \( B \) and \( \sqrt{ } \). Let's do one.
this gives the next instruction the name "\( \sqrt{\} \)"

Here, we used the symbol to define the beginning of the program, and jumped back there at the end.
SYMBOLIC ADDRESS TABLE

The machine uses a table to keep track of the locations of symbols in program memory. When you name an address in program memory with a symbol, the machine puts the address of the symbol next to the symbol slot in its little table. And when you BRANCH or JUMP to that symbolic address, the machine doesn't have to look all over program memory to find it. It just takes a quick glance at its little table and goes directly to the address in program memory that it finds next to the symbol in the table. Therefore, your programs will work just as fast no matter where you put them in program memory (since the machine doesn't have to keep running through all of program memory to find things). Also, when you INSERT, the machine automatically updates the table. And when you read in a magnetic card (see Chapter 6), any symbols in the program(s) there are also recorded in the table -- so you can load in a symbolically addressed program anyplace in program memory.

ENTER CODE

If you want to name a place in your program symbolically with a code for which there is no key (and there are many) or if you want to load into your program a function for which there is no key on the keyboard (see numeric code list in the Appendix), you have to use the ENTER CODE key. To enter a code, all you have to do is press and then the numeral keys for the code. (We've already done this
CHAPTER 4   -137-

once. Take a look at the program on page 120.) ENTER CODE is the only key on the keyboard which calls for a three-digit numeric entry. That makes it easy to remember that it's for three-digit numeric codes only.

If you had wanted to use numeric code 122 instead of \(\sqrt{122}\) (numeric code 055, as you can see on your tape) for your symbolic address in the example on page 135, you could have done it like this:

<table>
<thead>
<tr>
<th>S</th>
<th>Y</th>
<th>N</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>N</td>
<td>D</td>
<td>B</td>
</tr>
</tbody>
</table>

```
ENTER CODE nnn

PRINT ENTRY

X

3

=

PRINT ANS

HALT

0000  066

0001  122

0002  060

0003  023  X

0004  003  3

0005  020  =

0006  061  A

0007  056
```
As you can see from the tapes, the machine doesn't care whether you press a key or enter the numeric code. Either way, the octal code of the function you've entered will be stored in program memory.

You can BRANCH or JUMP from the keyboard to any symbolic address in a program for which there is a key on the keyboard. You do this by pressing \texttt{BRANCH} or \texttt{JUMP} then \texttt{I\textasciitilde S Y N D M B} and the key which names the address. But you cannot use \texttt{ENTER CODE} to get from the keyboard to a symbolic address in a program for which there is no key. \texttt{ENTER CODE} only loads numeric codes into a program -- it does not give you keyboard access to locations in program memory. Only your program can access symbolic addresses for which there are no keys on the keyboard. Similarly, you cannot use \texttt{ENTER CODE} to activate a
function from the keyboard. You can only tell your program to activate this function by means of $\text{ENTER CODE } nnn$. To put it simply, $\text{ENTER CODE } nnn$ is in no way a keyboard operation key. It is strictly a programming key.

**BACK SPACE**

A convenient key to know about while loading your program is $\text{BACK SPACE}$. Every time you press the key, you move back a space. The key is operative only in LOAD mode, and is remarkably appropriate when you make an entry mistake in the middle of your program. All you do is back up and put in the correct instruction over the incorrect one. It replaces it. If your mistake consisted of leaving out a step, then you have to use INSERT. That's next. But first, let's have a small mistake to illustrate BACK SPACE. Suppose we made a mistake while entering the program on page 135.

```
  S  I  Y  M
  N  D  B
```

Use BACK SPACE to back up and correct the mistake.

```
  PRINT ENTRY
```

```
0000  066
0001  055
0002  060
```
Yes, it works just like BACK SPACE on your typewriter. Except our BACK SPACE leaves a record of what it did.

**INSERT**

Here's where you get to put in a program step (or steps) that you left
out the first time. Use BRANCH, JUMP, BACK SPACE or whatever
to get to the instruction just before the location where the insertion
is to take place. Then press \textbf{INSERT}. The address, code and
print symbol of the instruction \textit{before} the insert location (the one you
just got to) will be printed out, and the 445 will automatically move
every subsequent instruction in program memory down one address
until it gets to an empty space (a NOOP code), which it fills. Then it
stops moving instructions. Check the address on the tape to be sure
you're in the right place and then load in your new instruction. The
IDLE light will stay off while the machine is rearranging addresses,
and you may have to wait a second for it to come back on (if it has to
move a lot of instructions) before you can insert another instruction.
You have to press \textbf{INSERT} before each instruction you insert. If
you have more than one instruction to add, load them in the same order
you want them to appear in the program, pressing \textbf{INSERT} before
loading in each one. Symbolic addressing is not affected by insertion
because of the nifty little symbolic address table the machine keeps.
When you insert, the machine up-dates the table. But it's a good idea
to load several NOOP codes at the end of every program to keep the
programs separated by neutral steps. This will prevent any inaccuracy
in the symbolic address table due to insertion and consequent shifting
of program addresses. A NOOP code (it means "no operation") just
creates a \textit{very} empty space. The code is 377, and you load it by pushing
SUBROUTINES

A subroutine is simply a small program within another program. Subroutines allow you to avoid repeating an identical series of steps many times in the same program. If, for example, you have a set of instructions that perform a calculation which will take place with many items, or many times, you can set up those instructions as a subroutine -- and branch to the subroutine when you need it. This is where symbolic addressing becomes really handy, because with symbolic addresses you can load a series of subroutines one after another right after your main program. You can have subroutines within subroutines within subroutines... up to six levels. And remember, you BRANCH to subroutines, and BRANCH remembers where you were when you branched. RESUME gets you back there at the end of the subroutine. Let's have an example. Suppose you wanted to solve this formula:

\[ Y = \frac{\sqrt{1 + \ln A}}{\sqrt{1 + \ln B}} \cdot \frac{1}{\sqrt{1 + \ln C}} \]

Note that \( \sqrt{1 + \ln A} \), \( \sqrt{1 + \ln B} \) and \( \sqrt{1 + \ln C} \) all have the same format. Therefore, we can set up a subroutine to calculate all of them rather than calculating each one separately.
Here's the program for solving this formula:

(A is in register 0
B is in register 1
C is in register 2)

```
0660000 establishes beginning of program as symbol "SECOND FUNCTION"
0001 052 F
0002 111 ↑
0003 000 0
0004 127 Br
0005 067
0006 025 aX
0007 024 ÷
0008 111 ↑
0009 001 1
```
here, we could enter new values for A, B and C into registers 0, 1 and 2
CHAPTER 4

**BRANCH**
nn
-

**SYN**
Y
M
DB
-

goes back to beginning
of program
-

**2ND**
FUNC
-

**SYN**
Y
M
DB
-

subroutine starts
here
-

\( a^x \)

( )

**Ln**
LOG
-

calculates
\( \sqrt{1 + \ln X} \)

1

) \( \)

\( \sqrt{\) 

**RESUME**
 sends machine back
where it came from
-

<table>
<thead>
<tr>
<th>0022</th>
<th>127</th>
<th>Br</th>
</tr>
</thead>
<tbody>
<tr>
<td>0023</td>
<td>067</td>
<td></td>
</tr>
<tr>
<td>0024</td>
<td>052</td>
<td>F</td>
</tr>
<tr>
<td>0025</td>
<td>066</td>
<td></td>
</tr>
<tr>
<td>0026</td>
<td>025</td>
<td>ax</td>
</tr>
<tr>
<td>0027</td>
<td>026</td>
<td></td>
</tr>
<tr>
<td>0028</td>
<td>050</td>
<td>lg</td>
</tr>
<tr>
<td>0029</td>
<td>021</td>
<td>+</td>
</tr>
<tr>
<td>0030</td>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>0031</td>
<td>027</td>
<td></td>
</tr>
<tr>
<td>0032</td>
<td>055</td>
<td></td>
</tr>
<tr>
<td>0033</td>
<td>057</td>
<td></td>
</tr>
</tbody>
</table>
You can have as many subroutines as you have room for in program memory, as long as there is an individual address for each one. If you have several subroutines, symbolic addressing will really save you trouble -- and program memory space. You can just line them up after your program like dominoes.

**CONDITIONALS**

Conditionals are tests that you can build into your program. You can set up a test and have the program go one way if you get one answer, and another way if you get another answer. Each test (or condition) has two possibilities. You set it up to either BRANCH or JUMP if a condition is met. If the condition is not met, the program will just continue as if there were no test. (It's called "falling through.") When the condition is met, the program will BRANCH (or JUMP) to wherever the instruction tells it to. Here are the possible conditions on the 445 and how to set them up:

(BRANCH is used in these examples for convenience. But remember that JUMP can also be used in each case.)

**BRANCH** to branch point nn if the entry register is positive.

\[
\text{BRANCH } \begin{array}{c} nn \end{array} + \begin{array}{c} n \end{array} \begin{array}{c} n \end{array}
\]

**BRANCH** to branch point nn if the entry register is negative.

\[
\text{BRANCH } \begin{array}{c} nn \end{array} - \begin{array}{c} n \end{array} \begin{array}{c} n \end{array}
\]
BRANCH to branch point nn if the entry register is zero.

\[
\text{BRANCH} \quad nn = n \ n
\]

BRANCH to branch point nn if Flag 1 is set. (This is the FLAG on the keyboard.)

\[
\text{BRANCH} \quad nn \quad \text{FLAG} \ n \ n
\]

BRANCH to branch point nn if the OPTION switch is down.

\[
\text{BRANCH} \quad nn \quad X \ n \ n
\]

In each case, symbolic addressing can be used in place of \[ n \ n \].

So, the example

\[
\text{BRANCH} \quad nn \quad + \ n \ n
\]

could also be

\[
\text{BRANCH} \quad nn \quad + \ I \ Y \ M \ DB \ \sqrt{ }
\]

Also, remember that if RESUME is programmed at the end of the subroutine you branch to, the machine will return to the instruction directly following the second \[ n \] (or the symbol).

INDIRECT ADDRESSING

You may remember we said earlier that there was another use for the pointer register other than as an extra scratch pad register. This is it. Register \[ \ ] can also be used to hold the address of a main data register. (It can't be used for scratch pad registers.) You just enter
the register number you want held directly into the pointer register, say main data register 52, by

```
5 2 STn
```

Now anytime you want to access register 52, you can do so indirectly by means of the pointer register. Let's say you want to store 123.45 into register 52. Just go

```
1 2 3 . 4 5 STnn
```

after main data STORE, RECALL or EXCHANGE instructions automatically looks into the pointer register for an indirect address.

So 123.45 is now in register 52. To prove it, you can recall register 52 directly and see what's in it:

```
RCLnn 5 2
```

You can also recall or exchange the value in register 52 indirectly by

```
RECALLnn or EXCHANGE nn
```

INDIRECT/SYMBOL

INDIRECT/SYMBOL

Of course, you can still do all the normal arithmetic operations in and out of the pointer register. This may change the number there, and
thus the indirect address. Therefore, incrementing and decrementing
the address in the pointer register is extremely simple. You can ac-
access a series of registers in increasing or decreasing numerical se-
quence just by incrementing or decrementing the pointer and addressing
indirectly.

All of the operations that can be performed in a register by means of
direct addressing can also be performed in that register by means of
indirect addressing. If, for example, you wanted to add 46 to register
52, you just go

\[ 46.0000 + \]

Here's an example which shows the print symbols for arithmetic oper-
ations using indirect addressing.

\[ 39.0000 \downarrow \]

\[ 400.0000 \downarrow I \]

\[ 105.0000 \downarrow I I \]

\[ 505.0000 \uparrow 39 \]
Note that the print symbols are coded for arithmetic operations with indirect addressing: + is 1, - is 2, X is 3 and ÷ is 4.

SOME MECHANICAL NICETIES

Stepping Through Your Program

Bet you were wondering when we were going to get to that mysterious STEP on the RUN-STEP-LOAD switch. This is the place. With the switch on STEP, you can go through your program one step at a time. Each time you press RESUME, the address, code and print symbol for the next step are printed and the step is executed. If execution of
that step normally causes printing, then it prints. If the switch is returned to RUN, \texttt{RESUME} will cause the program to begin at the next step.

\textbf{Listing Your Program}

This doesn't execute anything, it just lists the address, code and print symbol for each step in your program. You can do it either one-step-at-a-time or continuously. With the switch on LOAD, press \texttt{LIST PROG} and release it immediately. This will print the address, code and symbol of the current location and stop. Press it again, and it'll do the same thing at the next location, and so on. If you hold \texttt{LIST PROG} down until the first line is printed, the entire program will be listed continuously. In fact, the entire program memory will be listed, so unless you just like to see it list, put it in \texttt{RUN} when you've seen enough. That'll just stop the listing. You can start it up again where it left off by putting it back in LOAD and pressing \texttt{LIST PROG} again. (You can also stop the \texttt{LIST PROG} process by pressing \texttt{HALT}.) But be sure to remember you're still in LOAD mode.

\textbf{Where Am I?}

If you ever wonder where you are when there's no program running, there's a way to find out. Any time the 445 is halted (IDLE light on and not flashing) the current program address can be determined by
putting the switch on either STEP or RUN and pushing LIST PROG.

The address, code and print symbol of the current location will print.

If you press LIST PROG again, the exact same line will print out again. And again. And again. The entry register is not changed, and no matter how many times you press LIST PROG, you'll stay in the same place. So now you know where you are. If the switch is on RUN and RESUME is now pressed, execution of the program will begin from the address printed.

Setting Flag 1

The FLAG key, which you'll find almost directly in the center of that top row, is a simple, direct way to set up a condition which your program can test. A program may test the flag, and conditionally BRANCH or JUMP depending on its setting. This keyboard flag is called Flag 1, and is changed only by the FLAG key and RESET Flag 1 instruction.

THE WONDERFUL WORLD OF ENTER CODE

We've already seen some of the things ENTER CODE nnn lets us do. There's more. If you press ENTER CODE nnn and then enter the code of the following functions, your program will do the marvelous things described below.
These programming functions and ENTER CODE work very much like the Additional Functions and \[\text{\text{.function}}\] on the keyboard. Let's take them one at a time.

**List-Mode Arithmetic**

List-mode arithmetic uses a separate accumulator register. This accumulator is altered only by the list-mode add, subtract and total functions. It is reset to zero by RESET.

**List-Mode Add — code 041**

This function adds the contents of the entry register to the list-mode accumulator. The entry register is not changed. The number in the entry register is printed.

**List-Mode Subtract — code 042**

This function subtracts the contents of the entry register from the list-mode accumulator. The entry register is not changed. The number in the entry register is printed.

**List-Mode Subtotal — code 043**

This function copies the accumulator into the entry register and prints it. The accumulator is not altered.

**List-Mode Total — code 040**

This function totals the accumulator, copies the total into the entry register and prints that number. The accumulator is reset to zero.
CHAPTER 4 -154-

Increment Entry — code 151
This instruction increments the entry register by one.

Decrement Entry — code 152
As you might expect, this instruction decrements the entry register by one.

Absolute Value — code 045
We've seen this one before. This instruction makes the sign of the entry register positive.

Square — code 053
This function calculates the square of the number in the entry register. Both the number and its square are printed.

Flag 2 — code 017
This instruction sets program Flag 2. A program may interrogate (test) this Flag and use its setting as a condition to branch or not branch. This works just like Flag 1, which you can set from the keyboard. A program for testing Flag 2 looks like this:

```
BRANCH
FLAG 2
  n
n
```
This translates as "BRANCH, if Flag 2 is set, to program address nn."

You can use Flag 2 the same way with JUMP.
Exchange Main Data — code 122

This function exchanges data between the entry register and a main data register using the same procedure that \[ \text{ST} \]\[ \text{nn} \] uses for storage.

Dot Print — code 176

This instruction causes a line of dots to be printed, regardless of the PRINT switch position.

Equals-Sum-Zero — code 037

Equals-Sum-Zero does two things. First, it acts exactly like equals, and will produce the same result with respect to any operation you have going as would be produced by a plain \[ = \]. But in addition, this function adds the resulting value of the entry register to scratch pad register 0. At any time, you can recall register 0 and put the accumulated total into the entry register. You can alternate between regular equals and Equals-Sum-Zero in your program all you like, keeping a running total of the values you want accumulated in one place. Remember, though, that each additional value will be added to what's already in register 0, it will not replace it. When you intend to use Equals-Sum-Zero in your program, it's a good idea to make sure register 0 is clear when you begin. There are several ways to do this, as you probably remember, including the direct procedure of storing zeroes in register 0 to begin with. The procedure you adopt in each case will depend upon what other registers you're using in your program, and whether you want to clear a series of registers or just one.
OPEN CHANNEL REGISTER ACCESS

This is another form of indirect addressing. It allows you to STORE, RECALL and EXCHANGE with a main data register by means of a fast, open channel to that register. Whenever you use the main data STORE, RECALL or EXCHANGE functions, the address of the register you've accessed is automatically kept in a separate hidden place. As long as you don't use main data STORE, RECALL or EXCHANGE instructions again, (or increment or decrement the address, see below) the address in that special place will remain the same. And there are codes for storing into the register with which this channel is open, recalling from it, and exchanging its contents with the entry register.

Register arithmetic is not available with this method -- just storing, recalling and exchanging. But you can increment and decrement the address stored in that special place, and therefore access a whole series of data locations by means of the open channel method. The advantage of doing it this way is that the open channel is actually "machine level" access, and is much faster than the normal addressing procedure. The codes for using the open channel method are as follows:

**Code 331** - Store the contents of the entry register into the register by the open channel pointer (the special place).

**Code 332** - Recall contents of the open channel register.
CHAPTER 4

Code 333 - Exchange the contents of the open channel register with the contents of the entry register.

Code 335 - Increment the address stored in the special place by 1.

Code 337 - Decrement the address stored in the special place by 1.

IDENTIFIER - Code 177

This code will cause the contents of the entry register to be printed on the very left-hand side of the tape as an identifier. Fractional digits will be included, with the decimal point in the correct position -- but trailing zeroes will be suppressed. In order to be used as an identifier, a number must have an absolute value of 1 or more.

There is a way of using the identifier code which allows an identifier to be printed while still saving the value of the entry register. If your program enters a number to be printed out as an identifier before executing code 177, that number will be printed and the former value of the entry register will be restored. If your program doesn't enter a new number before invoking code 177, the contents of the entry register will be printed as a very fine identifier, but you can't be sure of what will be left in the entry register. Only a numeral entry is considered a new number. Recalling a value from a register is not numeral entry.

Here's part of a program using 2 as an identifier:
CHAPTER 4 -158-  

<table>
<thead>
<tr>
<th>INSTRUCTION</th>
<th>CODE</th>
<th>EXPLANATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>π</td>
<td>015</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>023</td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>020</td>
<td>π² now in entry register</td>
</tr>
<tr>
<td>2</td>
<td>002</td>
<td>numeral entry</td>
</tr>
<tr>
<td>IDENT</td>
<td>177</td>
<td>causes 2 to be printed as an identifier</td>
</tr>
<tr>
<td>PRINT ENTRY</td>
<td>060</td>
<td>prints π², which has been restored to entry register.</td>
</tr>
</tbody>
</table>

With this sequence in your program, the number 2 (or any other number you use) will be printed on the left-hand side of the tape as an identifier and then abandoned. The value of the entry register just prior to your 2 (or other number) will be restored.

PRINT ENABLE - PRINT DISABLE

PRINT DISABLE (code 154) is an internal, programmable print switch. It tells the 445 not to print anything your program does except the "Print Always" functions (see Print Control, Chapter 3). PRINT DISABLE does not affect keyboard printing. It merely overrides the keyboard PRINT switch with respect to the program. Therefore, you can leave the keyboard PRINT switch ON, execute code 154 in your program, and have complete keyboard printing while your program has only "Print Always" printing capability.

PRINT ENABLE (code 155) reverses PRINT DISABLE and returns
printing control of the program as well as the keyboard to the keyboard
PRINT switch.

NOTE: If you find that your program won't print, and you don't know
why, it may be that someone has left the machine in PRINT DISABLE
mode. To remove it, just load the instructions

155
HALT

and then execute these instructions. This will return the machine to
normal PRINT control.

RECALL DP - code 157

This is another convenient device for the programming aspect of the
445. When you execute SET DP, either from the keyboard or from a
program, the previous decimal point setting is automatically saved.
Then, when you execute RECALL DP (code 157) from your program,
the decimal point setting is returned back to the previous setting.
Where this is particularly useful is when the same machine is used
for both programmed and keyboard operation. The programmer can
set a decimal point that gives him the degree of accuracy he wants in
his program (usually many more places than you'd want from the key-
board), and by entering code 157 at the end of his program, he returns
the machine to whatever setting it had before his program began.
ADD TO REGISTER

You can add the contents of the entry register directly into any main data register simply by entering code 123 and then the address of the main data register to which you'd like the value added.

You can also add the contents of the entry register directly into any scratch pad register by entering code 113 and the numeral name of the scratch pad register.

KEYBOARD PROGRAM ACCESS

In Chapter 2, under OPTIONAL KEYS, we mentioned that there were three keys you can have added to your keyboard which provide direct keyboard access to specific locations in program memory. Since these would be your keys, you can call them whatever you like. For convenience in discussing them, though, let's call them F₁, F₂ and F₃.

Pressing \([F₁]\) would take the machine to program memory step 495. If there was an instruction there, the machine would immediately begin executing instructions. \([F₂]\) would take you to program memory step 500. And \([F₃]\) would take you to program memory step 505. The purpose of these keys is to allow you to get from the keyboard directly to a program which performs some operation you don't want to do on the keyboard. With three keys, you can access three separate operations in this way. And, of course, you can change these operations any
time you want to. Since there are only five program steps at each of these locations in program memory, the logical thing to do is to use these five steps to BRANCH to some other location in program memory where you have the real program salted away.

Be sure there's a RESUME at the end of this program, so that you can return to the access area after the operation is completed. Since you got there with BRANCH, the RESUME will take you back where you came from. To return the machine to keyboard operation, there has to be a HALT instruction immediately following the BRANCH n n instructions in the access area.

If you want to use that same operation program as a subroutine in another program, just BRANCH to it from your main program. The RESUME at the end of the operation program will take you back where you branched from.
CHAPTER FIVE

EXAMPLES AND PROBLEMS
As the farmer said while he sat back and watched his wild new stallion chase his city cousin around the corral, "Showin's better'n tellin'." On that principle, this chapter consists of practical examples of how you'd go about solving certain kinds of problems on the 445. There are some pretty nifty techniques here, and it's not a bad idea to run through the examples on your own machine so you can really see how some of the subtler relationships are structured. By comparing your tape with our tape, you can also see what you did wrong if you decide to try a few out by yourself.

EXAMPLES

\[
\begin{array}{c}
25 \times 32 \\
\hline
1.21 \times 15.07
\end{array}
\]

25 \times 32

\[
\begin{array}{c}
25.000 \quad \times \\
32.000 \quad \div \\
1.210 \quad \div \\
15.070 \quad \div \\
43.872 \quad \ast
\end{array}
\]

OR

\[
\begin{array}{c}
25 \times \\
32 \div \\
( \\
800.000 \\
\end{array}
\]
\[ (1 + 0.07)^{15} \]

\[ 1 + \]

\[ 0.07 \times 1.000 = 2.759 \times 1.000 = 2.759 - 1.759 = 1 - (1 + 0.07)^{15} \]
\[
\frac{1 - (1 + .07)^{15}}{1 - .07}
\]

\[
\sqrt{3 + \frac{1}{17.3}}
\]

\[
3 \div 0.07 = 15 - 1.5
\]

\[
1 \div 0.07 = 15 - 15
\]

\[
0.930 \times 0.057 = -1.891
\]

\[
17.300 \div 0.057 = 1/x
\]
0.057 = 3.057 * 1.748 * log (3 + e^{1.2})

3.000 + 1.200 lg^{-1} 3.320 *

3.320 = 6.320 *

6.320 lg 1.843 *

0.800 F 2

Solve:

Q = \frac{2}{3} b - \sqrt{2g H^{3/7}}

g = 32.2

H = 16.27

b = [R_{107}]

2 \div 

3 \times

2/3b

2.000 

3.000 X

12.500 ↑ 07
\[
\begin{align*}
32.2 + 2x32.2 &= 64.4 \quad \text{(1)} \\
\sqrt{64.4} &= 8.024 \\
16.270 &= 221.018
\end{align*}
\]
CHAPTER 5 -168-

PROBLEMS

Problem 1

Given the following group of scores:

- 21.7
- 28.5
- 31.2
- 24.6
- 27.0
- 38.1
- 23.2

A. Compute the standard deviation, mean and standard error of the mean for the group.

\[
\begin{align*}
\text{SET} & \quad 1 \\
\text{D.P.} & \quad 3 \\
\text{GROUP} & \quad 1 \\
\bar{x} & \quad 0 \\
2 & \quad 1 & \quad 7 & \frac{\sum x^2}{n} \\
2 & \quad 8 & \quad 5 & \frac{\sum x^2}{n} \\
3 & \quad 1 & \quad 2 & \frac{\sum x^2}{n} \\
2 & \quad 4 & \quad 6 & \frac{\sum x^2}{n}
\end{align*}
\]

\[
\begin{align*}
0.000 & \quad \text{CL} \\
21.700 & \quad \Sigma \quad 1 \\
28.500 & \quad \Sigma \quad 1 \\
31.200 & \quad \Sigma \quad 1 \\
24.600 & \quad \Sigma \quad 1
\end{align*}
\]
B. Compute the Z-statistic and normal probability associated with each score.

\[ \frac{27.000}{\Sigma} 1 \]
\[ \frac{38.100}{\Sigma} 1 \]
\[ \frac{23.200}{\Sigma} 1 \]
\[ \frac{5.588}{SD_{n-1}} 1 \]
\[ \frac{27.757}{F_2} \]
\[ \frac{3.223}{F_3} \]

\[ \frac{21.700}{X} -1.170 Z_1 \]
\[ -1.170 P z * \]
\[ 0.120 \]

\[ \frac{28.500}{X} 0.143 Z_1 \]
\[ 0.143 P z * \]
\[ 0.557 \]

\[ \frac{31.200}{X} 0.665 Z_1 \]
\[ 0.665 P z * \]
\[ 0.747 \]
C. Delete the 38.1 and add seven 26.1's to the group.
D. Compute the new standard deviation, mean and standard error of the mean for the revised group.

\[
\begin{align*}
\text{SD} & \quad \text{MEAN} \\
26.069 & \quad \bar{x}_n = 3 \\
0.633 & \\
\end{align*}
\]

E. Compute the z-statistic and normal probability associated with each score in the new group.

\[
\begin{align*}
\text{Z} & \quad \text{P} \\
21.700 & \quad -1.991 \quad \bar{x}_n = 1 \\
-1.991 & \quad 0.023 \quad * \\
28.500 & \quad 1.107 \quad \bar{x}_n = 1 \\
1.107 & \quad 0.866 \quad * \\
31.200 & \quad 2.338 \quad \bar{x}_n = 1 \\
2.338 & \quad 0.990 \quad * \\
\end{align*}
\]
Problem 2

Here's an example of linear regression using transformed data. A two-variable regression is used in the example, but three-variable may be done the same way.

A. Do a linear regression using these raw data:
### CHAPTER 5

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.5</td>
</tr>
<tr>
<td>3.5</td>
<td>3.5</td>
</tr>
<tr>
<td>5.5</td>
<td>4.0</td>
</tr>
<tr>
<td>6.0</td>
<td>6.5</td>
</tr>
<tr>
<td>6.5</td>
<td>9.0</td>
</tr>
<tr>
<td>5.5</td>
<td>11.0</td>
</tr>
<tr>
<td>7.0</td>
<td>11.5</td>
</tr>
</tbody>
</table>

**SET D.P.**

| 3 |

**UNIT**

| 0 |

**XY**

<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5</td>
</tr>
</tbody>
</table>

| 3.5 |
| X  |

| 3.5 |
| Y  |

| 5.5 |
| X  |

| 4.000 |
| Y |

| 0.000 |
| CL |

| 1.000 |
| X  |

| 1.500 |
| Y  |

| 3.500 |
| X  |

| 3.500 |
| Y  |

| 5.500 |
| X  |

| 4.000 |
| Y  |
Therefore, $r = .805$ and $Y_{est} = 1.509X - .833$. 
B. Now let's do a regression on these data using Ln(Y) in place of Y.

<table>
<thead>
<tr>
<th>θ_n</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>XY</td>
</tr>
<tr>
<td>1.5</td>
<td>Ln LOG</td>
</tr>
<tr>
<td>=</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>3.5</td>
<td>Ln LOG</td>
</tr>
<tr>
<td>=</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>Ln LOG</td>
</tr>
<tr>
<td>=</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>XY</td>
</tr>
<tr>
<td>6.5</td>
<td>Ln LOG</td>
</tr>
<tr>
<td>=</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CL</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>1.000</td>
<td>X</td>
</tr>
<tr>
<td>1.500</td>
<td>lg</td>
</tr>
<tr>
<td>0.405</td>
<td>*</td>
</tr>
<tr>
<td>0.405</td>
<td>Y</td>
</tr>
<tr>
<td>3.500</td>
<td>X</td>
</tr>
<tr>
<td>3.500</td>
<td>lg</td>
</tr>
<tr>
<td>1.252</td>
<td>*</td>
</tr>
<tr>
<td>1.252</td>
<td>Y</td>
</tr>
<tr>
<td>5.500</td>
<td>X</td>
</tr>
<tr>
<td>4.000</td>
<td>lg</td>
</tr>
<tr>
<td>1.386</td>
<td>*</td>
</tr>
<tr>
<td>1.386</td>
<td>Y</td>
</tr>
<tr>
<td>6.000</td>
<td>X</td>
</tr>
<tr>
<td>6.500</td>
<td>lg</td>
</tr>
<tr>
<td>1.871</td>
<td>*</td>
</tr>
<tr>
<td>1.871</td>
<td>Y</td>
</tr>
</tbody>
</table>
CHAPTER 5

6.500  X
9.000  lg
2.197  *
2.197  Y

5.500  X
11.000  lg
2.397  *
2.397  Y

7.000  X
11.500  lg
2.442  *
2.442  Y

SET GROUP 1
LIN REG
2ND FUNC

0.913  LR 1
0.324  F 2
0.083  F 3
Here, \( r = .913 \) and \( \ln Y_{est} = .324X + .083 \).

\[ Y_{est} = \exp (.324X + .083) \]

You can use the normal functions on the 445 to transform any variable during XY data entry. If you want to use an algebraic expression to transform a variable, enclose the expression in parentheses, like this:

**enter X, \( XY \)**

\(( \)

**enter Y**

\[]

**enter Z, \( = \)**

This series of steps has entered \( X, \frac{Y + 5.5}{3} \) and \( Z \).

**C. You can also do a quadratic regression, such as**

\[ Z = i + m_1X + m_2X^2 \]

You do it like this:

**enter X, \( XY \)**

\( (X) \)

**enter Z, \( = \)**
Problem -- perform a quadratic regression on these data:

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0</td>
<td>3.1</td>
</tr>
<tr>
<td>3.1</td>
<td>4.9</td>
</tr>
<tr>
<td>3.8</td>
<td>6.2</td>
</tr>
<tr>
<td>3.6</td>
<td>5.6</td>
</tr>
</tbody>
</table>

$$0.000 \quad \text{CL}$$

$$2.000 \quad X$$

$$2.000 \quad ($$

$$2.000 \quad X$$

$$4.000 \quad *)$$

$$4.000 \quad Y$$

$$3.100 \quad Z$$

$$3.100 \quad X$$

$$3.100 \quad ($$

$$3.100 \quad X$$
Therefore, the regression equation for these data is

\[ Z_{est} = 0.762 + 0.909X + 0.131X^2. \]

**Problem 3**

Here we'll be concerned with multiple and partial correlation coefficients for three-variable linear regression. From the three two-variable correlation coefficients that can be calculated between the pairs XY, XZ and YZ, both the multiple and partial correlation coefficients can be calculated. For simplicity, in this example we define X as Group 1, Y as Group 2 and Z as Group 3. Thus, the correlation coefficients calculated by the two-variable linear regression are \( r_{12} \), \( r_{23} \) and \( r_{31} \).

First, let's put in some three-variable data.
<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>1.5</td>
<td>8.6</td>
<td>3.1</td>
</tr>
<tr>
<td>7.5</td>
<td>4.6</td>
<td>1.8</td>
</tr>
</tbody>
</table>

0.000  

- CL  

<table>
<thead>
<tr>
<th>1.000</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.000</td>
<td>Y</td>
</tr>
<tr>
<td>3.000</td>
<td>Z</td>
</tr>
<tr>
<td>7.000</td>
<td>X</td>
</tr>
<tr>
<td>8.000</td>
<td>Y</td>
</tr>
<tr>
<td>9.000</td>
<td>Z</td>
</tr>
<tr>
<td>1.500</td>
<td>X</td>
</tr>
<tr>
<td>8.600</td>
<td>Y</td>
</tr>
<tr>
<td>3.100</td>
<td>Z</td>
</tr>
</tbody>
</table>
A. Calculate the multiple correlation coefficient of X on Y and Z as given by

\[ R_{1.23} = \sqrt{\frac{r_{12}^2 + r_{31}^2 - 2r_{12}r_{23}r_{31}}{1 - r_{23}^2}} \]

Here's one good way to do it:

```
RESET

SET GROUP 1 LIN

X

+)
```

```
0.000  \wedge
0.000  (
0.211  LR  1
0.211  X
0.211  )
0.044  *
0.044  +
```
CHAPTER 5  -183-

SET GROUP 3 LIN REG

X

) 0.044

)

- 0.365 LR 3

( 0.365

) 0.133 *

0.177 )

2.000 X

( 0.211 LR 1

) 0.211 X

SET GROUP 2 LIN REG

X

0.497 LR 2

0.497 X

SET GROUP 3 LIN REG

) 0.365 LR 3

) 0.076 *

0.076 \div
B. Calculate the multiple correlation coefficient of Y on X and Z as given by

\[ R_{2.13} = \sqrt{ \frac{r_{12}^2 + r_{23}^2 - 2r_{12}r_{23}r_{31}}{1 - r_{31}^2} } \]

This expression can be evaluated in this way:

\[ (r_{12}^2) + (r_{23})^2 - (2 \times r_{12} \times r_{23} \times r_{31}) \div (1 - (r_{31}^2)) \]
then \( \sqrt{\phantom{0.000}} \). The result is \( R_{2,13} = 0.498 \). Here's a tape which shows one way of doing it.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>( \wedge )</td>
<td></td>
</tr>
<tr>
<td>0.000</td>
<td>(</td>
<td></td>
</tr>
<tr>
<td>0.211</td>
<td>LR</td>
<td>1</td>
</tr>
<tr>
<td>0.211</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>0.211</td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>0.044</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>0.044</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>0.044</td>
<td>(</td>
<td></td>
</tr>
<tr>
<td>0.497</td>
<td>LR</td>
<td>2</td>
</tr>
<tr>
<td>0.497</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>0.497</td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>0.247</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>0.247</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>0.292</td>
<td>(</td>
<td></td>
</tr>
<tr>
<td>2.000</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>0.211</td>
<td>LR</td>
<td>1</td>
</tr>
<tr>
<td>0.211</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>0.497</td>
<td>LR</td>
<td>2</td>
</tr>
<tr>
<td>0.497</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>0.365</td>
<td>LR</td>
<td>3</td>
</tr>
<tr>
<td>0.365</td>
<td>)</td>
<td></td>
</tr>
</tbody>
</table>
C. Now calculate the multiple correlation coefficient of $Z$ on $X$ and $Y$

as given by

$$R_{3.12} = \sqrt{\frac{r_{31}^2 + r_{23}^2 - 2r_{12}r_{23}r_{31}}{1 - r_{12}^2}}$$

This expression can be evaluated in this way:

$$(r_{31}^2) + (r_{23})^2 - (2 \times r_{12} \times r_{23} \times r_{31}) \div (1 - (r_{12}^2))$$
then \( \sqrt{1} \). The result is \( R_{3.12} = .564 \).

D. Calculate the partial correlation coefficient between \( X \) and \( Y \) as given by

\[
\rho_{12.3} = \frac{\rho_{12} - \rho_{31} \rho_{23}}{\sqrt{(1 - \rho_{31}^2)(1 - \rho_{23}^2)}}
\]

Here's one way of doing it:

```
RESET
(1 -
(1
SET
GROUP 3
LIN REG
X
)
)
X
(
```

```
0.0000 ^
0.0000 ( 
1.0000 -
1.0000 ( 
0.3650 LR 3
0.3650 X
0.3650 )
0.1332 *
0.1332 )
0.8667 *
0.8667 X
0.8667 ( 
```
<table>
<thead>
<tr>
<th>Operation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET GROUP 2</td>
<td>LIN REG</td>
</tr>
<tr>
<td>X</td>
<td></td>
</tr>
<tr>
<td>)</td>
<td></td>
</tr>
<tr>
<td>)</td>
<td></td>
</tr>
<tr>
<td>=</td>
<td></td>
</tr>
<tr>
<td>√</td>
<td></td>
</tr>
<tr>
<td>1/x</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
</tr>
<tr>
<td>(</td>
<td></td>
</tr>
<tr>
<td>SET GROUP 1</td>
<td>LIN REG</td>
</tr>
<tr>
<td>-</td>
<td></td>
</tr>
<tr>
<td>(</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0000</td>
<td>-</td>
</tr>
<tr>
<td>1.0000</td>
<td>(</td>
</tr>
<tr>
<td>0.4978</td>
<td>LR 2</td>
</tr>
<tr>
<td>0.4978</td>
<td>X</td>
</tr>
<tr>
<td>0.4978</td>
<td>)</td>
</tr>
<tr>
<td>0.2478</td>
<td>*</td>
</tr>
<tr>
<td>0.2478</td>
<td>)</td>
</tr>
<tr>
<td>0.7521</td>
<td>*</td>
</tr>
<tr>
<td>0.7521</td>
<td>=</td>
</tr>
<tr>
<td>0.6519</td>
<td>*</td>
</tr>
<tr>
<td>0.6519</td>
<td>√</td>
</tr>
<tr>
<td>0.8074</td>
<td>*</td>
</tr>
<tr>
<td>0.8074</td>
<td>1/x</td>
</tr>
<tr>
<td>1.2385</td>
<td>*</td>
</tr>
<tr>
<td>1.2385</td>
<td>X</td>
</tr>
<tr>
<td>1.2385</td>
<td>(</td>
</tr>
<tr>
<td>0.2113</td>
<td>LR 1</td>
</tr>
<tr>
<td>0.2113</td>
<td>-</td>
</tr>
<tr>
<td>0.2113</td>
<td>(</td>
</tr>
</tbody>
</table>
As you can see on the tape, $r_{12,3} = 0.0366$.

E. Calculate the partial correlation coefficient between $Y$ and $Z$ as given by

$$r_{23,1} = \frac{r_{23} - r_{12}r_{31}}{\sqrt{(1 - r_{12}^2)(1 - r_{31}^2)}}$$

This expression can be evaluated in this way:

$$(1 - (r_{12}^2)) \times (1 - (r_{31}^2))$$

then \[= \sqrt{\frac{1}{x}} \div (r_{23} - (r_{12} \times r_{31})) \] and \[= \].
The result is $r_{23.1} = .4623$. Here's a tape which shows one way to do it:

```
0.0000 ^
0.0000 (  
1.0000 - 
1.0000 (  
0.2113 LR 1
0.2113 X 
0.2113 )  
0.0446 *  
0.0446 )  
0.9553 *  
0.9553 X  
0.9553 (  
1.0000 - 
1.0000 (  
0.3650 LR 3
0.3650 X  
0.3650 )  
0.1332 *  
0.1332 )  
0.8667 *  
0.8667 = 
```
F. Calculate the partial correlation coefficient between Z and X as given by

\[
\begin{align*}
0.8280 & \quad * \\
0.8280 & \quad \sqrt{ } \\
0.9099 & \quad * \\
0.9099 & \quad 1/x \\
1.0989 & \quad * \\
1.0989 & \quad X \\
1.0989 & \quad ( \\
0.4978 & \quad LR \quad 2 \\
0.4978 & \quad - \\
0.4978 & \quad ( \\
0.2113 & \quad LR \quad 1 \\
0.2113 & \quad X \\
0.3650 & \quad LR \quad 3 \\
0.3650 & \quad ) \\
0.0771 & \quad * \\
0.0771 & \quad ) \\
0.4207 & \quad * \\
0.4207 & \quad = \\
0.4623 & \quad * \\
0.4623 & \quad \\
\end{align*}
\]
\[ r_{31.2} = \frac{r_{31} - r_{12} r_{23}}{\sqrt{(1 - r_{12}^2) (1 - r_{23}^2)}} \]

The result is \( r_{31.2} = .3065 \).

Problem 4

Fisher's exact test gives a probability statement for a 2X2 matrix,

\[
\begin{array}{c|c|c}
A & B & n_1 \\
C & D & n_2 \\
\hline
n_a & n_b & N
\end{array}
\]

\[
p = \frac{n_1 ! n_2 ! n_a ! n_b !}{N! A! B! C! D!}
\]

Calculate \( p \) for the following matrix:

\[
\begin{array}{c|c|c}
2 & 5 & 7 \\
7 & 4 & 11 \\
9 & 9 & 18
\end{array}
\]

Here, \( p = \frac{7! 11! 9! 9!}{18! 2! 5! 7! 4!} \).

Here's how you do it on the machine:
Problem 5

Here, we shall take several different kinds of means for the following data:

21.7
28.5
31.2
24.6
27.0
23.2
(7x) 26.1

A. Calculate the Harmonic Mean for the above data.

\[
\text{Harmonic Mean} = \frac{N}{\sum \frac{1}{x}}
\]
CHAPTER 5

\[ \text{SET}\]
\[ \text{GROUP} \quad 2 \quad \frac{1}{x} \quad 0 \]
\[ \sum nxx^2 \]
\[ 2 \quad 1 \quad . \quad 7 \quad \frac{1}{x} \]
\[ \sum nxx^2 \]
\[ 2 \quad 8 \quad . \quad 5 \quad \frac{1}{x} \]
\[ \sum nxx^2 \]
\[ 3 \quad 1 \quad . \quad 2 \quad \frac{1}{x} \]
\[ \sum nxx^2 \]
\[ 2 \quad 4 \quad . \quad 6 \quad \frac{1}{x} \]
\[ \sum nxx^2 \]
\[ 2 \quad 7 \quad \frac{1}{x} \]
\[ \sum nxx^2 \]
\[ 2 \quad 3 \quad . \quad 2 \quad \frac{1}{x} \]
\[ \sum nxx^2 \]
\[ 2 \quad 6 \quad . \quad 1 \quad \frac{1}{x} \]

<table>
<thead>
<tr>
<th>Value</th>
<th>(1/x)</th>
<th>(\sum)</th>
</tr>
</thead>
<tbody>
<tr>
<td>21.700</td>
<td>0.046</td>
<td>*</td>
</tr>
<tr>
<td>0.046</td>
<td>(\Sigma) 2</td>
<td></td>
</tr>
<tr>
<td>28.500</td>
<td>0.035</td>
<td>*</td>
</tr>
<tr>
<td>0.035</td>
<td>(\Sigma) 2</td>
<td></td>
</tr>
<tr>
<td>31.200</td>
<td>0.032</td>
<td>*</td>
</tr>
<tr>
<td>0.032</td>
<td>(\Sigma) 2</td>
<td></td>
</tr>
<tr>
<td>24.600</td>
<td>0.040</td>
<td>*</td>
</tr>
<tr>
<td>0.040</td>
<td>(\Sigma) 2</td>
<td></td>
</tr>
<tr>
<td>27.000</td>
<td>0.037</td>
<td>*</td>
</tr>
<tr>
<td>0.037</td>
<td>(\Sigma) 2</td>
<td></td>
</tr>
<tr>
<td>23.200</td>
<td>0.043</td>
<td>*</td>
</tr>
<tr>
<td>0.043</td>
<td>(\Sigma) 2</td>
<td></td>
</tr>
<tr>
<td>26.100</td>
<td>0.038</td>
<td>*</td>
</tr>
</tbody>
</table>
The Harmonic Mean is 25.885.

B. Take the Root Mean Square for the same data as above.

\[ \text{rms} = \sqrt{\frac{\sum X^2}{N}} \]
The Root Mean Square is 26.161.

C. Now let's take a Geometric Mean for these data.

\[
\text{Geometric Mean} = \sqrt[\sum (x_i)]{\prod x_i}
\]
The Geometric Mean is 25.977.
PROGRAMMING EXAMPLES

Try Stirling's Formula for $\log(n!)$. Here, $n$ is in register 4 and has the value 1728. (Before you start, put 1728 into register 4.)

$$\ln(n!) \approx (n+\frac{1}{2}) \ln(n) - n + \ln\sqrt{2\pi}$$

```
RCLn 4
+
. 5 x
RCLn 4

Ln
LOG

-

RCLn 4
+

(\text{doubles } \pi)

\text{log } \sqrt{2\pi}

\begin{align*}
1,728.000 & \uparrow 4 \\
1,728.000 & + \\
0.500 & x \\
1,728.000 & \uparrow 4 \\
1,728.000 & \lg \\
7.454 & * \\
7.454 & - \\
1,728.000 & \uparrow 4 \\
1,728.000 & + \\
11,157.483 & ( \\
3.141 & + \\
3.141 & = \\
6.283 & *
\end{align*}
```
Let's program Stirling's Formula. We'll put it at location 0030, give
it the symbolic name 'FLAG' and assume n is in register 4.

Do the following:

**RESET** just a precaution - not usually necessary

**BRANCH nn** (or could be **JUMP nn**) **0 3**

put switch on LOAD

Now you're loading a program - the program will be the same sequence
of keys as before, but with a SYMBOL in the front and PRINT and
HALT instructions at the end.
<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Value</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0032</td>
<td>RCL ( n )</td>
<td>111</td>
<td>↑</td>
</tr>
<tr>
<td>0033</td>
<td>4</td>
<td>004</td>
<td>4</td>
</tr>
<tr>
<td>0034</td>
<td>+</td>
<td>021</td>
<td>+</td>
</tr>
<tr>
<td>0035</td>
<td>5</td>
<td>012</td>
<td></td>
</tr>
<tr>
<td>0036</td>
<td>X</td>
<td>005</td>
<td>5</td>
</tr>
<tr>
<td>0037</td>
<td>X</td>
<td>023</td>
<td>X</td>
</tr>
<tr>
<td>0038</td>
<td>RCL ( n )</td>
<td>111</td>
<td>↑</td>
</tr>
<tr>
<td>0039</td>
<td>4</td>
<td>004</td>
<td>4</td>
</tr>
<tr>
<td>0040</td>
<td>Ln</td>
<td>050</td>
<td>lg</td>
</tr>
<tr>
<td>0041</td>
<td>LOG</td>
<td>022</td>
<td>-</td>
</tr>
<tr>
<td>0042</td>
<td>RCL ( n )</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>0043</td>
<td>4</td>
<td>004</td>
<td>4</td>
</tr>
<tr>
<td>0044</td>
<td>+</td>
<td>021</td>
<td>+</td>
</tr>
<tr>
<td>0045</td>
<td>(</td>
<td>026</td>
<td>(</td>
</tr>
<tr>
<td>0046</td>
<td>π</td>
<td>015</td>
<td></td>
</tr>
</tbody>
</table>
Now take the switch off LOAD and put it on RUN, before you forget it's in LOAD and accidentally load in another step.

Put 1728 in register 4 (1728 \( ST_n \ 4 \)) and push \textit{RESUME}.
If you didn't turn off the PRINT switch, all the steps will print out — so turn it off. The answer will print, and the program will stop.

Now you can store another number in register 4, press RESUME and the 445 will do it all over again.

Now try a program with some logic in it. The quadratic is a good one.

\[ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]

where \( ax^2 + bx + c = 0 \)

Now we'll use identifiers and dot lines to indicate what's going on. We want the tape to print identifiers 1, 2 and 3 to indicate \( a, b, \) and \( c \); and identifiers 4 and 5 to indicate whether the roots are real or imaginary.

So a sample tape might look like this for \( a = 17, b = 18, c = 19 \):

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>17.00</td>
</tr>
<tr>
<td>2.</td>
<td>18.00</td>
</tr>
<tr>
<td>3.</td>
<td>19.00</td>
</tr>
<tr>
<td></td>
<td>.................</td>
</tr>
<tr>
<td>5.</td>
<td>0.91</td>
</tr>
<tr>
<td></td>
<td>-0.52</td>
</tr>
</tbody>
</table>

Here's a flow chart for the program:
Here's the program. Since it's written with symbolic addressing, it can go anywhere in memory. It happens to start at step 0140.
<table>
<thead>
<tr>
<th>IDENT</th>
<th>identifier 2</th>
<th>0149</th>
<th>177</th>
</tr>
</thead>
<tbody>
<tr>
<td>HALT</td>
<td></td>
<td>0150</td>
<td>056</td>
</tr>
<tr>
<td>PRINT ENTRY</td>
<td>enter, print &amp; store b</td>
<td>0151</td>
<td>060</td>
</tr>
<tr>
<td>ST_n</td>
<td></td>
<td>0152</td>
<td>110 →</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0153</td>
<td>001</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>0154</td>
<td>003</td>
</tr>
<tr>
<td>IDENT</td>
<td>identifier 3</td>
<td>0155</td>
<td>177</td>
</tr>
<tr>
<td>HALT</td>
<td></td>
<td>0156</td>
<td>056</td>
</tr>
<tr>
<td>PRINT ENTRY</td>
<td>enter, print &amp; store c</td>
<td>0157</td>
<td>060</td>
</tr>
<tr>
<td>ST_n</td>
<td></td>
<td>0158</td>
<td>110 →</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0159</td>
<td>002</td>
</tr>
<tr>
<td>DOT PRINT</td>
<td>dot line</td>
<td>0160</td>
<td>176</td>
</tr>
<tr>
<td>RCL_n</td>
<td></td>
<td>0161</td>
<td>111 ↑</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0162</td>
<td>001</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>0163</td>
<td>023</td>
</tr>
<tr>
<td>-</td>
<td></td>
<td>0164</td>
<td>022</td>
</tr>
<tr>
<td>(</td>
<td></td>
<td>0165</td>
<td>026</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>0166</td>
<td>004</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>0167</td>
<td>023</td>
</tr>
<tr>
<td>RCL_n</td>
<td>b²-4ac</td>
<td>0168</td>
<td>111 ↑</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0169</td>
<td>000</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>0170</td>
<td>023</td>
</tr>
<tr>
<td>RCL_n</td>
<td></td>
<td>0171</td>
<td>111</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0172</td>
<td>002</td>
</tr>
<tr>
<td>)</td>
<td></td>
<td>0173</td>
<td>027</td>
</tr>
<tr>
<td>=</td>
<td>Jump if positive</td>
<td>0174</td>
<td>020</td>
</tr>
<tr>
<td>+</td>
<td>to symbolic address &quot;11&quot;</td>
<td>0175</td>
<td>126</td>
</tr>
<tr>
<td>SYMB</td>
<td>otherwise, ident 5, - complex roots</td>
<td>0176</td>
<td>021</td>
</tr>
<tr>
<td>1</td>
<td>4ac-b² in entry register</td>
<td>0177</td>
<td>067</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>0178</td>
<td>001</td>
</tr>
<tr>
<td>IDENT</td>
<td></td>
<td>0179</td>
<td>005</td>
</tr>
<tr>
<td>CHG SIGN</td>
<td></td>
<td>0180</td>
<td>177</td>
</tr>
<tr>
<td>√</td>
<td></td>
<td>0181</td>
<td>013</td>
</tr>
<tr>
<td>÷</td>
<td></td>
<td>0182</td>
<td>055</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0183</td>
<td>024</td>
</tr>
<tr>
<td>÷</td>
<td></td>
<td>0184</td>
<td>002</td>
</tr>
<tr>
<td>RCL_n</td>
<td>√4ac-b²/2a</td>
<td>0185</td>
<td>024</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0186</td>
<td>111</td>
</tr>
<tr>
<td>=</td>
<td>Print imag. term</td>
<td>0187</td>
<td>000</td>
</tr>
<tr>
<td>PRINT ENTRY</td>
<td></td>
<td>0188</td>
<td>020</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0189</td>
<td>060</td>
</tr>
</tbody>
</table>
RCLn
1
CHG SIGN
⁄
2
⁄
RCLn
0
=
PRINT ENTRY

Print real term
ADV
JUnn
SYMB
0
Return to start
SYMB
1
Begin real root routine
4
IDENT
Ident 4 - real roots
√
STn
Store \( \sqrt{b^2 - 4ac} \) for later use
3
-
RCLn
1
⁄
-b + \( \sqrt{b^2 - 4ac} \)
2a
RCLn
0
=
PRINT ENTRY
1st real root
RCLn
3
CHG SIGN
-
RCLn
1
⁄
-b - \( \sqrt{b^2 - 4ac} \)
2a
RCLn
0
=
PRINT ENTRY
2nd real root
ADV
JUnn
SYMB
0
Back to START

0190  111  ↑
0191  001  1
0192  013  -
0193  024  ÷
0194  002  2
0195  024  ÷
0196  111  ↑
0197  000  0
0198  020  =
0199  060
0200  065
0201  126  Ju
0202  067
0203  000  0
0204  066
0205  001  1
0206  004  4
0207  177
0208  055  \( \sqrt{ \) 
0209  110  ↓
0210  003  3
0211  022  -
0212  111  ↑
0213  001  1
0214  024  ÷
0215  002  2
0216  024  ÷
0217  111  ↑
0218  000  0
0219  020  =
0220  060
0221  111  ↑
0222  003  3
0223  013  -
0224  022  -
0225  111  ↑
0226  001  1
0227  024  ÷
0228  002  2
0229  024  ÷
0230  111  ↑
0231  000  0
0232  020  =
0233  060
0234  065
0235  126  Ju
0236  067
0237  000  0
After you've entered the program, put the switch back to RUN and push RESUME to get the first identifier printed out. Then enter the sample problem (a=17, b=18, c=19). Your tape should look like the sample above.
CHAPTER SIX

MAGNETIC CARDS AND OTHER IMPORTANT THINGS
MAGNETIC CARDS

That little slot above the keyboard is for mag (magnetic) cards. You can record either a program or data on these cards and store them for ever and ever. Then, whenever you need that program or data again, just stick in the card and it's in the machine. Automatically. Each card has two edges, and each edge can hold up to 256 program steps or 32 data registers. So each card can hold up to 512 program steps or 64 data registers. The ends (with the oblong holes) are the ones you stick into the machine, and you can write with a felt-tip pen on a half-inch strip lengthwise down the middle of the light side of the card to identify what's on each edge and which end to stick in first. You can protect the contents of a card permanently by cutting off the thin strip of card material on the outer edge alongside the oblong hole. Then the oblong hole becomes somewhat of a rectangle along the leading edge of the card. Once you've done that, you can never record anything else on that card again—not even by mistake. But the information on the card can be read just the same.

Recording A Program

Access the first instruction to be written with JUMP, BRANCH or symbolic addressing, put the mag card switch on RECORD and stick the mag card into the slot—firmly, but gently—with the dark side
down. The card reader/writer unit will grab the card, pull it in, fill it with 256 program steps and then push it out again. If you want to record more or less than 256 steps, use the keyboard numeral keys to enter the total number of steps you're going to record just before you start sticking in cards (and after you've branched, jumped or symbolically addressed to the first location.) Then stick in a card and wait for it to come out. If you've asked for less than 256 steps, that's it. If you've asked for more, turn the card around, stick it in again, stick in a second card, etc. When you stick in the first card, the IDLE light will go off. And it will stay off until you've stuck in enough cards to record the number of steps you've asked for.

If the program instructions you want to record don't start at a branch point or symbolically addressed location, you have to use a very specific series of steps to record them on mag cards. And you have to do it in this order:

1) Put the mag card switch on RECORD.
2) JUMP or BRANCH to a nearby branch point.
3) If you will be recording more or less than 256 steps, enter the number of steps now.
4) Put the switch on LOAD.
5) LIST or BACK SPACE to the instruction before the first step you want recorded on the card.
6) Stick in the required number of cards.

7) Don't forget to take the machine out of LOAD.

**Recording Data**

The procedure for recording data (registers) on mag cards is essentially the same as recording a program, with the following exceptions:

1) Access the first register (the beginning of your data series) with \texttt{ST}_{nn} or \texttt{RCL}_{nn} instead of JUMP, BRANCH, etc.

2) The number of information units that can be recorded on each edge is 32 registers (instead of 256 steps). If you're going to record more or less than 32 registers, enter the number of registers you need just before sticking in the first card.

**Entering A Program**

Access the address in program memory where you wish the program to begin. Then put the mag card switch on ENTER. Now just start sticking in cards (in their proper order, of course). The machine will load all program steps, up to the capacity of program memory, in sequential order as they are read, no matter how many cards have to be fed in. If your card only contains 25 steps, however, the machine will enter only the 25 steps and then spit the card out.

**Entering Data**

Access the register where you want the data to begin. Put the mag
switch on ENTER. Then start sticking in cards until all the data are entered. The machine will load data sequentially into consecutive registers until there is no more information being fed in. If there are less than 32 registers on your card, however, the machine will enter what's there and then spit the card out.

**Verification Of Card Information**

Each time information is recorded on a mag card, a special number — consisting of the sum of all the numbers contained on the card— is also recorded. Then, whenever the card is entered, the machine adds up this sum again and compares it with the number recorded on the card. If they're not the same the machine goes into ERROR mode.

**Care Of Magnetic Cards**

Mag cards are pretty durable. But they still require proper care. Write on them only with a felt-tip pen. And write in the half-inch strip running lengthwise down the middle of the light side. The dark side is the magnetic side, and should be kept from contamination (coffee, glue, tears, etc.). Don't store mag cards with their magnetic surfaces against each other. Use the envelopes provided. And try not to bend them too much. If you take care of them, these cards can last virtually forever.
CHAPTER 6  -213-

ACCURACY OF THE 445

As we've mentioned, the 445 holds a 13-digit signed mantissa with a 2-digit signed exponent for all numbers. This is true of the data registers as well as the entry register. In addition there are some specific levels of accuracy to consider. And these are them:

<table>
<thead>
<tr>
<th>Function</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>13 digits</td>
</tr>
<tr>
<td>Subtract</td>
<td>13 digits</td>
</tr>
<tr>
<td>Multiply</td>
<td>13 digits, rounded to the 13th digit</td>
</tr>
<tr>
<td>Divide</td>
<td>13 digits, rounded to the 13th digit (i.e., 2 ÷ 3 = 0.6666666666667)</td>
</tr>
<tr>
<td>a^x</td>
<td>±1 in the 11th digit</td>
</tr>
<tr>
<td>Invert (1/x)</td>
<td>Same as divide</td>
</tr>
<tr>
<td>Square root</td>
<td>±3 in the 13th digit</td>
</tr>
<tr>
<td>Logarithm</td>
<td>±1 in the 11th digit</td>
</tr>
<tr>
<td>Antilogarithm</td>
<td>±1 in the 11th digit</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>±5 in the 13th digit</td>
</tr>
<tr>
<td>Normal probability</td>
<td>to 6 fractional digits</td>
</tr>
<tr>
<td>z-statistic</td>
<td>±5 in the 13th digit</td>
</tr>
<tr>
<td>X^2 statistic</td>
<td>±5 in the 13th digit</td>
</tr>
<tr>
<td>Linear regression</td>
<td>±5 in the 13th digit</td>
</tr>
</tbody>
</table>

PERIPHERAL EQUIPMENT

In addition to all the things the 445 will do by itself, there are some specialized things it can do with the addition of peripheral equipment. Listed below are the peripheral devices designed for use with the 445 Statistician.

Model 490 - Mark Sense/Punch Card Reader

This device will read program steps and data from punched paper cards or mark sense cards and store the information directly into the 445 memory.
CHAPTER 6 -214-

Model 492 - Magnetic Tape Cassette Unit

Reads and writes data or program steps (or a combination of both) on standard digital cassettes. You can store up to 100,000 program steps or 12,500 data registers on a single cassette.

Model 493 - X Y Plotter

This unit transforms computational results into graphic form. It plots functions, makes graphs, draws axes and does alphanumeric labelling.

Model 494 - Typewriter

There are two versions of the typewriter:

1) Output Only Typewriter - Does automatic form preparation. It can, for instance, prepare formatted, tabular documents with alphanumeric headings and complete tab facilities.

2) Input/Output Typewriter - Has the same output capability as above, with the additional capacity to accept alphanumeric input from the typewriter keyboard.
1. ADVANCE advances the tape. If you press it and release it, the tape will stop and return to the point where it was last stopped. If you hold it down, the tape will continue advancing until it reaches the end of the roll.

2. The ENTER/RECORD switch controls the start and stop of the machine. When you press it and release it, the machine will begin to operate. If you hold it down, the machine will operate continuously until you press it and release it to stop.

3. The ON/STAND BY/OFF switch controls the power supply to the machine. When the switch is in the ON position, the machine will operate. When the switch is in the STAND BY position, the machine will be in a standby mode, ready to start when needed. When the switch is in the OFF position, the machine will be turned off, and power will be shut off.

4. The NEXT/LOAD switch advances the machine to the next operation or loads the machine with a new program. When the switch is in the NEXT position, the machine will advance to the next operation in the program. When the switch is in the LOAD position, the machine will be loaded with a new program.

5. The ENTRY switch allows you to enter data into the machine. When the switch is pressed, the machine will enter the data into its memory. When the switch is released, the data will be stored.

6. The SAVE switch allows you to save a program or data set. When the switch is pressed, the machine will save the current program or data set. When the switch is released, the program or data set will be saved.

7. The PRINT switch allows you to print out the contents of the machine's memory. When the switch is pressed, the machine will print out the contents of its memory. When the switch is released, the printing will stop.

8. The CLEAR switch allows you to clear the contents of the machine's memory. When the switch is pressed, the machine will clear the contents of its memory. When the switch is released, the clearing will stop.

9. The EXIT switch allows you to exit the machine's program. When the switch is pressed, the machine will exit its current program and return to the main menu.

10. The DISCARD SYMBOL/SIZE performs two functions. It can be used in conjunction with the arithmetic keys to discard symbols and perform calculations. It can also be used with the main data STORE, to store data from the machine to the tape. The function is selected by pressing the DISCARD key and then selecting the desired function using the arithmetic keys.

11. The STORE switch allows you to store data from the machine to the tape. When the switch is pressed and released, the machine will store the data on the tape. When the switch is held down, the machine will continue to store data until the switch is released.

12. The PROGRAM switch allows you to control the machine's program. When the switch is pressed, the machine will begin to operate according to the program. When the switch is released, the machine will stop and return to the program menu.

13. The CLEAR PRINT prevents the data from being printed. When the switch is pressed, the data will not be printed. When the switch is released, the data will be printed.

14. The RECALL switch allows you to recall data from the machine's memory. When the switch is pressed, the machine will recall the data from its memory. When the switch is released, the data will be cleared.

15. The JUMP switch allows you to jump to a specific location in the program. When the switch is pressed, the machine will jump to the specified location. When the switch is released, the jump will be canceled.

16. The EXIT switch allows you to exit the program. When the switch is pressed, the program will exit. When the switch is released, the program will continue.

17. The EXIT switch sums the values for the first and second function registers. When the switch is pressed, the values will be summed. When the switch is released, the sum will be displayed.

18. The EXIT switch computes the standard deviation, mean, and standard error of the data set. When the switch is pressed, the standard deviation, mean, and standard error will be calculated. When the switch is released, the calculation will be cancelled.

19. The EXIT switch computes the correlation coefficient. When the switch is pressed, the correlation coefficient will be calculated. When the switch is released, the calculation will be cancelled.

20. The EXIT switch computes the regression for two-and three-variable dependent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

21. The EXIT switch computes the linear regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

22. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

23. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

24. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

25. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

26. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

27. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

28. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

29. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

30. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

31. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

32. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

33. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

34. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

35. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

36. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

37. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

38. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

39. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

40. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

41. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

42. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

43. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

44. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

45. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

46. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

47. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

48. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

49. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

50. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

51. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

52. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

53. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

54. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

55. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

56. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

57. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

58. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.

59. The EXIT switch computes the regression for two- and three-variable independent data. When the switch is pressed, the regression will be calculated. When the switch is released, the calculation will be cancelled.
APPENDIX

Functions and codes
# SUMMARY OF FUNCTIONS

Following is a summary of the functions available, and their numeric codes.

<table>
<thead>
<tr>
<th>Function</th>
<th>Page in text</th>
<th>On Keyboard</th>
<th>Numeric Code</th>
<th>Print Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set D. P.</td>
<td>17-18</td>
<td>Yes</td>
<td>117</td>
<td>None</td>
</tr>
<tr>
<td>Reset</td>
<td>17</td>
<td>Yes</td>
<td>062</td>
<td>(\wedge)</td>
</tr>
<tr>
<td>Clear Entry</td>
<td>20</td>
<td>Yes</td>
<td>063</td>
<td>None</td>
</tr>
<tr>
<td>Print Entry</td>
<td>20</td>
<td>Yes</td>
<td>060</td>
<td>None</td>
</tr>
<tr>
<td>Print Answer</td>
<td>20</td>
<td>Yes</td>
<td>061</td>
<td>A</td>
</tr>
<tr>
<td>Paper Advance</td>
<td>22</td>
<td>Yes</td>
<td>065</td>
<td>None</td>
</tr>
<tr>
<td>Numerals 0-9</td>
<td>22</td>
<td>Yes</td>
<td>000-011</td>
<td>None</td>
</tr>
<tr>
<td>Decimal Point</td>
<td>23</td>
<td>Yes</td>
<td>012</td>
<td>None</td>
</tr>
<tr>
<td>Exponent</td>
<td>23</td>
<td>Yes</td>
<td>014</td>
<td>None</td>
</tr>
<tr>
<td>Change Sign</td>
<td>23</td>
<td>Yes</td>
<td>013</td>
<td>None</td>
</tr>
<tr>
<td>Σ &amp; e</td>
<td>41</td>
<td>Yes</td>
<td>015</td>
<td>None</td>
</tr>
<tr>
<td>Second Function</td>
<td>41</td>
<td>Yes</td>
<td>052</td>
<td>F2</td>
</tr>
<tr>
<td>Third Function</td>
<td>42</td>
<td>Yes</td>
<td>116,003</td>
<td>F3</td>
</tr>
<tr>
<td>Plus</td>
<td>24</td>
<td>Yes</td>
<td>021</td>
<td>+</td>
</tr>
<tr>
<td>Minus</td>
<td>24</td>
<td>Yes</td>
<td>022</td>
<td>-</td>
</tr>
<tr>
<td>Multiply</td>
<td>24</td>
<td>Yes</td>
<td>023</td>
<td>X</td>
</tr>
<tr>
<td>Divide</td>
<td>24</td>
<td>Yes</td>
<td>024</td>
<td>÷</td>
</tr>
<tr>
<td>Equals</td>
<td>24, 38</td>
<td>Yes</td>
<td>020</td>
<td>=</td>
</tr>
<tr>
<td>List-Mode Add</td>
<td>153</td>
<td>No</td>
<td>041</td>
<td>+</td>
</tr>
<tr>
<td>List-Mode Subtract</td>
<td>153</td>
<td>No</td>
<td>042</td>
<td>-</td>
</tr>
<tr>
<td>List-Mode Subtotal</td>
<td>153</td>
<td>No</td>
<td>043</td>
<td>(\odot)</td>
</tr>
<tr>
<td>List-Mode Total</td>
<td>153</td>
<td>No</td>
<td>040</td>
<td>(*)</td>
</tr>
<tr>
<td>a^x</td>
<td>25</td>
<td>Yes</td>
<td>025</td>
<td>a^x</td>
</tr>
<tr>
<td>Open Parenthesis</td>
<td>35-36</td>
<td>Yes</td>
<td>026</td>
<td>(</td>
</tr>
<tr>
<td>Close Parenthesis</td>
<td>35-36</td>
<td>Yes</td>
<td>027</td>
<td>)</td>
</tr>
<tr>
<td>Invert</td>
<td>25</td>
<td>Yes</td>
<td>054</td>
<td>1/x</td>
</tr>
<tr>
<td>Square Root</td>
<td>26</td>
<td>Yes</td>
<td>055</td>
<td>(\sqrt{_})</td>
</tr>
<tr>
<td>Logarithm</td>
<td>41</td>
<td>Yes</td>
<td>050</td>
<td>lg</td>
</tr>
<tr>
<td>Antilogarithm</td>
<td>41</td>
<td>Yes</td>
<td>051</td>
<td>lg^{-1}</td>
</tr>
<tr>
<td>Integer Fraction</td>
<td>43</td>
<td>Yes</td>
<td>116,005</td>
<td>I</td>
</tr>
<tr>
<td>Absolute Value</td>
<td>154</td>
<td>No</td>
<td>045</td>
<td>X</td>
</tr>
<tr>
<td>Equals-Sum-Zero</td>
<td>155</td>
<td>No</td>
<td>037</td>
<td>(\Sigma_g)</td>
</tr>
<tr>
<td>Square</td>
<td>154</td>
<td>No</td>
<td>053</td>
<td>(\Sigma_n)</td>
</tr>
<tr>
<td>Factorial</td>
<td>90</td>
<td>Yes</td>
<td>116,006</td>
<td>!</td>
</tr>
<tr>
<td>Increment Entry</td>
<td>154</td>
<td>No</td>
<td>151</td>
<td>None</td>
</tr>
<tr>
<td>Decrement Entry</td>
<td>154</td>
<td>No</td>
<td>152</td>
<td>None</td>
</tr>
<tr>
<td>Identifier</td>
<td>157</td>
<td>No</td>
<td>177</td>
<td>None</td>
</tr>
<tr>
<td>Dot Print</td>
<td>155</td>
<td>No</td>
<td>176</td>
<td>None</td>
</tr>
<tr>
<td>Clear Scratch Pad Registers</td>
<td>46</td>
<td>Yes</td>
<td>116,000</td>
<td>None</td>
</tr>
<tr>
<td>Set Group</td>
<td>44</td>
<td>Yes</td>
<td>115</td>
<td>None</td>
</tr>
<tr>
<td>Sum-Square</td>
<td>46</td>
<td>Yes</td>
<td>047</td>
<td>(\Sigma_n)</td>
</tr>
<tr>
<td>Function</td>
<td>Page in text</td>
<td>On Keyboard</td>
<td>Numeric Code</td>
<td>Print Symbol</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------</td>
<td>-------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Two Variable Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accumulation</td>
<td>58</td>
<td>Yes</td>
<td>036</td>
<td>X, Y</td>
</tr>
<tr>
<td>Three Variable Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accumulation</td>
<td>61</td>
<td>Yes</td>
<td>036</td>
<td>X, Y, Z</td>
</tr>
<tr>
<td>Standard Deviation (n-1)</td>
<td>53</td>
<td>Yes</td>
<td>077</td>
<td>SDn-1</td>
</tr>
<tr>
<td>Standard Deviation (n)</td>
<td>54</td>
<td>Yes</td>
<td>116,004</td>
<td>SDn</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>65</td>
<td>Yes</td>
<td>076</td>
<td>LR, n</td>
</tr>
<tr>
<td>Two Variable Linear Regression</td>
<td>66</td>
<td>Yes</td>
<td>076</td>
<td>LR, n</td>
</tr>
<tr>
<td>Three Variable Linear Regression</td>
<td>69</td>
<td>Yes</td>
<td>076</td>
<td>LR, 4</td>
</tr>
<tr>
<td>Line</td>
<td>72</td>
<td>Yes</td>
<td>116,011</td>
<td>X, Y</td>
</tr>
<tr>
<td>Line-Two Variable</td>
<td>72</td>
<td>Yes</td>
<td>116,011</td>
<td>X, Y</td>
</tr>
<tr>
<td>Line-Three Variable</td>
<td>74</td>
<td>Yes</td>
<td>116,011</td>
<td>X, Y, Z</td>
</tr>
<tr>
<td>t-statistic</td>
<td>76</td>
<td>Yes</td>
<td>073</td>
<td>t</td>
</tr>
<tr>
<td>Chi-Square</td>
<td>85</td>
<td>Yes</td>
<td>116,010</td>
<td>X²</td>
</tr>
<tr>
<td>Delete</td>
<td>50</td>
<td>Yes</td>
<td>046</td>
<td></td>
</tr>
<tr>
<td>z-statistic</td>
<td>55</td>
<td>Yes</td>
<td>116,001</td>
<td>Z</td>
</tr>
<tr>
<td>Permutation/Combination</td>
<td>88</td>
<td>Yes</td>
<td>116,007</td>
<td>n, r, p</td>
</tr>
<tr>
<td>Normal Distribution</td>
<td>56</td>
<td>Yes</td>
<td>116,002</td>
<td>P, z</td>
</tr>
<tr>
<td>Store Scratch Pad</td>
<td>26</td>
<td>Yes</td>
<td>110</td>
<td>↓ n</td>
</tr>
<tr>
<td>Recall Scratch Pad</td>
<td>26</td>
<td>Yes</td>
<td>111</td>
<td>↑ n</td>
</tr>
<tr>
<td>Exchange Scratch Pad</td>
<td>-</td>
<td>No</td>
<td>112</td>
<td>$ n</td>
</tr>
<tr>
<td>Add-to-Register (Scratch Pad)</td>
<td>160</td>
<td>No</td>
<td>113</td>
<td>+n n</td>
</tr>
<tr>
<td>Total Scratch Pad</td>
<td>-</td>
<td>No</td>
<td>114</td>
<td>* n</td>
</tr>
<tr>
<td>Store Main Data</td>
<td>31</td>
<td>Yes</td>
<td>120</td>
<td>↓ n n</td>
</tr>
<tr>
<td>Recall Main Data</td>
<td>31</td>
<td>Yes</td>
<td>121</td>
<td>↑ n n</td>
</tr>
<tr>
<td>Exchange Main Data</td>
<td>155</td>
<td>No</td>
<td>122</td>
<td>$ n n</td>
</tr>
<tr>
<td>Add-to-Register (Main Data)</td>
<td>160</td>
<td>No</td>
<td>123</td>
<td>+n n n</td>
</tr>
<tr>
<td>Branch</td>
<td>128-129</td>
<td>Yes</td>
<td>127</td>
<td>None</td>
</tr>
<tr>
<td>Jump</td>
<td>128-129</td>
<td>Yes</td>
<td>126</td>
<td>None</td>
</tr>
<tr>
<td>Halt</td>
<td>99</td>
<td>Yes</td>
<td>056</td>
<td>None</td>
</tr>
<tr>
<td>Flag 1</td>
<td>152</td>
<td>Yes</td>
<td>016</td>
<td>None</td>
</tr>
<tr>
<td>Flag 2</td>
<td>154</td>
<td>No</td>
<td>017</td>
<td>None</td>
</tr>
<tr>
<td>Reset Flag 1</td>
<td>152</td>
<td>No</td>
<td>166</td>
<td>None</td>
</tr>
<tr>
<td>Reset Flag 2</td>
<td>-</td>
<td>No</td>
<td>167</td>
<td>None</td>
</tr>
<tr>
<td>Resume</td>
<td>130</td>
<td>Yes</td>
<td>057</td>
<td>None</td>
</tr>
<tr>
<td>Symbol/Indirect</td>
<td>133,147</td>
<td>Yes</td>
<td>067</td>
<td>None</td>
</tr>
<tr>
<td>Print Disable</td>
<td>158</td>
<td>No</td>
<td>154</td>
<td>None</td>
</tr>
<tr>
<td>Print Enable</td>
<td>158</td>
<td>No</td>
<td>155</td>
<td>None</td>
</tr>
<tr>
<td>Program Address Keys</td>
<td>90,160</td>
<td>Optional</td>
<td>105-107</td>
<td>None</td>
</tr>
<tr>
<td>Code</td>
<td>Function</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>002</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>003</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>004</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>005</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>006</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>007</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>012</td>
<td>DECIMAL POINT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>013</td>
<td>CHANGE SIGN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>014</td>
<td>EXPONENT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>015</td>
<td>( \pi / e )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>016</td>
<td>Flag 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>017</td>
<td>Flag 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>020</td>
<td>= (equals)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>021</td>
<td>+ (plus)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>022</td>
<td>- (minus)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>023</td>
<td>X (multiply)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>024</td>
<td>( \div ) (divide)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>025</td>
<td>( a^x )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>026</td>
<td>( (open parenthesis)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>027</td>
<td>) (close parenthesis)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>034</td>
<td>Permutations/combinations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>035</td>
<td>Chi-square</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>036</td>
<td>XY data input</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>037</td>
<td>Equals-Sum-Zero</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>040</td>
<td>List-Mode Total</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>041</td>
<td>List-Mode Add</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>042</td>
<td>List-Mode Subtract</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>043</td>
<td>List-Mode Sub-total</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>044</td>
<td>Integer/Fraction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>045</td>
<td>Absolute Value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>046</td>
<td>DELETE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>047</td>
<td>SUM-SQUARE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>050</td>
<td>Ln/LOG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>051</td>
<td>( e^x / 10^x )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>052</td>
<td>SECOND FUNCTION</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>053</td>
<td>Square</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>054</td>
<td>Invert ( \frac{1}{x} )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>055</td>
<td>Square Root</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>056</td>
<td>HALT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>057</td>
<td>RESUME</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
445 FUNCTION CODES IN NUMERICAL ORDER

060  PRINT ENTRY
061  PRINT ANSWER
062  RESET
063  CLEAR ENTRY REGISTER
065  PAPER ADVANCE
066  DEFINE SYMBOL
067  INDIRECT/SYMBOL
073  t-dependent/t-independent
075  z-statistic
076  Linear Regression
077  Standard Deviation
105  (optional)
106  (optional)
107  (optional)
110  STOREn
111  RECALLn
112  EXCHANGEn
113  Add to scratch pad register n
114  Total scratch pad register n
115  SET GROUP
116  \( \varepsilon \)n
117  SET DECIMAL POINT
120  STOREnn
121  RECALLnn
122  EXCHANGE_{nn}
123  Add to register nn
126  JUMP nn
127  BRANCH nn
151  Increment entry register (+1)
152  Decrement entry register (-1)
154  PRINT DISABLE
155  PRINT ENABLE
157  RECALL DECIMAL POINT
166  Reset Flag 1
167  Reset Flag 2
176  PRINT DOT LINE
177  IDENTIFIER
INDEX

Absolute value, 120, 154

Absolute (direct) addressing, 128-129

Accuracy of the machine, 213

Add

keyboard addition, 24

list-mode, 153

to register, 160

Additional functions, description, 43

Addressing

direct (absolute), 128-129

indirect, 147-150

machine level (open channel), 156-157

symbolic, 133-136

Advance key, 22

Algebra

algebraic chaining, 36-38

algebraic entry on keyboard, 24-26
INDEX -2-

Alphanumeric labelling, XY plotter, 214

Antilogarithm, 41-42

Arithmetic
  basic keyboard, 24-25
  in and out of main data registers, 33
  in and out of scratch pad registers, 28-31
  list-mode, 153
  repeated, 40

Back space, 139-140

Branch
  branching and jumping, 129-130
  branch points, 128-129
  conditional branching, 146-147
  diagram, 130

Buffering of keyboard, 23-24

Chaining, algebraic, 36-38

Change Sign key, 23
Changing the sign of numbers, 22

Chi-square, 85-88

Combinations, and permutations, 88-90

Conditionals
  description, 146-147

Constant dividend, 33-35

Constant divisor, 35

Constant multiplier, 33-35

Contents, table of, i

Correlation
  examples of, 6-8, 9-12

Data summation
  for sum-square, 46-52
  register usage charts, 46, 58, 62, 66
  three-variable, 61-65
  two-variable, 58-61
INDEX -4-

Decimal point
- automatic setting with On, 16
- automatic shifting, 18
- entered with numbers, 22-23
- recall decimal point, 159
- Set Decimal Point key, 18

Decrementing
- entry register, 154
- machine level (open channel) register, 157
- program counter, 113

Degree/Grad switch, 17

Deleting data from summation, 50-52

Dependent t-statistic, 76-85

Direct (absolute) addressing, 128-129

Distance, related to earth's curvature, 92-93
Divide

constant dividend, 33-35
constant divisor, 35
keyboard division, 24-25

Dot line, 155

Dummies, 8

Enter code

description, 136-139
examples, 120, 137-138
functions, 152-160

Entering "grouped" data into summation, 49-50

Entering numbers, 22

Entering summed data, 45, 72

Equals

equals-sum-zero, 155
repeated, 38-40
with keyboard arithmetic, 24-26
Error mode

- description, 20-21
- with Clear Entry, 20
- with Reset, 17

Examples and problems

- general examples, 163-167
- problems (see separate listing)
- programming examples, 199-207

Exchange registers

- indirectly, 147-150
- machine level (open channel), 157

Exponent

- entering numbers exponentially, 23
- Exponent key, 23
- exponential mode, automatic, 18
- exponential mode, printing format, 18
- exponential mode, with Set D.P. key, 18
- raising a number to a power with a^x key, 25

Factorial, 90
Flags

flag 1, 152
flag 2, 154
with conditions, 147

Flow chart

description, 108-109
examples, 109, 116, 204

Fractional numbers

printing format, 18

Fraction/integer separation, 43-44

Grads, 17

Degree/Grad switch, 17

Granaries of Isis, The, 1

Greek Ships and Other Phenomena, 91A

"Grouped" data

deleting, 51-52
entering into summation, 49-50
Groups, 44-46
  data storage chart, 46
  description, 44-46
  set automatically with On, 16

Halt, 99, 151

Identifier, 157-158

Idle light
  description, 22

Incrementing
  entry register, 154
  machine level (open channel) register, 157
  program counter, 112

Independent t-statistic, 76-85

Indirect addressing, 147-150
  machine level (open channel) 156-157
INDEX

Input

keyboard, 22
magnetic cards, 209, 211-212
magnetic tape cassette, 214
mark sense/punch card reader, 213
typewriter, 214

Insert, 140-142

Integer/fraction separation, 43-44

Introduction, ii

Invert (reciprocal)

as multiple root, 198
key $\frac{1}{\sqrt{x}}$, 25-26
used for constant divisor, 35

Isis, 8, 9, 12

Jump

branching and jumping, 129-130
branch points, 128-129
conditional jumping, 146-147
diagram, 130
Keyboard

- arithmetic, 24-25
- buffering, 23-24
- program access, 160-161

Line, 72-76

Linear regression, 65-72

- three-variable, 69-72
- two-variable, 66-68

Listing program, 151

List-mode arithmetic, 153

Load

- loading a program, 131-133

Location, current, in program memory, 151-152

Logarithm, 41-42

Machine level (open channel) addressing, 156-157
Magnetic (mag) cards

capacity, 209

care of mag cards, 212

entering a program, 211

entering data, 211-212

read/write unit, 209

recording a program, 209-211

recording data, 211

verification of card information, 212

Magnetic tape cassette unit, 214

Main data registers

addressing, 32-33

add to registers, 160

arithmetic in and out of, 33

description, 31-33

example of use, 33

Mark sense/punch card reader, 213

Mean

example of mathematical average, 4

n version, 54-55

n-1 version, 53-54
Memory

cleared with On, 16
lost in Off mode, 16
map, 127
overview, 126-128
program memory filled with NOOP codes, 16
retained with Standby, 16

Multiple functions, 41-43

Multiple variable data summation, 58-65
  three-variable, 61-65
two-variable, 58-61

Multiply
  constant multiplier, 33-35
  keyboard multiplication, 24-25

Negative numbers
  position of minus sign, 18

NOOP code
  automatic with On, 16
  with insert, 141-142
Normal distribution, 56-57

Numbers
  changing the sign of, 22
  entering on keyboard, 22

Off mode, 16

Omens
  agricultural, 9
  of Isis, 8, 9, 12

On-Standby-Off switch, 16

Open channel (machine level) addressing, 156-157

Optional keys, 90-91, 160-161

Output
  magnetic cards, 209-211
  magnetic tape cassette, 214
  tape, 17-18
  typewriter, 214
  XY plotter, 214
Overflow mode

- description, 22
- with Clear Entry, 20
- with Reset, 17

Paper advance, 22

Parentheses, 35-36

Peripheral equipment, 213-214

Permutations, and combinations, 88-90

Pi and e key, 41

Plotter, XY, 214

Print Answer

- in a program, 131
- key, 20

Print enable/disable, 158-159
INDEX -15-

Print Entry

in a program, 131
key, 20

Printing

control, 18-19
exponential mode, 18
format, 17
fractional numbers, 18
limits, 17
"never print" operations, 19
"print always" operations, 19
Print Answer key, 20
print enable/disable, 158-159
Print Entry key, 20
while loading program, 103, 131
while running program, 103, 131

Problems and examples

general examples, 163-167
problems
linear regression, 172-180
means, 194-199
Problems and examples (continued)

problems (continued)

multiple and partial correlation coefficients, 180-192
probability for 2x2 matrix, 192-194
SD, mean, standard error, 168-169, 171
z-statistic, normal possibility, 169-170, 171-172

programming examples, 199-207

Program

flow charts, 109, 116, 204
intermediate example, 108-114
introductory example, 100-103
introductory reasoning, 95-100
keyboard program access, 90-91, 160-161
loading, 131-133
print control, 131
print symbols, 103, 132-133
program memory, 99
program steps, 128

Punch card/mark sense reader, 213

Pythagoras, 92
Recall decimal point, 159

Recall from register
- indirectly, 147-150
- machine level (open channel) 156
- main data, 32
- scratch pad, 26-27

Reciprocal (invert)
- as multiple root, 198
- key \( \left( \frac{1}{x} \right) \), 25-26
- used for constant divisor, 35

Registers
- add to registers, 160
- cleared with On, 16
- main data, 31-33 (and see separate listing)
- register arithmetic, 28-31, 33
- scratch pad, 26-28 (and see separate listing)

Register usage charts
- for data summation, 46, 58, 62
- for Groups, 46
- for linear regression, 66
Regression, linear, 65-72
  three-variable, 69-72
  two-variable, 66-68

Repeated arithmetic, 40

Repeated equals, 38-40

Reset
  automatic with On, 16
  other functions, 17

Resume
  with a program, 104
  with branch, 129-130
  with step, 150-151

Run-Step-Load switch, 17

Scratch pad registers
  add to register, 160
  arithmetic in and out of, 28-31
  decimal point as a register, 28
INDEX -19-

Scratch pad registers (continued)

description, 26-27
example of use, 27
print symbols of, 27

Second function key, 41, 42-43

Set Group, 44-46
data storage chart, 46

Setting flag 1, 152
flag 2, 154

Square
code, 154
with times-equals, 39

Square root key, 26

Standard deviation
n version, 54-55
n-1 version, 53-54
Standard error

  n version, 54-55

  n-1 version, 53-54

Standby mode, 16

Statistical functions, 44-90

  chi-square, 85-88

  factorial, 90

  independent, dependent t-statistic, 76-85

  line, 72-76

  linear regression, 65-72

  normal distribution, 56-57

  permutations and combinations, 88-90

Set Group, 44-46

  standard deviation, mean, standard error (n), 54-55

  standard deviation, mean, standard error (n-1), 53-54

  sum-square, 46-52

  z-statistic, 55-56

Stepping through program, 150-151
INDEX -21-

Store in register

indirectly, 147-150

machine level (open channel), 156

main data, 32

scratch pad, 26-27

Subroutine

description, 142-146

entering with branch, 130

Subtotal, list-mode, 153

Subtract

keyboard subtraction, 24-25

list-mode, 153

Summed data, entering, 45, 72

Sum-square, 46-52

Symbolic addressing

description, 133-136

examples, 111, 113-114

symbolic address table, 136
Table of contents, i

Tape cassette unit, 214

Third function, 42-43

Three-variable data summation, 61-65

Total
   list-mode, 153
   with equals, 24-26

Turning the machine ON/OFF, plus STANDBY, 16

T-statistic, independent and dependent, 76-85

Two-variable data summation, 58-61

Typewriter, 214

Where am I?
   current location, 151-152
INDEX -23-

XY plotter, 214

z-statistic, 55-56
NOTE

If the machine is turned OFF and then quickly ON again, the IDLE light may go out and the machine become inoperative. The same thing may happen if RESET or CLEAR is initiated immediately after turning the machine ON, before the automatic clearing and resetting functions are completed. To correct this condition, turn the machine OFF and leave it off for a few seconds, then turn it ON again and give it a few seconds to complete its automatic clearing and resetting operations.