Datapoint

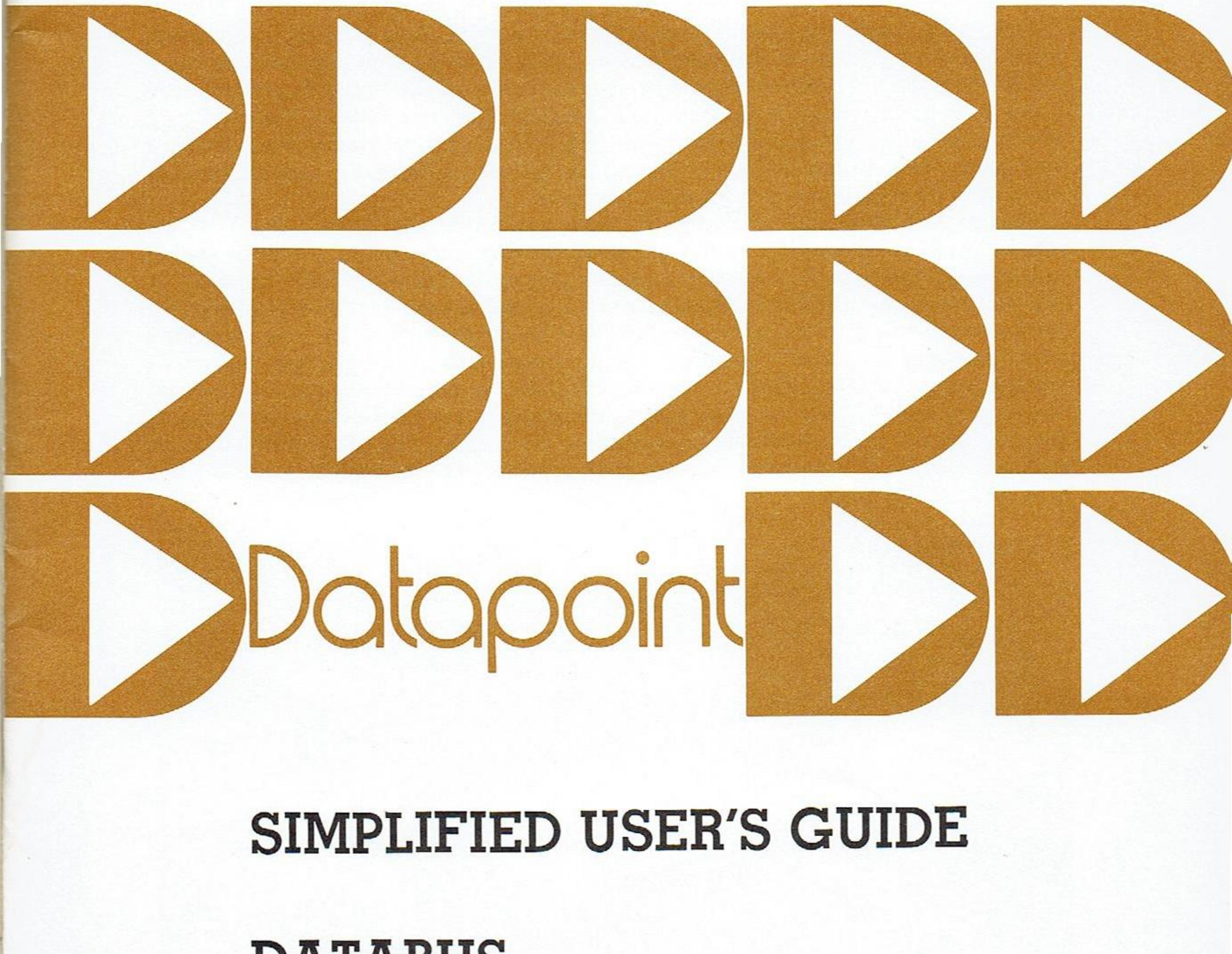
JEKK

SIMPLIFIED USER'S GUIDE

DATABUS PROGRAMMING LANGUAGE

by GERARD CULLEN

Created especially for the business professional and executive



SIMPLIFIED USER'S GUIDE

DATABUS PROGRAMMING LANGUAGE

by GERARD CULLEN

Foreword

The Datapoint 2200, from Datapoint Corporation, has established a secure reputation as the business computer system. Currently, many hundreds of Datapoint 2200 systems are being used in a wide variety of commercial data applications throughout the world. The 2200 represents a truly unique combination of capabilities, which are indicated in the terms, "dispersed data processing", "data communications", "remote job entry", "interactive computer communications," "dual tape cassettes for program and source data", and "full line of peripherals including tapes, disks and printers", all of which are descriptive of the system capability and work potential of the 2200; but the real key to the Datapoint 2200's amazing versatility is discovered in the fact that it is a fully programmable, general purpose computer.

Additionally, comprehensive software support is provided along with system hardware, which includes the family of DATABUS programming languages. This high-level language is available in 7 different versions depending upon the individual 2200 system's configuration. It enjoys a special status in the family of computer languages because it constitutes a powerful and sophisticated tool in the hands of a professional programmer and at the same time permits easy mastery and use by the non-professional programmer – such as a business executive – for applications germane to his job duties.

This booklet is written primarily for those persons in the latter category – the once-in-a-while programmers who find it useful, convenient and productive to be able to communicate directly with a desk-top computer such as the Datapoint 2200. The study of this text should, in a short time, allow you to create and operate your own programs on a 2200 system and will at the same time give you a much better "feel" for your company's total computing operation. Computers are, of course, such an integral part of today's business world that it is difficult to function effectively as an executive without knowledge of how they operate, their potential, and their limitations. The booklet also constitutes an easy introduction for professional programmers to the language and will provide a basis for reading the DATABUS Reference Manual which contains all of the booklet instructions for each DATABUS language.

Datapoint 2200 system, shown with associated printer and magnetic tape units.

TABLE OF CONTENTS

Page

Chapter 1 .... 4 How to Use This Booklet

Chapter 2 .... 5 Flowcharting

Chapter 3 .... 8 DATABUS Lesson 1

Displaying on the screen

Chapter 4 .... 14 DATABUS Lesson 2

Keying-in data

Labels

Chapter 5 .... 22 DATABUS Lesson 3

Printing

Chapter 6 .... 26 DATABUS Lesson 4

Arithmetic

Chapter 7 .... 33 DATABUS Lesson 5

Loops

Keyboard and Display Buttons

Beep

Click

Chapter 8 .... 42 DATABUS Lesson 6

Cassette Tape Operations

Chapter 9 .... 48 DATABUS Lesson 7

Subroutines

Linking up with other DATABUS programs

Chapter 10 .... 51 DATABUS Lesson 8 (Advanced Techniques)

Character Manipulations

Communications

Summary of Databus Languages

Chapter 11 .... 54 How To Run a Program On The Datapoint 2200

The Editor

The Compiler

The Interpreter

Appendix .... 60 DATABUS 2 instructions

Chapter I

How To Use This Booklet

Most professional businessmen and women reach a point in their careers when writing a computer program or demonstrating a computer method becomes important . . . or even essential.

Since its introduction, the Datapoint 2200 has offered computer and systems personnel a real desk-top computer system. In the past, many business applications demanded that programs be written in assembly language, the primary (and most time-consuming) computer language. Unfortunately, this language prevents many management and executive personnel from using a computer effectively because of its complexity.

Now, with development of a whole family of Databus languages, a person with no previous programming experience can write English-language programs with just a small amount of training. Even normally complex operations, such as creating records on magnetic tapes, printing forms, and communicating via telephone lines can be done with a minimum of effort.

This booklet does not cover all the capabilities of DATABUS, nor is it intended to make you an expert programmer. Only fundamental concepts are covered, though enough insights are given to allow further study. When you are through, you should be able to create a program of your own for any practical business purpose.

Each of the first five chapters covers some aspect of DATABUS usage. Chapter 11 deals with editing and "compiling" a program so the Datapoint can accept and run it. It is not necessary to read the entire book, for instance, if all you want is to display a message on the screen. The first lesson will arm you with enough knowledge to do that. Then you simply skip over to the section on compiling and running a program which is Chapter 11.

Read as much as you need. If for some reason, you really get hooked, write for a copy of our DATABUS Reference Manual and become an expert. There's a whole family of DATABUS languages, each with specific powers, such as communications ability or extra commands to handle the Disk and reel-to-reel magnetic tapes.The Datapoint Systems Book describes these other versions of DATABUS. The "Simplified Users Guide" is intended to give you a start towards becoming a more confident and competent Datapoint 2200 user.

And, as you go through the booklet, keep these points in mind:

Probably 50 percent of all work by professional programmers is simply discovering what management wants done.

People, unlike machines, tend to learn their jobs by osmosis as they work, just as they learn the internal politics and exceptions inherent to all business systems.

A "loose" business organization will never be so shaken up as when an uncaring and uninterested computer attempts to learn its rules, since the rules might not be well defined.

And finally, people are the critical ingredient in any computer operation, no matter what the capabilities of any particular machine might be.

Chapter 2

Flow Charting

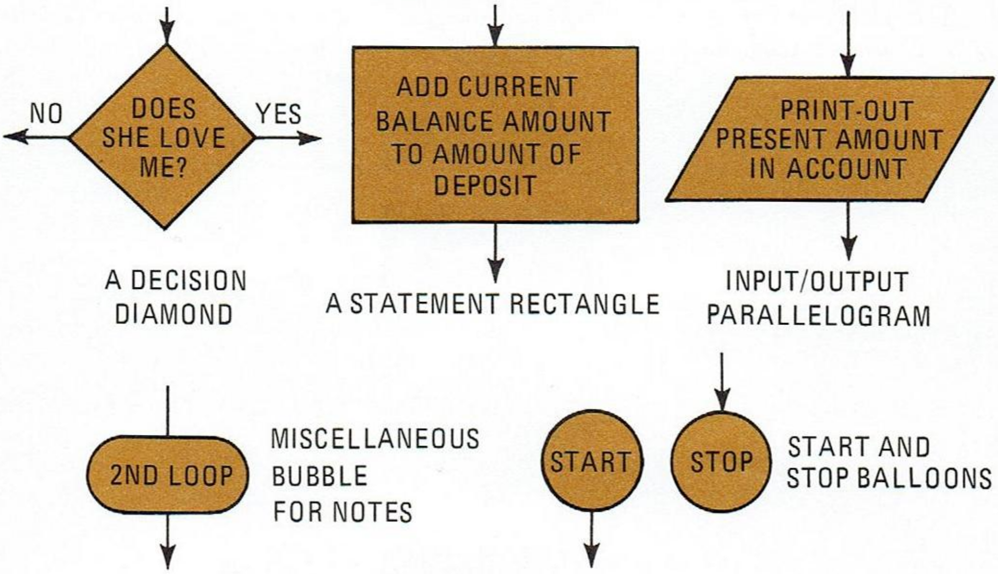
Entire books have been written about flowcharting, but we need only deal with the basics. The concept of schematically illustrating a thought process is the first step in writing an actual program. Some people attach an aura of mystery to flowcharts, but our goal is to dispel mystery.

You should be able to define almost any task by using three fundamental flowchart symbols. If your flowchart contains enough detail, your program should almost be a step-for-step textual replica. Simple programs will not always require a flowchart but, at the onset of your programming career, flowchart everything.

The following examples offer some insights into the logical thought processes that make up a flowchart.

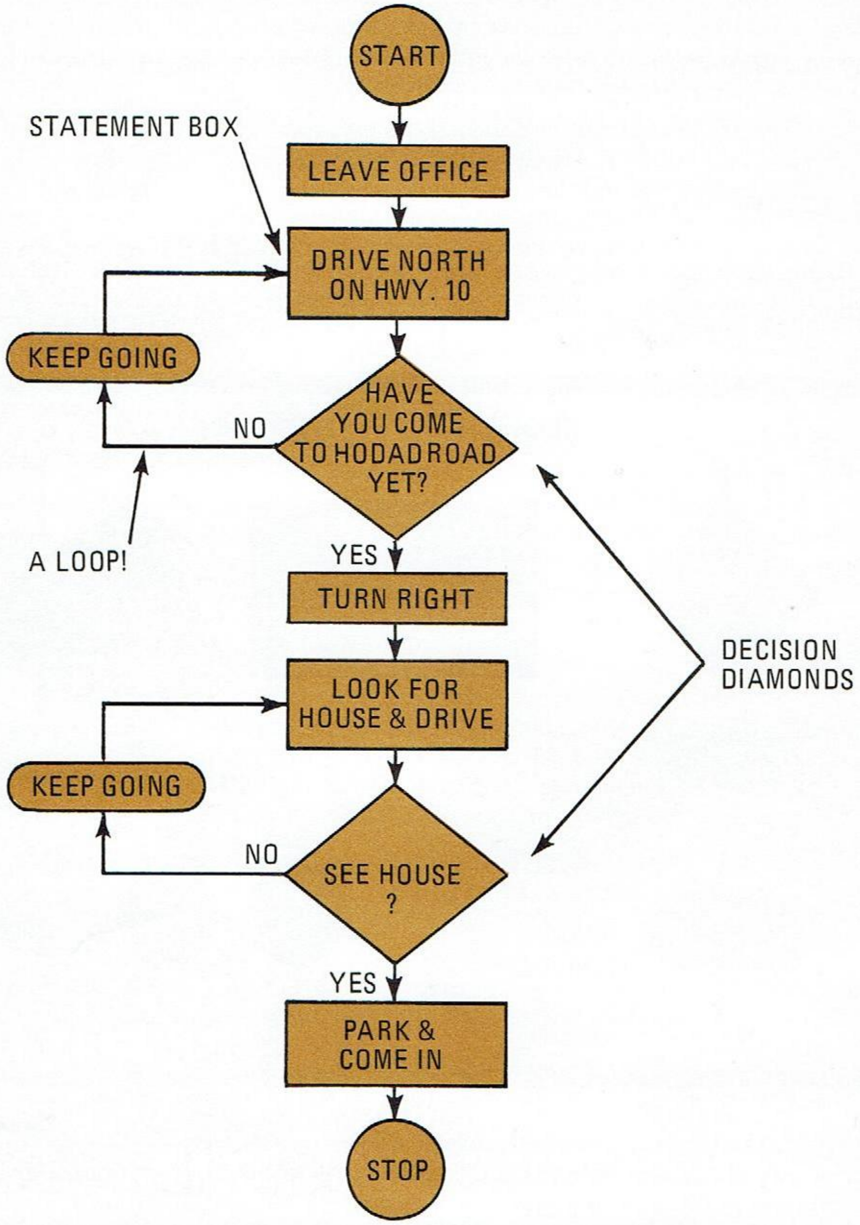
There once was a programmer who became so enamored with the computer's required logical way of thinking that his whole life revolved around small diagrams known as flowcharts.

FLOWCHART ESSENTIALS


NO
DOES SHE LOVE ME?
YES
A DECISION DIAMOND
ADD CURRENT BALANCE AMOUNT TO AMOUNT OF DEPOSIT
A STATEMENT RECTANGLE
PRINT-OUT PRESENT AMOUNT IN ACCOUNT
INPUT/OUTPUT PARALLELOGRAM
2ND LOOP
MISCELLANEOUS BUBBLE FOR NOTES
START
STOP
START AND STOP BALLOONS

Using these symbols, the programmer could astound his friends by being very lucid about the things he wanted to tell them. Note that, like computers, no two things are happening at the same time, and all possibilities are covered. Things happen one after the other in a step-by-step fashion. No daydreaming is allowed.

FRED'S FLOWCHART MAP TO HIS HOUSE



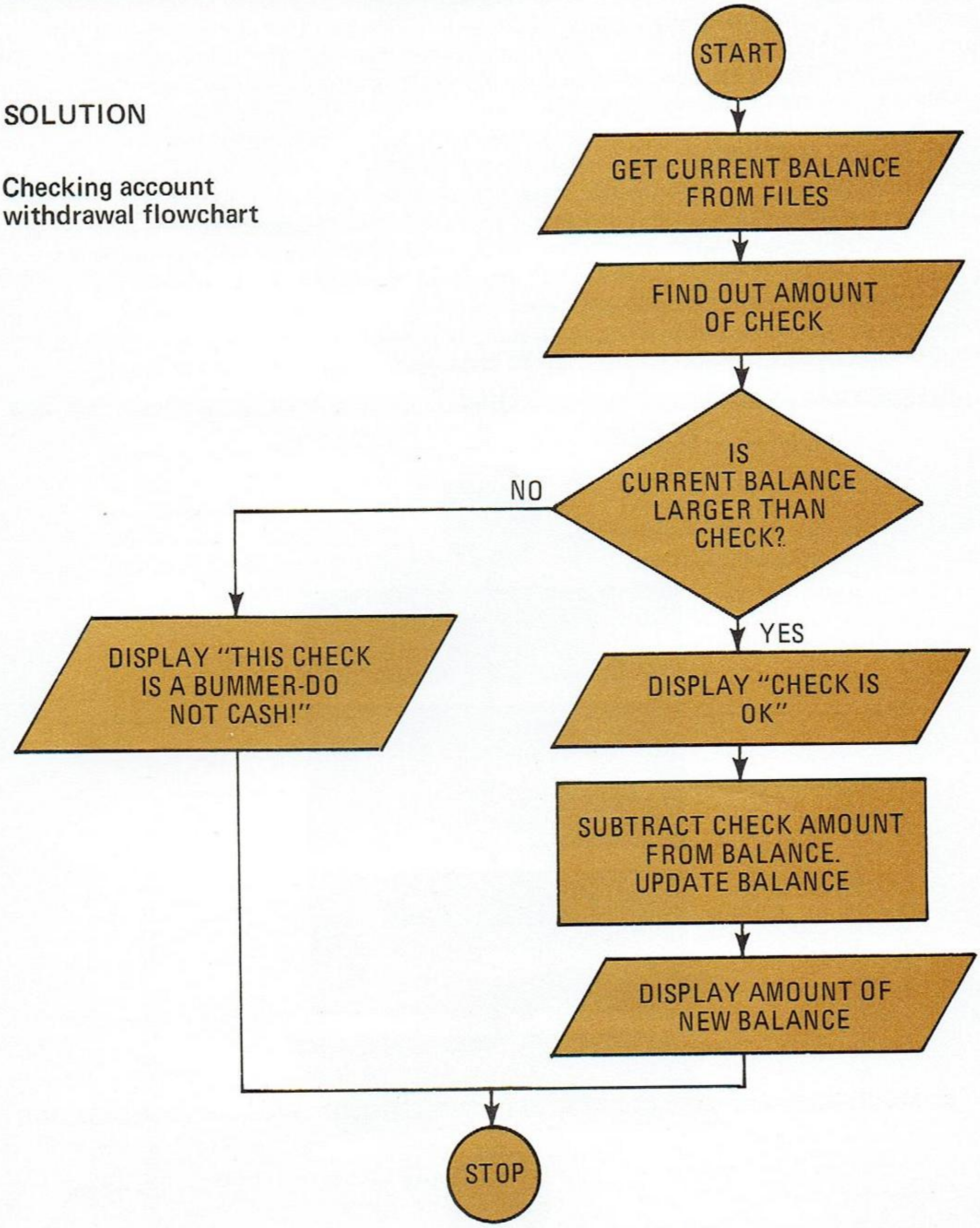
Fred could have added more to his flowchart map to cover someone getting lost, or a flat tire, or another calamity, but enough detail has been shown to demonstrate the pains one must take to insure clarity and to assure that events are taken in their logical order.

Note also that we must take one step after another to make sure our plan fits the task at hand. If you're the kind of person who outlined all his school reports before writing them, you'll have no trouble with programming.

PROBLEM: Flowchart the process of withdrawing money from your checking account. Check for overdrawing. Use decision diamonds and statement boxes.

SOLUTION

Checking account withdrawal flowchart


START
GET CURRENT BALANCE FROM FILES
FIND OUT AMOUNT OF CHECK
IS CURRENT BALANCE LARGER THAN CHECK?
NO
YES
DISPLAY "THIS CHECK IS A BUMMER-DO NOT CASH!"
DISPLAY "CHECK IS OK"
SUBTRACT CHECK AMOUNT FROM BALANCE. UPDATE BALANCE
DISPLAY AMOUNT OF NEW BALANCE
STOP

Chapter 3

Databus Lesson 1
Displaying On The Screen

One of the nicest features about the Datapoint 2200, as far as programmers are concerned, is the similarity of the various models of the system. Memory sizes can vary from 2K to 16K, but all Datapoint 2200's have keyboards, display screens, cassette tapes, and of course, the internal computers are alike as far as the DATABUS language is concerned whether they are Version I or II.

This provides many advantages over conventional minicomputer systems, where the system programmer designing a conversational-level language often has no idea what attachments or peripherals will be added to the main system. Knowing the basic configuration as a constant beforehand has allowed our people to create in DATABUS a language that already had provisions for the Datapoint's peripherals.

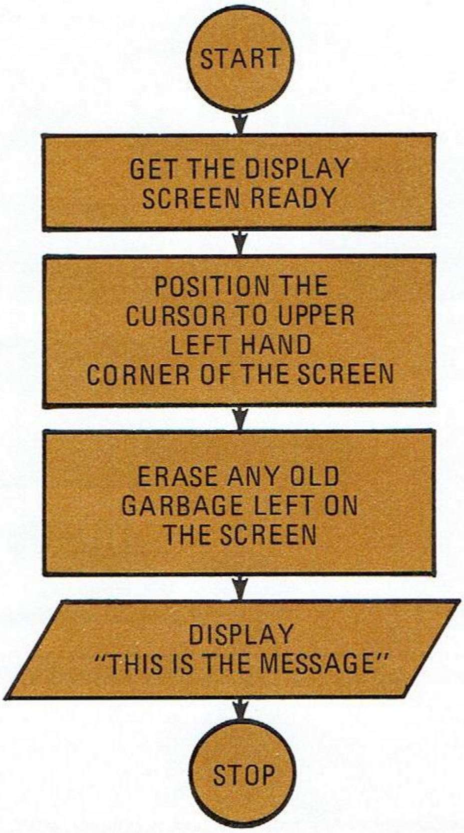
The Display Instruction

With this in mind, we'll write a short program to put some text on the screen, which is 12 lines deep by 80 columns wide. You can put your message anywhere you like.

Incidentally, you'll see the word "cursor" frequently. It's a shorthand word for the across-and-down position in which the letters will appear on the screen.

Let's write a short program to put a message on the screen. We can even flowchart it. By the way, as you study this booklet, leave lots of flowcharts lying around your desk and on lunch napkins. Before you know it, you'll be considered the staff expert on computers.

SCREEN DISPLAY FLOWCHART


START
GET THE DISPLAY SCREEN READY
POSITION THE CURSOR TO UPPER LEFT HAND CORNER OF THE SCREEN
ERASE ANY OLD GARBAGE LEFT ON THE SCREEN
DISPLAY "THIS IS THE MESSAGE"
STOP

Now, we have the flowcharts and we'll jump ahead and write the program even before you know the rules.

```
DISPLAY   *V1, *H1, *EF, "THIS IS THE MESSAGE."
STOP
```

THE COMMAND

CURSOR POSITION

ERASE THE SCREEN

THE MESSAGE! (NOTE QUOTES)

Not too bad, right? Remember that the computer works from left to right and executes each operation as it comes to it.

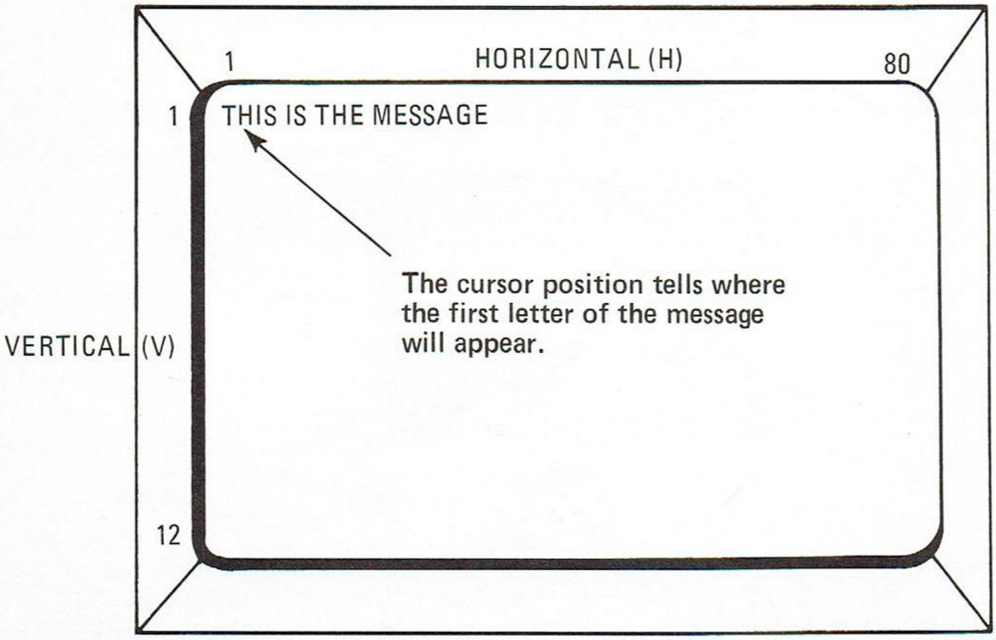
What's the *H1, *V1 stuff? You've probably already guessed that the letters H (Horizontal) and V (Vertical), prefaced by asterisks, tell the computer where to put the first letter of the message, which must be bracketed by quotes. Basically, it's just like playing a game of "Battleship" because you must give the "across and down" number of every shot. Note where there are commas and spaces and don't forget to include them.

The command *EF instructs the computer to erase the present cursor position, all characters from that position to the right side of the screen, and all lines after that. As the cursor was in the upper left hand corner the entire screen is amazingly cleared including the cursor position. It's a good idea to start every message with an erase command to wipe out old love letters or whatever was left on the screen when you arrived.

The quotation marks around the message itself tell the computer that this is what's to be displayed. Since the quotes indicate when to start and stop displaying, you can't use them as part of your message or you'll mess things up.

Below is the result of what our short program would produce. Notice that once we have defined the beginning cursor position, the rest of the message falls into place after it and no further definition of the cursor position is necessary.

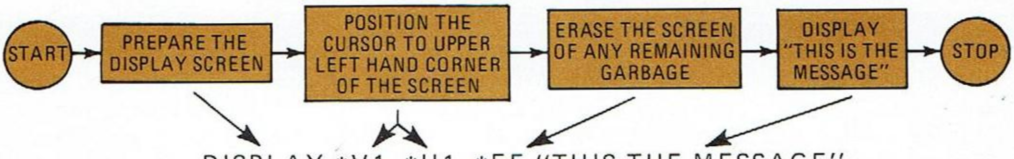
DATAPOINT 2200 DISPLAY SCREEN



The cursor position tells where the first letter of the message will appear.

If this is still confusing, let's compare the program to the flowchart. The flowchart remains the same except that it's now written horizontally.

FLOWCHART (HORIZONTAL)


START
PREPARE THE DISPLAY SCREEN
POSITION THE CURSOR TO UPPER LEFT HAND CORNER OF THE SCREEN
ERASE THE SCREEN OF ANY REMAINING GARBAGE
DISPLAY "THIS IS THE MESSAGE"
STOP

```
DISPLAY *V1, *H1, *EF,"THIS THE MESSAGE"
STOP
```

Flowcharts can be very helpful, as you can see, if you include enough details. Once you fully grasp the concept, your flowcharts will be more abbreviated.

There are always two ways to do anything in programming. We can rewrite our program and accomplish the same end result.

```
DISPLAY *H1,*V1, *EF
DISPLAY *H1,*V1, "THIS IS THE MESSAGE"
STOP
```

The first line of the program erases the screen, and the second displays the message. Obviously, this involves more work, and you should string things together whenever possible to save work.

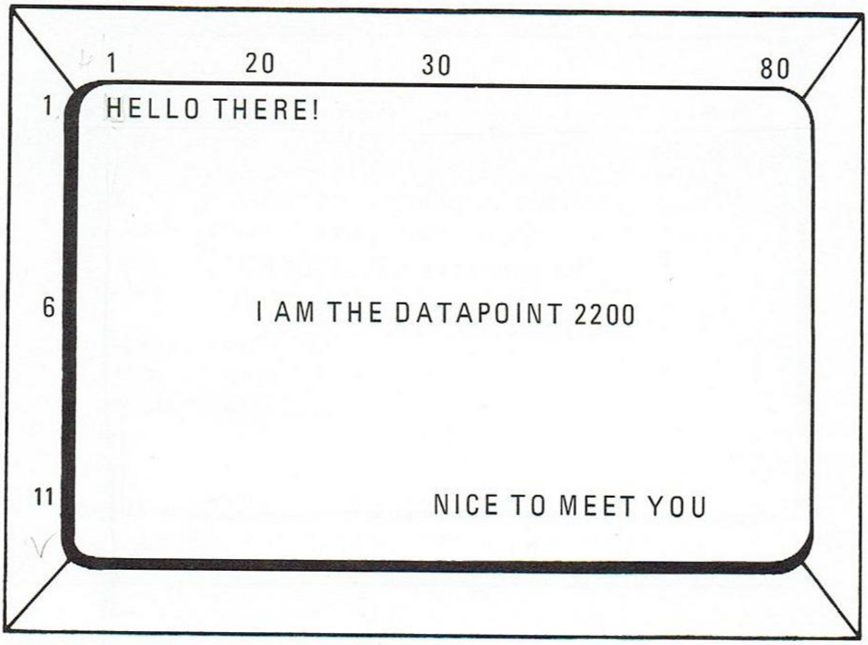
You'll see that most of the example programs have a STOP instruction at the end. This tells the computer that there are no more instructions and to stop running this program. If you forget to put it in, the DATABUS compiler* (which will be covered later) will automatically put it in for you.

To illustrate this point, let's write another program to display a more complex message. Look at the program and then look at the result.

```
DISPLAY  *H1,*V1,*EF, "HELLO THERE! ", *H20, *V6:
"I AM THE DATAPOINT 2200", *H30, *V11,  "NICE TO MEET YOU"
STOP
```

*Compiler – A special software package which automatically allows a program written in DATABUS to be translated into the basic "language" or symbols understood by the computer.

DATAPOINT 2200 DISPLAY SCREEN



Did you notice that there was a colon (:) at the end of the program's first sentence? That colon is a handy little device as it allows you to write instructions that are longer than the space of a single line. In effect, the colon tells the computer to "keep reading".

If it weren't for the colon, we would have had to write it this way:

```
DISPLAY   *H1, *V1, *EF, "HELLO THERE! "
DISPLAY   *H20, *V6, "I AM THE DATAPOINT 2200"
DISPLAY   *H30, *V11, "NICE TO MEET YOU"
STOP
```

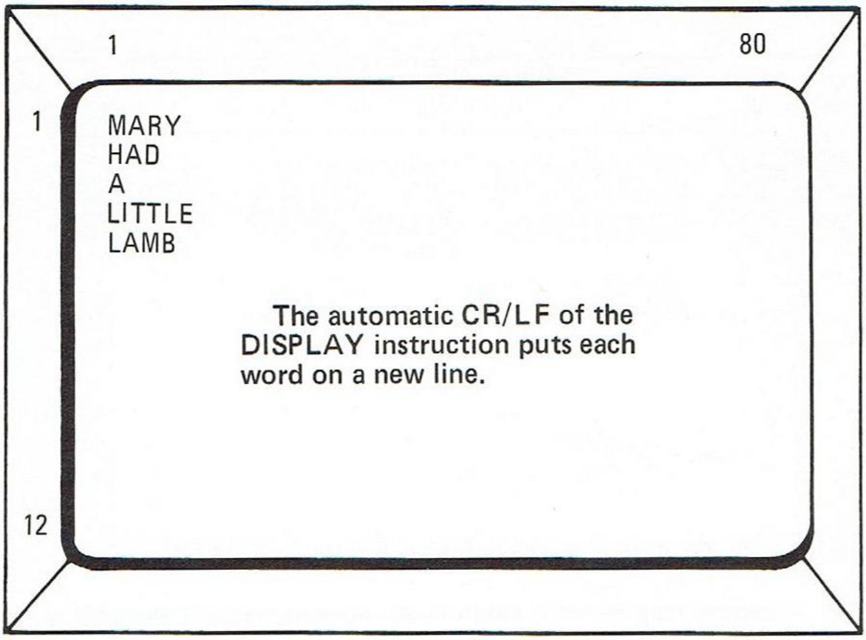
If you're paid by the word, you can write your programs the long way, but the short method with the colon works fine.

You're almost finished with DISPLAY, but we should cover one more thing. Every time the computer finishes a DISPLAY statement, it automatically sends a carriage return (CR) and line feed (LF) to the screen. The names imply the same operation that occurs when you hit the return key on a regular typewriter. Unless told otherwise, the cursor jumps to the beginning of the next line. This example should help explain things.

```
DISPLAY   *H1, *V1, *EF, "MARY"
DISPLAY   "HAD"
DISPLAY   "A"
DISPLAY   "LITTLE"
DISPLAY   "LAMB"
STOP
```

DISPLAYING ON THE SCREEN

DATAPOINT 2200 DISPLAY SCREEN



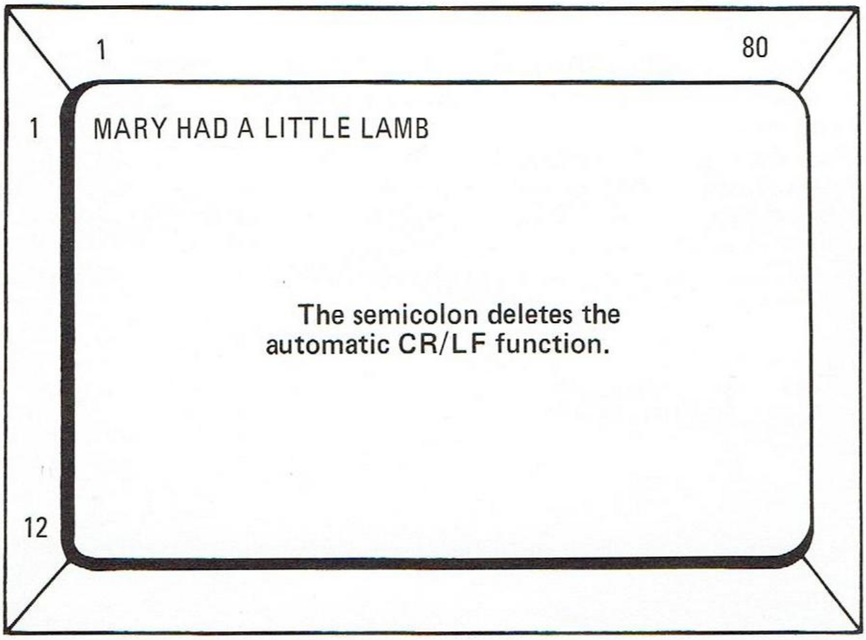
```
MARY
HAD
A
LITTLE
LAMB
```

The automatic CR/LF of the DISPLAY instruction puts each word on a new line.

At some time, you might not want this to happen. You might want to leave the cursor where it was for one reason or another. If this is the case, end the DISPLAY instruction with the semicolon (;). This cancels the automatic CR/LF function.

```
DISPLAY  *H1,*V1,*EF,"MARY ";
DISPLAY  "HAD ";
DISPLAY  "A ";
DISPLAY  "LITTLE ";
DISPLAY  "LAMB ";
STOP
```

DATAPOINT 2200 DISPLAY SCREEN



```
MARY HAD A LITTLE LAMB
```

The semicolon deletes the automatic CR/LF function.

Naturally, we could have written the sentence on the screen originally by putting the entire thing in one program sentence, but you can see how the semicolon works. Notice that each word after DISPLAY has a space after it. This was done so that we wouldn't end up with something like Maryhadalittlelamb, which is perhaps economic but aesthetically unappealing.

None of the examples used the bottom (12) line of the screen, but you should be aware that if the bottom line is used the screen is automatically "rolled up" one line. So if you write in line 12, whatever was in the top line (Line 1) rolls up and out-of-sight unless you write a semicolon which suppresses the line roll-up feature in this case.

That's it. If all you want to do is get something on the screen, jump to Chapter 11.

Here's a summary of commands we covered and some extras we didn't.

Chapter Summary

DISPLAY	the screen instruction
*H(1-80)	horizontal cursor position
*V(1-12)	vertical cursor position
*EF	erases all after cursor position in left to right, top to bottom order
*EL	erases to end of the line only
*R	moves all lines up one (rolls up) the top line is rolled off the screen
"MESSAGE"	message must be bracketed by quotes (programmers call this a literal)
:	allows instructions longer than one line
;	suppresses automatic carriage return and line feed in DISPLAY instruction
STOP	the end of the program. The interpreter program is reloaded.

Problems:

1. Write a program that will erase the screen and then display your name in the center of the screen.
2. Problem 1, but erase it after you write it, then write it again.

Solution:

1.

```
DISPLAY *V1, *H1, *EF
DISPLAY *V6, *H33, "ATTILLA THE HUN"
STOP
```

2.

```
DISPLAY *V1, *H1, *EF
DISPLAY *V6, *H35, "COPERNICUS"
DISPLAY *V1, *H1, *EF
DISPLAY *V6, *H35, "COPERNICUS"
STOP
```

Chapter 4
Databus Lesson 2
Using The Keyboard For Data Entry

If you have ever sat in on any Datapoint demonstrations, you know that the keyboard provides one of the primary means of communicating to the 2200 what it is you wish done. You also know that if the program running in the Datapoint's computer is written so as to ignore the keyboard, you can pound on the keys to a fare thee well and still have no effect on the computer's actions.

It's evident that, like the display screen, the keyboard must be "recognized" by the computer in order to enter data. The Datapoint's keyboard is almost identical to that of a standard office typewriter with the addition of an adding machine keyboard and five special function keys which we'll cover in another lesson. Upper and lower case letters can be used, from the keyboard and to the display screen.

The Keyin Instruction

With the DISPLAY instruction, we had our messages all neatly tucked between the quotes, and we effectively reserved space for the message when the program was loaded into the computer.

KEYIN – the term for keyboard entered information – is somewhat different. You don't know exactly what an operator might type in as a response to a question. But, space must be reserved to accommodate this incoming data. For instance, if the operator was to answer a query regarding a person's name via the keyboard, and that person's name was Harold Joy Hupmobile when the longest name you had planned for was Bill Ford, Harold would lose some of his name because the program would not accept such a long name.

How do we reserve space for incoming data? Simply by taking a claim on some space and putting a name on it. Many programming books use the analogy of the mailman's pigeon hole sorting-case with addresses indicated for each "hole."

That's pretty close, and you should keep the idea of a space with a label on it in mind.

We'll get back to the problem of reserving space after we examine the KEYIN instructions.

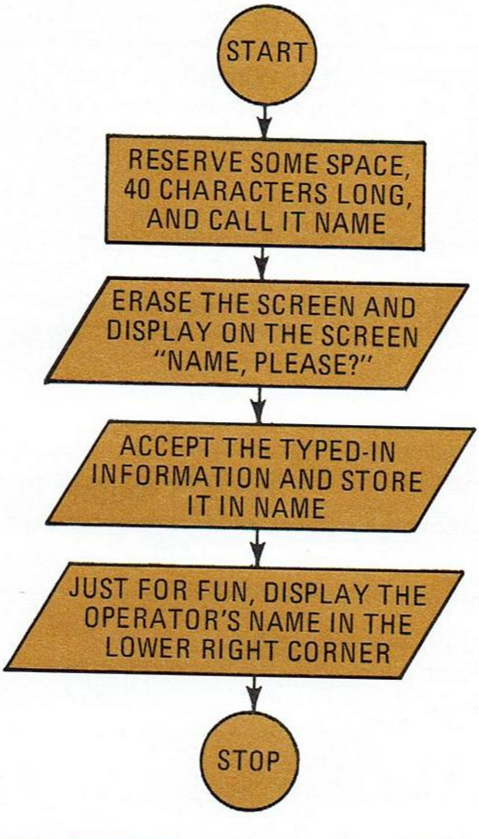
KEYIN shares several traits of the DISPLAY instruction so aside from the new concept of storing the incoming keyboard data, much of this concept will be already familiar to you.

Let's write a short program to ask the operator to type in his name. Since we're still fairly new at this game, a flowchart will help organize the task and will also impress any onlookers that this must be a very erudite and technical booklet.

And, for fun, we'll display back to the operator what his name is, as if he didn't know.

FLOWCHART

To accept keyboard data


START
RESERVE SOME SPACE, 40 CHARACTERS LONG, AND CALL IT NAME
ERASE THE SCREEN AND DISPLAY ON THE SCREEN "NAME, PLEASE?"
ACCEPT THE TYPED-IN INFORMATION AND STORE IT IN NAME
JUST FOR FUN, DISPLAY THE OPERATOR'S NAME IN THE LOWER RIGHT CORNER
STOP

You probably have noticed that this program doesn't accomplish anything useful but it should serve as a good example. Look at the instructions:

```
NAME     DIM 40
         DISPLAY  *H1,*V1,*EF,"NAME PLEASE?"
         KEYIN *H13,*V1,NAME
         DISPLAY  *H40, *V12, NAME
         STOP
```

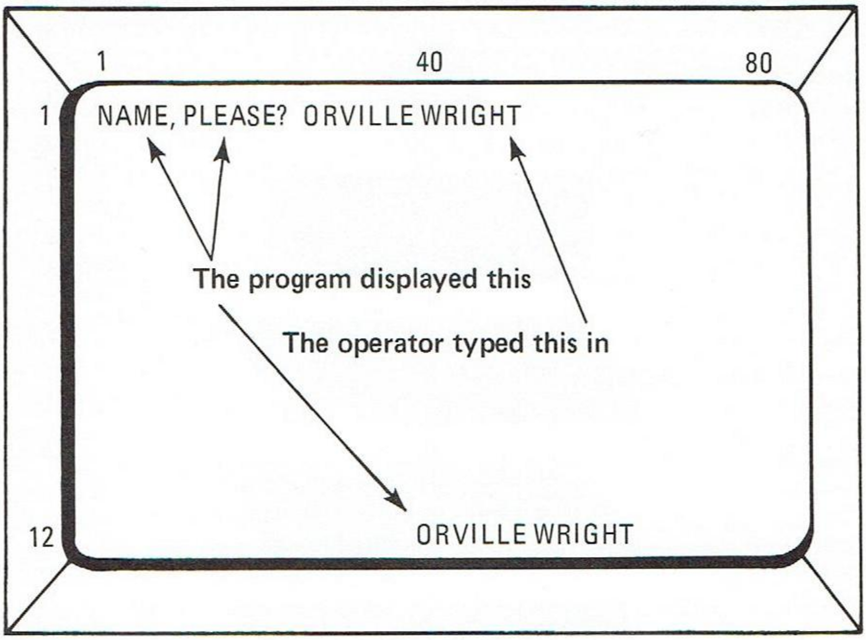
THIS IS A LABEL

THESE ARE INSTRUCTIONS

KEYING-IN DATA

Now look at the result of the program:

DATAPOINT 2200 DISPLAY SCREEN



```
NAME, PLEASE? ORVILLE WRIGHT

                                    ORVILLE WRIGHT
```

The arrows indicate what the operator typed in.

Let's look at the program and find out what happened. Notice the NAME DIM 40 statement. This looks complex, but the end result is a space labeled, NAME, with room for 40 characters. You might think of this as "dimensioning" a space in the computer called NAME to allow for a typed input of up to 40 characters.

The DISPLAY should be an old friend by now. Note that we cleared the screen with an *EF, and asked the question.

Look closely at the KEYIN instruction. There are the *V, *H symbols which are familiar to you. But, instead of a message in quotes, we now have the label, NAME. Several things happen here and all are important.

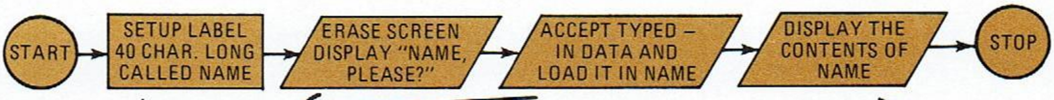
First, the NAME label has been previously defined in the NAME DIM 40 statement and the KEYIN instruction loads the data into that reserved space.

Secondly, when keyboard data is required from the operator, the cursor actually is visible as a flashing rectangle beginning at the spot defined by the *V, *H position coordinates. As the operator types in his name, the flashing cursor automatically moves over to the next adjacent character position.

Thirdly, if the operator attempts to type in more than 40 characters, including punctuation and spaces, the cursor will halt and an audio "beep" will indicate that the limit of the space allowed has been reached.

Lastly, the operator tells the computer he's finished entering his name by tapping the Enter key. The flashing cursor will disappear, and the name will be loaded into the label, NAME. Up until the time the Enter key is tapped, the name can be modified by use of the Cancel or Backspace keys.

HORIZONTAL FLOWCHART AND PROGRAM FOR KEYIN INSTRUCTION EXAMPLE.


START
SETUP LABEL 40 CHAR. LONG CALLED NAME
ERASE SCREEN DISPLAY "NAME, PLEASE?"
ACCEPT TYPED – IN DATA AND LOAD IT IN NAME
DISPLAY THE CONTENTS OF NAME
STOP

```
NAME    DIM 40
        DISPLAY  *H1,*V1,*EF,"NAME, PLEASE? "
        KEYIN  *H13,*V1, NAME
        DISPLAY  *H40,*V12, NAME
        STOP
```

One more thing. Did you notice that the DISPLAY *H40, *V12, NAME instruction had no quotes? In this case, we wanted the contents of NAME displayed. No quotes are necessary. They're only needed when we want a specific message displayed. More on this later. As you can see, the use of labels provides considerable power. Here are some guidelines concerning labels or categories of data since you'll undoubtedly want to make up some of your own:

LABEL RULES:

a. No more than 6 characters
b. Begin with an alphabetic character
c. Don't use the same name twice – you'll confuse the computer
d. All labels must be written at the first part of program
e. Each KEYIN instruction must have at least one label

GOOD	WON'T WORK
ADDRES	73SKDO (begins with number)
TARZAN	ADDRESS (too long)
LOOP	!! HELP (illegal first character)
C4	44444 (illegal first character)

If nothing else, labels can save a pile of work, and you should become familiar with them. We've discussed the DIM or "dimension" labels, and there's one more that's very useful.

```
MESSAG  INIT "NAME, PLEASE?"
NAME    DIM 40
        DISPLAY *H1, *V1, *EF, MESSAG
        KEYIN  *H13, *V1, NAME
        DISPLAY  *H40, *V12, NAME
        STOP
```

This program produces exactly the same result as the example on page 16. The label MESSAG has been "initialized" by the INIT statement to contain a specific message. The DISPLAY instruction finds MESSAG and displays its contents, in this case, the message "NAME, PLEASE?"

KEYING-IN-DATA

At first glance, it appears that there is only a slight difference between the DIM and INIT statements. It is an important difference, however, and bears scrutiny.

If you write NAME DIM 40 the computer saves 40 character spaces, fills them with the equivalent of blanks and attaches to them a name, in this case, NAME. We primarily use this when incoming data is to be stored, such as keyboard inputs.

If we wrote NAME INIT "GEORGE GORDON", the computer reserves a space of 13 characters and places GEORGE GORDON in those spaces. The INIT statement is very handy when you might want to display one message many times during a job. By using INIT and the label, you only have to define it once and then simply call it by its name, which is the label. Data can be loaded into an INIT space just as with a DIM, but once you've done that, the original message is lost.

MORE ON KEYIN

Most programming languages contain an assortment of shortcuts that allow a system programmer to produce more work over a given amount of time. DATABUS is no exception. The following example illustrates a shortcut that can be done by means previously explained, but if a substantial task is planned, this technique might save time.

By this time you're familiar with our enter-your-name program:

```
NAME    DIM 40
        DISPLAY *H1,*V1,*EF,"NAME, PLEASE?"
        KEYIN *H13,*V1,NAME
        DISPLAY *H40,*V12,NAME
        STOP
```

We can write this same program and save a step by using KEYIN's power of display.

```
NAME    DIM 40
        KEYIN *H1, *V1, *EF, "NAME, PLEASE? ", NAME
        DISPLAY *H40, *V12, NAME
        STOP
```

That's pretty strange, right? But the result is still the same as the screen example on page 16. But now we have eliminated one of the DISPLAY instructions and made KEYIN do the work.

To understand this, simply remember that if you write something between quotes in the KEYIN instructions, it will be displayed, and the cursor will appear immediately after the displayed text and await the operator's response.

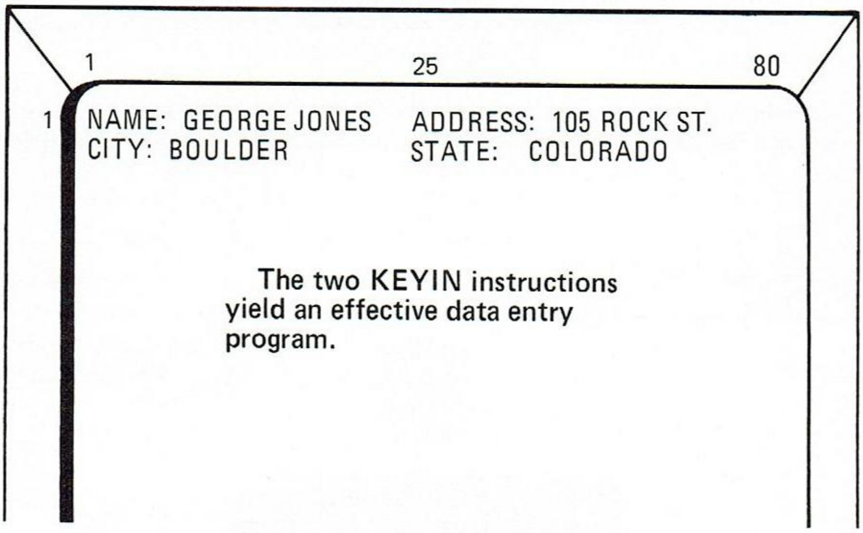
Why have this feature in KEYIN? Many application programs involve a fill-in-the-form technique, and this helps speed up the process. Here's a short program that asks the name and address of the operator.

```
NAME      DIM 40
ADDR      DIM 40
CITY      DIM 40
STATE     DIM15
          KEYIN *H1, *V1, *EF, "NAME: ", NAME, *H25, "ADDRESS: ",ADDR
          KEYIN "CITY:", CITY, *H25, "STATE: ", STATE
          STOP
```

Aside from the labels, the program uses only one type of instruction. This is confusing but follow through the KEYIN instruction. Notice that it displays and then waits for the operator to keyin the information.

The result appears as this:

DATAPOINT 2200 DISPLAY SCREEN

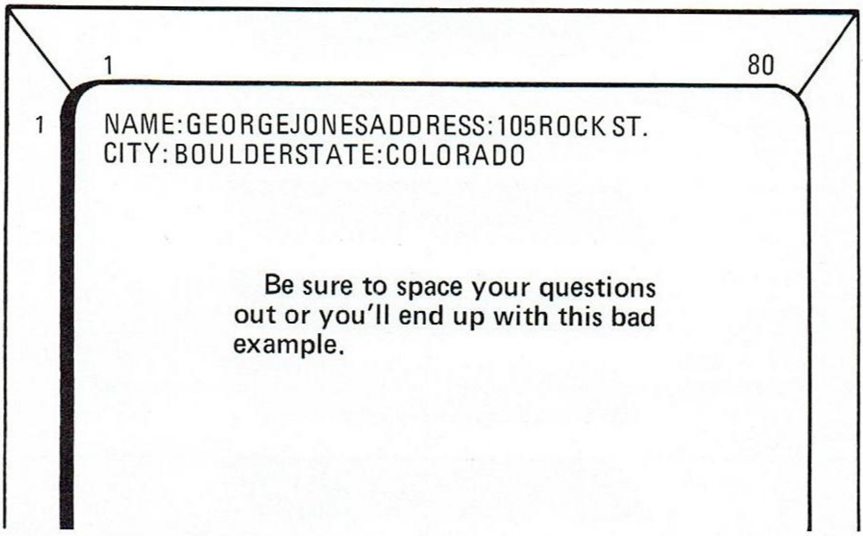


```
NAME: GEORGE JONES     ADDRESS: 105 ROCK ST.
CITY: BOULDER          STATE:   COLORADO
```

The two KEYIN instructions yield an effective data entry program.

We used the display power of KEYIN to show the operator the questions. Note that the colon used was inside the quotes – don't confuse it with the colon that allows you to write instructions larger than one sentence. The second KEYIN instruction has no V&H commands. Why? The auto CR/LF function put the cursor at the beginning of the next sentence. Note also that to move the cursor across the screen, it is not necessary to redefine the vertical (V) position.

DATAPOINT 2200 DISPLAY SCREEN



```
NAME:GEORGEJONESADDRESS:105ROCK ST.
CITY:BOULDERSTATE:COLORADO
```

Be sure to space your questions out or you'll end up with this bad example.

This is one of the features which makes DATABUS so easy for working up an applications program.

KEYING-IN-DATA

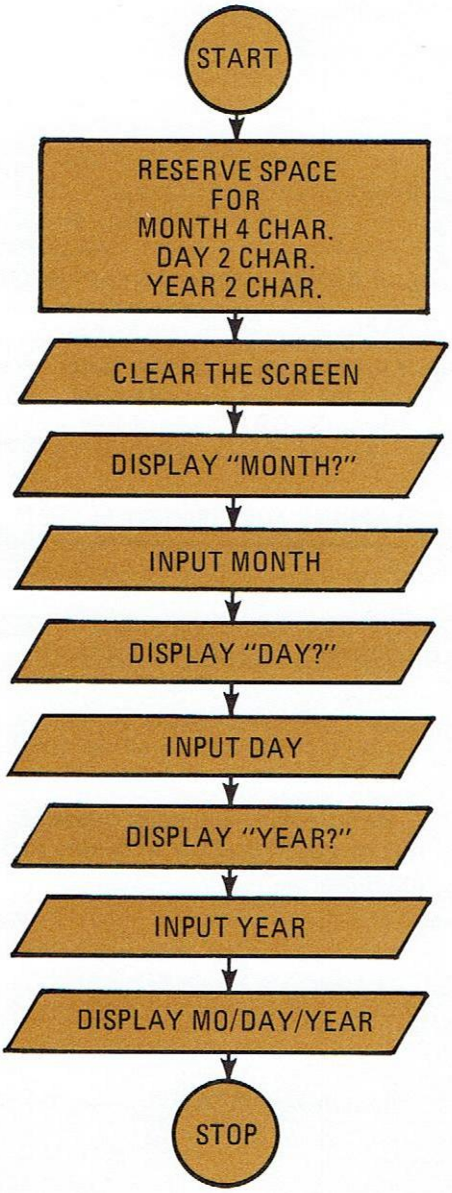
Chapter Summary

KEYIN	Accepts Data From Keyboard And Stores It In Labeled Areas
*H	Horizontal Location On Screen
*V	Vertical Location On Screen
	(See DISPLAY for more features)
Label DIM Amount	Names and reserves space for input data
Label INIT "MESSAGE"	Allows storage of message and later use by label and name

Problems:

1. Ask for the data to be typed in as answers to three questions, month?, day?, year?, . Display the date back in a MO/DAY/YR format.

Solution:



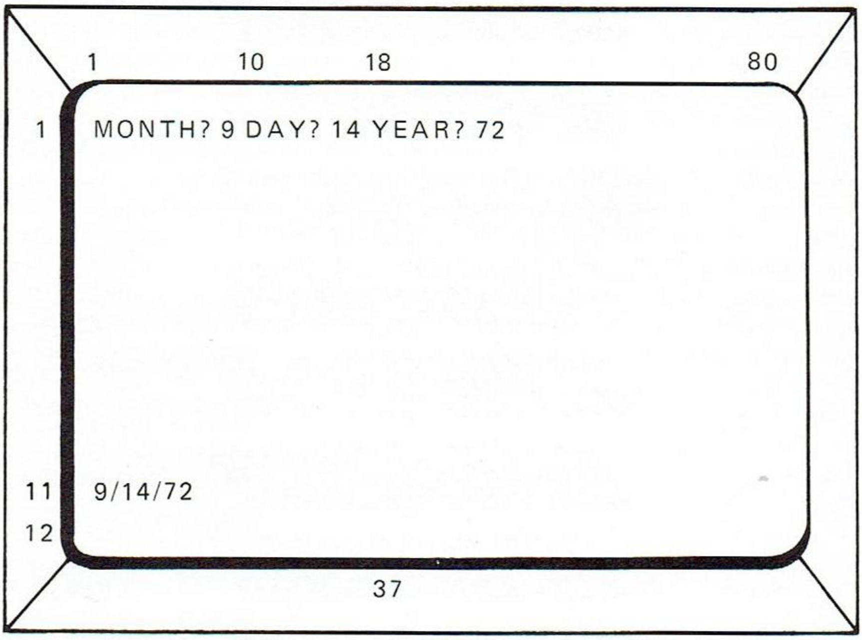
PROGRAM

```
MONTH      DIM 2
DAY        DIM 2
YEAR       DIM 2
           DISPLAY *V1, *H1 "MONTH? "
           KEYIN *V1, *H8, MONTH
           DISPLAY *V1, *H10, "DAY"
           KEYIN *V1, *H14, DAY
           DISPLAY *V1, *H18, "YEAR?"
           KEYIN *V1, * H24, YEAR
           DISPLAY *H1, *V11, MONTH, "/", DAY, "/", YEAR
```

2. Write problem Number 1 using only one DISPLAY instruction. Also, take advantage of the automatic carriage return and line feed that each KEYIN and DISPLAY instructions gives to save some work.

```
MONTH      DIM 2
DAY        DIM 2
YEAR       DIM 2
           KEYIN *V1, *H1, *EF, "MONTH?", MONTH, *H10, "DAY?", DAY:
           *H18, "YEAR", YEAR
           DISPLAY *H37, MONTH, "/",DAY,"/", YEAR
           STOP
```

DATAPOINT 2200 DISPLAY SCREEN



```
MONTH? 9 DAY? 14 YEAR? 72

9/14/72
```

Chapter 5

Databus Lesson 3
Printing the Data

Not every 2200 user is fortunate enough to have a printing unit attached to his Datapoint but, for those who do, this chapter will tell you how to go about getting your programming creations printed out.

At the time of this writing, four printers are available with the Datapoint 2200 and there will be more as new devices are introduced. It doesn't make much difference which printer you are blessed with except that it affects the line width which is usually either 80 or 132 columns. Columns are the number of spaces across.

You might even have a vintage teletype machine connected with your Datapoint through a communications adaptor or a high speed line printer. In either case, with the DATABUS language, the differences are taken care of automatically when you configure the compiler and the interpreter. All this is explained with striking clarity in Chapter 11.

So, for the moment, all you have to worry about is the number of characters across the page format of your printer unit and perhaps the availability of such goodies as a top-of-form feed, which we'll cover later.

The Print Instruction

If you have read and digested Chapters 2 and 3, this will be familiar stuff to you. The instruction PRINT follows this general pattern of DISPLAY and KEYIN.

For purposes of discussion, we will assume you have a printer with 132 columns, and further that it is an impact printer. Impact means the printer works like a typewriter – the letters are formed by smashing metal letters coated with ink (or through a carbon ribbon) onto paper. Not that this is of serious concern to us but there must be 50 ways of making data processing printers. Some squirt liquid ink at the paper while others use a heat technique. So be happy with what you have.

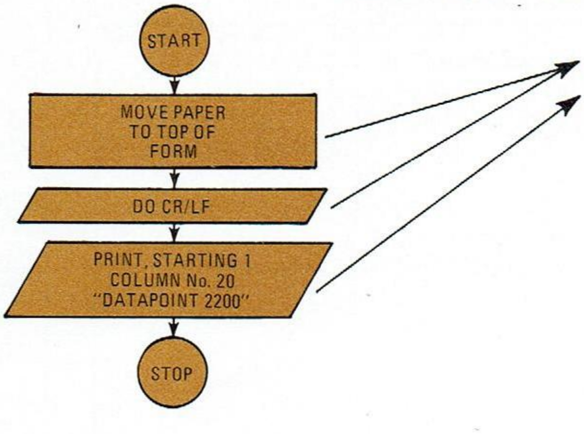
Before we write a program, consider the printer sitting next to you. Observe that there will be sprocket-holes along the sides, and the paper will be fed by moving the sprockets. Or, it might have a paper feed mechanism with a rubber roller and operate via friction as does a typewriter. Most of the expensive types with sprocket-holes contain a feature named "top-of-form". If you tell the printer to find the top of the form it will advance the paper a certain amount so that the printing mechanism will start at the upper left hand corner of the page. In most cases the paper will have perforations to delineate pages.

If you have the rubber roller friction feed type, then the top-of-form command will be meaningless to it, and you will have to position the paper manually to guarantee enough space from the tear-off.

One more thing, the last person to use the printer might not have left the print mechanism in column number 1 (the left hand side of the page), so we will want to make sure we get it over there before doing anything else with the printer.

Now let's write a short program to do something with the printer. First, of course, we'll make a flowchart and then write the program.

PRINT EXAMPLE 1.


START
MOVE PAPER TO TOP OF FORM
DO CR/LF
PRINT, STARTING 1 COLUMN No. 20 "DATAPOINT 2200"
STOP

```
PRINT *F
PRINT *20, "DATAPOINT 2200"
STOP.
```

top-of-form

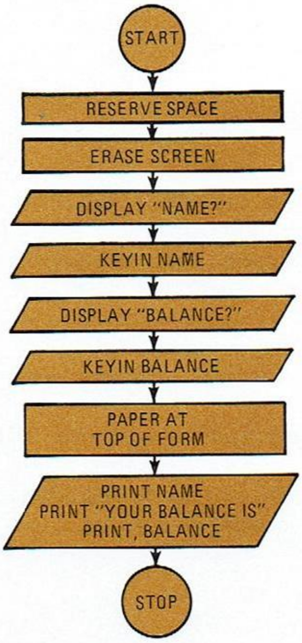
*H20

Taking the program line by line we find the PRINT *F instruction first. PRINT tells the computer to get ready to do something with the printer. *F is the command for top-of-form and, like all the DISPLAY AND KEYIN commands, it is prefaced with an asterisk. Also, like other instructions, you get an automatic carriage return and line feed unless you add the colon to suppress them. In this case we get the CR/LF.

The second line again tells the computer to PRINT and the *20 directs the printer to move to the right 20 spaces before printing the message. The message is "DATAPOINT 2200". There is no H or V required since the printer can only move across the page. Line feeds and top-of-form commands are the only way to move the paper up on most printers.

Printing is probably one of the most common data processing operations. Here is an example that could be useful to you. In this case, we want to type in a customer's name and his bank balance and have the printer type out a copy for him. While this is a limited operation, we will be adding to it later in the booklet.

FLOWCHART



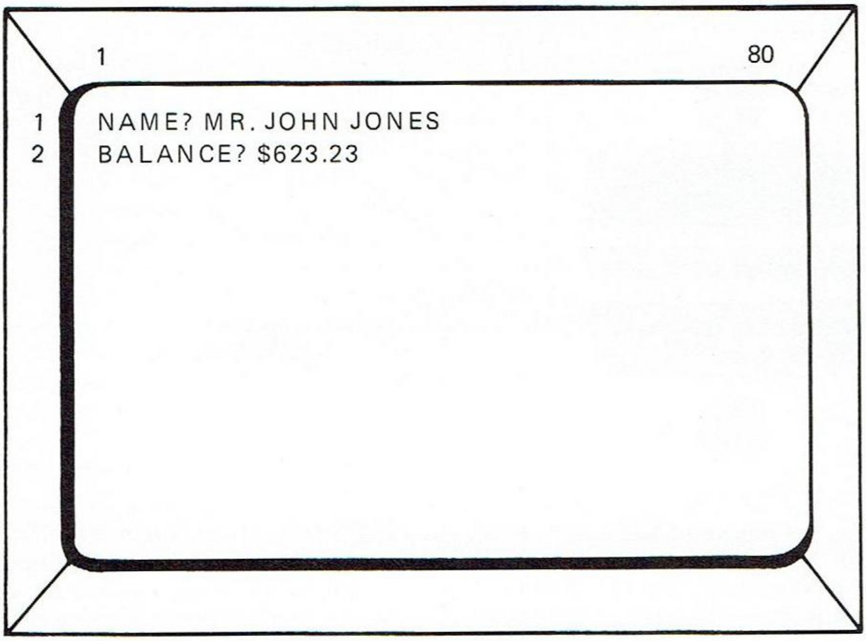
PROGRAM

```
NAME    DIM 40
BALNCE  DIM 6
        KEYIN  *V1, *H1, *EF, "NAME?", NAME
        KEYIN  "BALANCE?", BALNCE
        PRINT  *F
        PRINT  *10, "MR.", NAME:
        "YOUR BALANCE IS  ", BALNCE
        STOP
```

NOTE: Colon feature was used in PRINT instruction to conserve space. Rules are same as in DISPLAY instruction.

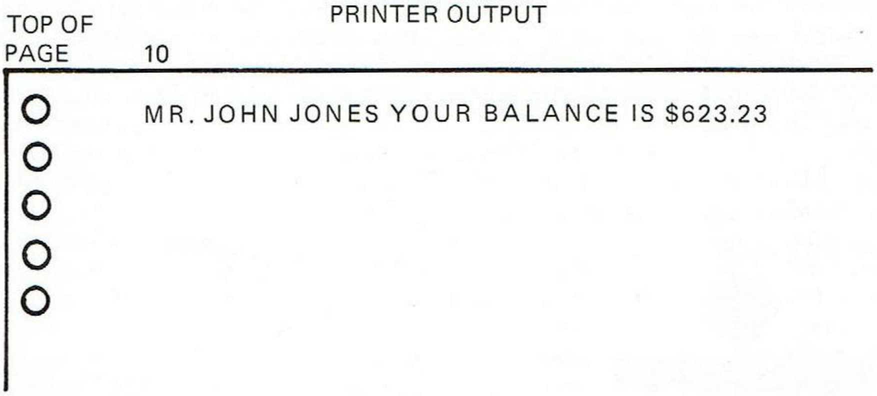
PRINTING THE DATA

DATAPOINT 2200 DISPLAY



```
NAME? MR. JOHN JONES
BALANCE? $623.23
```

PRINTER OUTPUT



```
MR. JOHN JONES YOUR BALANCE IS $623.23
```

This program uses all of what you have learned up to now, and hence you should be familiar with all the workings of the program. The DIM statements are arbitrarily assigned 40 spaces for the NAME and six spaces for the balance, which means five numbers plus the decimal point. If you anticipate larger balances for more affluent folk, the DIM for BALANCE should be increased.

KEYIN uses no new tricks, but be aware of the use of the automatic CR/LF on the printer. The next PRINT instruction spaces over 10 columns, and MR. is printed beginning in column No. 10.

If you don't have a top-of-page feature on your printer, the printer will ignore the *F but you will get the CR/LF anyway.

PRINT has a few other handy features such as line feed and carriage return. These are listed in the summary. See examples for proper spacing and placement of commas.

Chapter Summary

PRINT	Allows text to be printed. The type of printer to be used with the Datapoint 2200 can be specified during Compiler and Interpreter operation.
"MESSAGE"	Literal text must be bracketed in quotes
*F	Top-of-page
*L	Line Feed
*C	Carriage Return

Problems

1. Write a program that prints your name in the center of the page.
2. Write a program that asks name and social security number and prints this in the center of the paper.

Solutions

1.

```
         PRINT *F
         PRINT *L, *L, *L . . . . *L
         PRINT *59, "JOHN STEINMETZ"
         STOP
```

2.

```
NUMBER   DIM 9
         DISPLAY *V1, *H1, *EF
         KEYIN "WHAT IS YOUR SOCIAL SECURITY NUMBER?  ", NUMBER
         PRINT *F
         PRINT *L, *L, *L . . . . *L
         PRINT *60, NUMBER
         STOP
```

Chapter 6
Databus Lesson 4
Arithmetic

While some computers spend all their time pushing names and addresses and other such data around, it's nice if they can do arithmetic too. For instance, if we had known how to add and subtract in the previous chapter, the problem involving a bank balance could have been extended to include the actual computation.

Arithmetic is especially easy in DATABUS. The four operations – add, subtract, multiply, and divide – are demonstrated in the examples below:

```
ADD ONE TO TOTAL               (Addition)
SUB CHECK FROM BALANCE         (Subtraction)
MULT DISCOUNT BY PRICE         (Multiplication)
DIV NUMBER INTO TOTAL          (Division)
```

The operations are virtual English-language instructions. The labels, such as ONE and TOTAL, must contain only numbers now and are handled as special cases.

In the previous lessons, space for labels was handled with the DIM and INIT statements. DIM didn't mind if we loaded it with numbers, alphabet characters, or punctuation.

With numeric operations, however, the only items allowed in label space are numbers and, in some cases, a minus sign.

To accomplish this numbers-only label, a new directive is used, FORM.

FORM allows space to be reserved for numeric characters only. The statement AGE FORM 2 allows two digits to be loaded into AGE. If you try to load any more in, a beep will be heard and the number rejected. The statement PRICE FORM 5.2 would allow a number 5 places long to the left of the decimal point and two places to the right of the decimal point to be accepted. The minus sign counts as one place, also. With this in mind, PRICE FORM 5.2 holds a positive number as large as 99999.99 or a negative number of -9999.99. If the number to be used did not require a decimal point then we could have said the PRICE FORM 5. In this case, the largest value would be 99999.

FORM also allows space to be reserved and preset to some value. If the program to be written required use of the value of Pi, then we could write: PI FORM "3.1428" or FIFTY FORM "50", ONE FORM "1" and so on. You can replace these predefined numbers by loading them with other values in the program. Keep in mind the space limitations. FIFTY FORM "50" would allow a new number to replace 50 that is two numerals in length. If you anticipate replacing a preset FORM value, leave enough space, i. e., define FIFTY FORM "50.000" would keep the original value the same but open up 3 spaces to the right of the decimal point for later use.

In most arithmetic operations, one number acts upon another to form a third number, the answer. Such as:

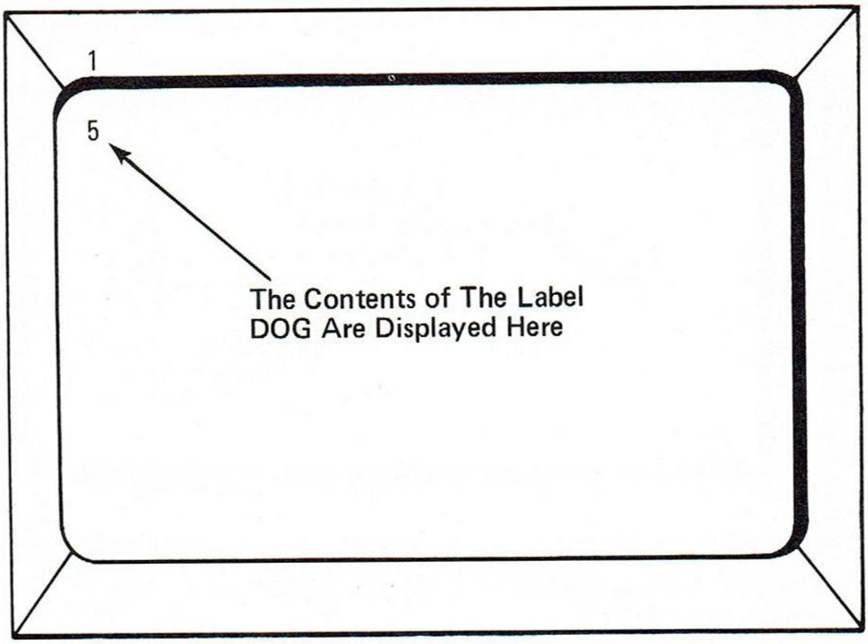
2 X 3 = 6
2 X 4 = 8

While these are names for the operators and operands, such as quotient and multiplicand, there's no real benefit from defining these unless you're fond of games of trivia.

DATABUS acts in the same way except that the answer ends up in the space where one of the numbers was originally. Read the example program:

```
CAT   FORM "2"
DOG   FORM "3"
                                      THE ANSWER
      ADD CAT TO DOG ←——————————      IS IN HERE
      DISPLAY *V1,*H1,*EF,DOG
      STOP
```

DATAPOINT 2200 DISPLAY SCREEN

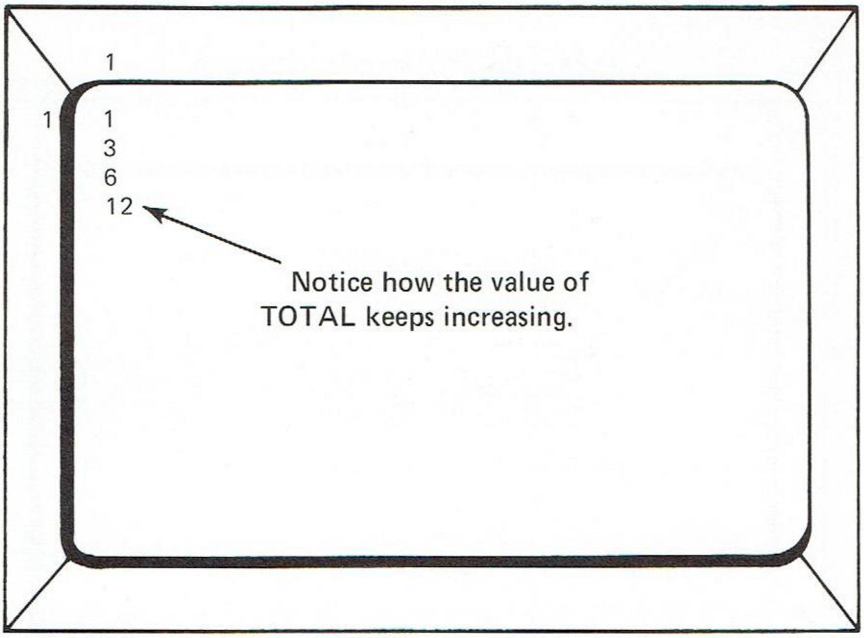


DOG is suddenly worth five, right? Yes, because now the answer is there. The original value of DOG, 3, is gone.

This trait of DATABUS can be handy in carrying totals ahead. The following example might be used in a checking account application.

```
ONE      FORM "1"
CAT      FORM "2"
DOG      FORM "3"
TOTAL    FORM  2
         ADD ONE TO TOTAL
         DISPLAY *V1, *H1, *EF, TOTAL
         ADD CAT TO TOTAL
         DISPLAY TOTAL
         ADD DOG TO TOTAL
         DISPLAY TOTAL
         STOP
```

DATAPOINT 2200 DISPLAY SCREEN



```
1
3
6
12
```

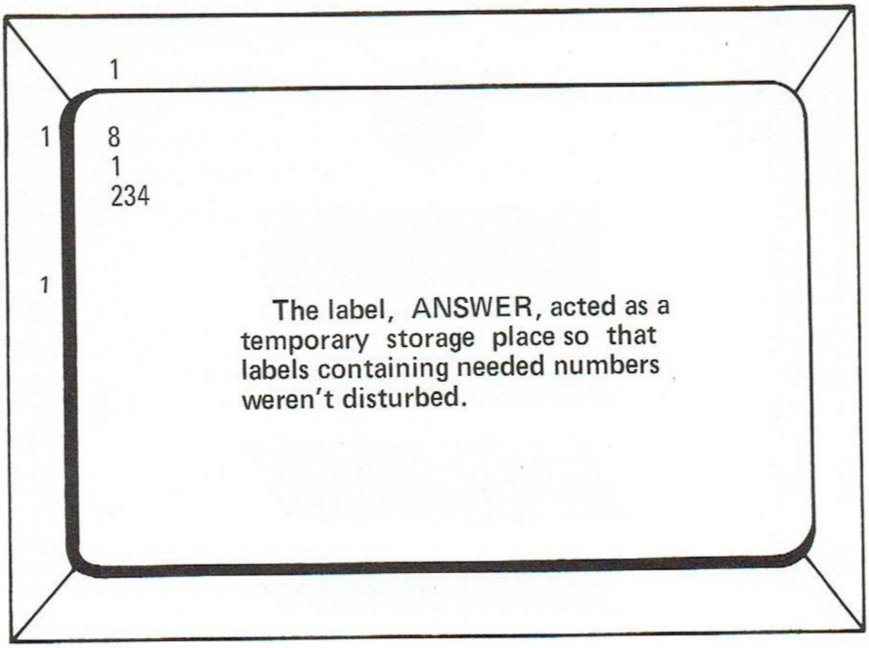
Notice how the value of TOTAL keeps increasing.

In this example, we used TOTAL to contain the answer, and the display screen reflects the changing value of TOTAL.

In some cases, you might need to keep both original numbers. We use a slightly different technique here involving an instruction called MOVE. In effect, the value we want to save is moved into another location, so that the value we need isn't disturbed. The example below illustrates this technique:

```
TWO       FORM "2"
THREE     FORM "3"
FOUR      FORM "4"
ANSWER    FORM 2
          MOVE TWO TO ANSWER
          MULT FOUR BY ANSWER
          DISPLAY *V1, *H1, *EF, ANSWER
          MOVE FOUR TO ANSWER
          SUB THREE FROM ANSWER
          DISPLAY ANSWER
          DISPLAY TWO, THREE, FOUR
          STOP
```

DATAPOINT 2200 DISPLAY SCREEN



```
8
1
234
```

The label, ANSWER, acted as a temporary storage place so that labels containing needed numbers weren't disturbed.

Notice that none of our preset values, TWO, THREE, FOUR were disturbed by the operations. ANSWER served as a temporary changeable space. We could store up to a two digit number in ANSWER, as the FORM 2 set that limit. You'll see in the display that the "8" appears in column 2 since we said ANSWER will be two spaces wide.

Incidentally, all arithmetic as well as other operations must be via predefined labels. You cannot say MULT 2 BY 5 or use any other numeric values in the instructions. Define everything in labels.

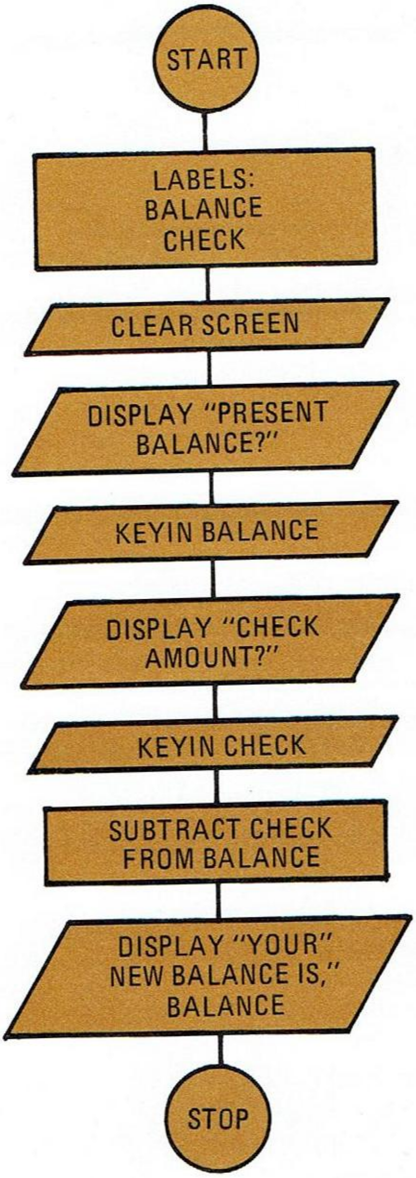
We can take the previous checkbook balance example and make it significantly more useful with arithmetic.

PROGRAM

```
BALNCE      FORM 6.2
CHECK       FORM 6.2
            KEYIN *V1, *H1, *EF, "WHAT IS PRESENT BALNCE?", BALNCE
            KEYIN "WHAT IS AMOUNT OF CHECK?", CHECK
            SUB CHECK FROM BALNCE
            DISPLAY "NEW BALANCE IS $", BALNCE
            STOP
```

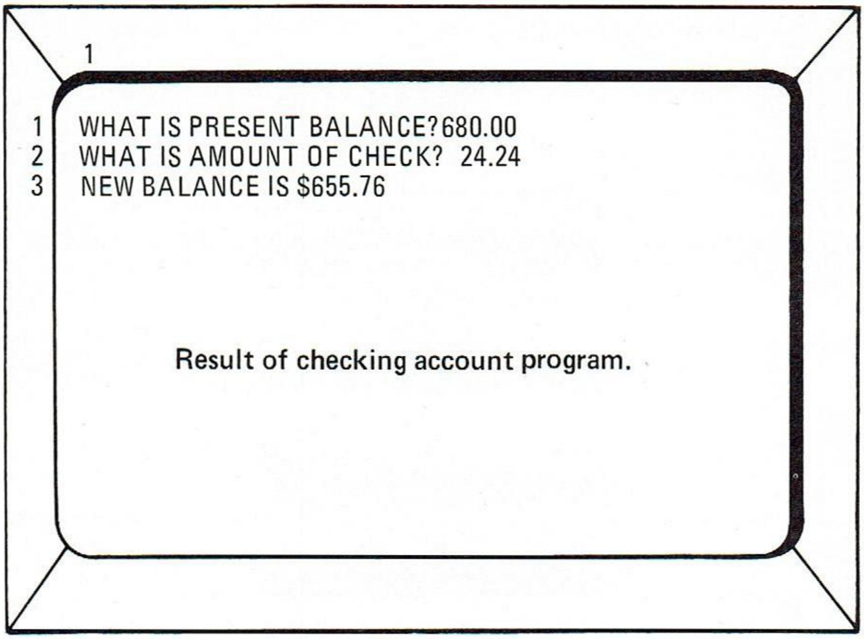
FLOWCHART

For checkbook balance example



The results of this flowchart and program are shown on the following page. Keep in mind the relation of spaces to numbers as you ask DATABUS to display such things as dollar signs or question marks. You can make your programs print out as well as your local bank can!

DATAPOINT 2200 DISPLAY SCREEN



```
WHAT IS PRESENT BALANCE?680.00
WHAT IS AMOUNT OF CHECK? 24.24
NEW BALANCE IS $655.76
```

Result of checking account program.

Occasionally, programmers encounter difficulties in using arithmetic instructions; two simple precautions should overcome most of these problems.

First, be certain you have allowed enough room for the result of the operation. If your result is larger than the space you have allowed, i.e., a result of 1000 in a FORM 3 space, the stored answer will be 000; the 1 would be lost and your answer would be misleading and you would alarm your company treasurer if you were writing an accounting program.

Secondly, in divide operations, be sure the number you're dividing into is larger than the number you're dividing, both to the left and right of the decimal place, if any. Two (2) divided into four (4) won't work. Two (2) divided into zero four (04) will work. If you have strange answers, open up some space around the number to be divided into.

One more item. When using the FORM label, only numeric characters including decimal point will be accepted. This is a handy feature to minimize operator errors, as any non-numeric characters will result in a BEEP sound.

Chapter Summary

Label FORM Number	allows a numeric variable of number to be stored.
Label FORM "Number"	allows a predetermined numeric value to be stored
MULT Label BY Label	Multiplication
DIV Label INTO Label	Division
ADD Label TO Label	Addition
SUB Label FROM Label	Subtraction
MOVE Label TO Label	Transfer value from label to label. Value transferred remains intact in first label.

ARITHMETIC

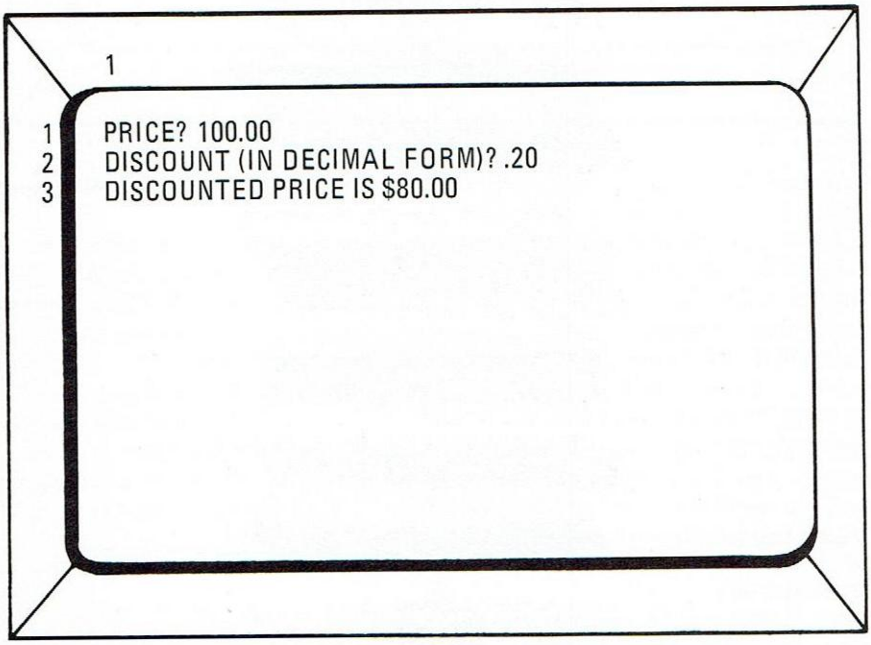
Problems:

1. Write a program to figure discount prices. Operator keys in price, discount rate, and sees new price on screen.

Answer

```
PRICE          FORM 6.2
DSCOUNT        FORM 0.3
TEMP           FORM 6.2
               KEYIN *V1, *H1, *EF,"PRICE?", PRICE
               KEYIN "DISCOUNT (IN DECIMAL FORM)?",DSCOUNT
               MOVE PRICE TO TEMP
               MULT DSCOUNT BY TEMP
               SUB TEMP FROM PRICE
               DISPLAY "DISCOUNTED PRICE IS $", PRICE
               STOP
```

DATAPOINT 2200 DISPLAY SCREEN



```
PRICE? 100.00
DISCOUNT (IN DECIMAL FORM)? .20
DISCOUNTED PRICE IS $80.00
```

Chapter 7

Databus Lesson 5

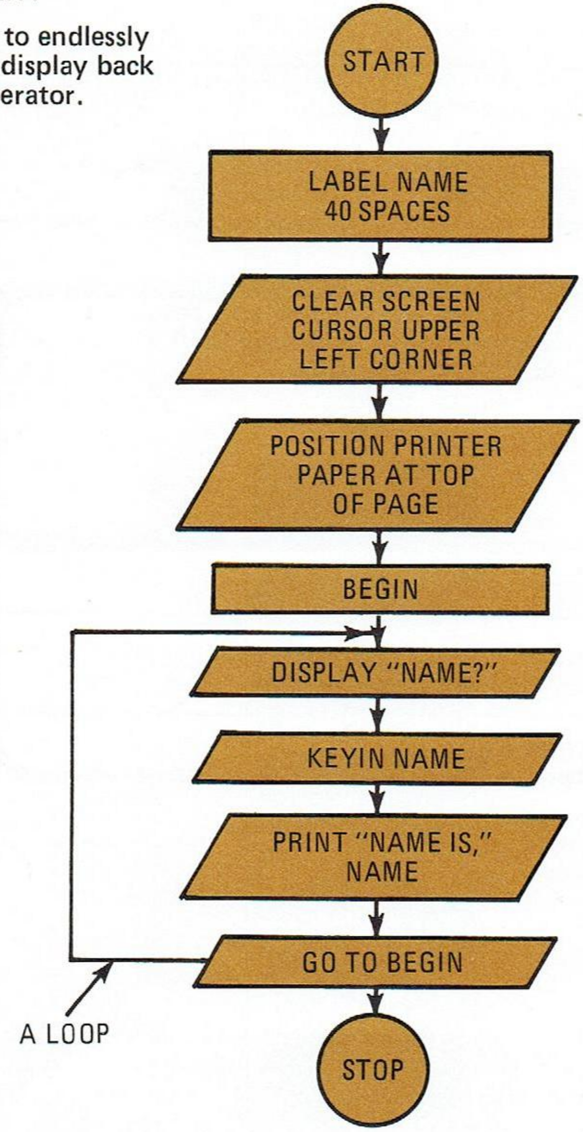
Loops

All of the programs we have written up to this point have done their job and ended. In many cases we might want to have the program repeat the task over and over. Programmers call this a loop – loops extend programming capabilities endlessly.

The simplest loop involves directing the program to go to a certain point and start working from there. The circle on a flowchart can be used to indicate a jump or "GOTO" instruction. Look over the flowchart below and then read the program that follows it.

FLOWCHART

Program to endlessly accept and display back name to operator.

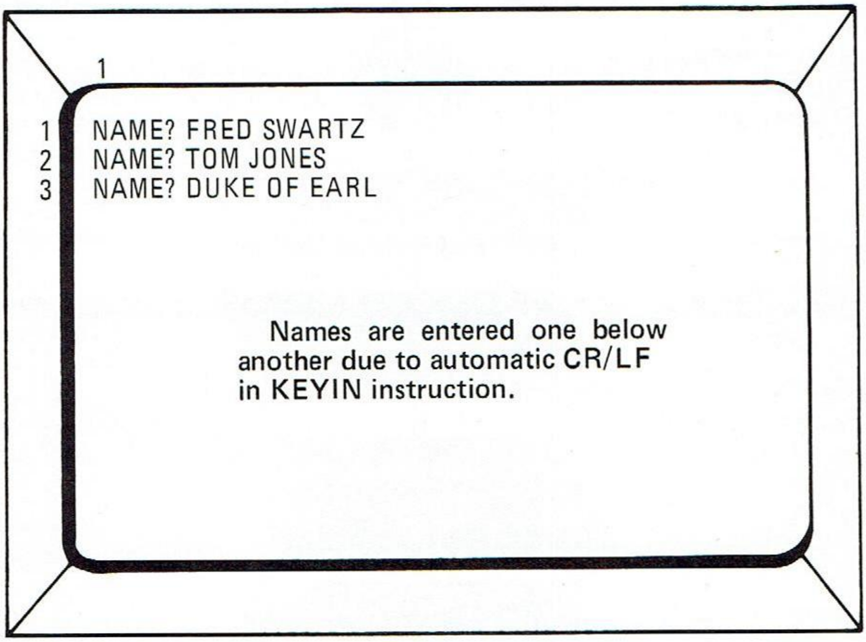


LOOPS

PROGRAM

```
NAME     DIM 40
         PRINT *F
         DISPLAY *V1, *H1, *EF
BEGIN    KEYIN "NAME? ",NAME
         PRINT*10 "NAME IS " ,NAME
         GOTO BEGIN
         STOP
```

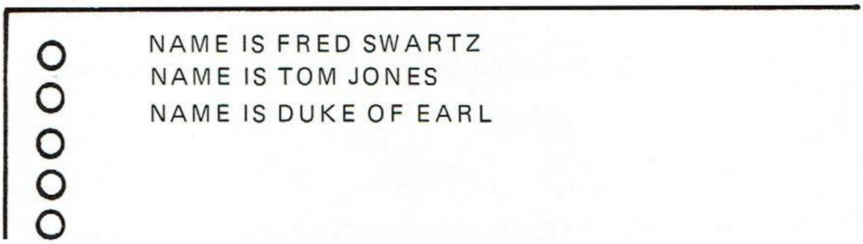
DATAPOINT 2200 DISPLAY SCREEN



```
   1
1  NAME? FRED SWARTZ
2  NAME? TOM JONES
3  NAME? DUKE OF EARL
```

Names are entered one below another due to automatic CR/LF in KEYIN instruction.

TOP OF PAGE   10   PRINTER OUTPUT



```
NAME IS FRED SWARTZ
NAME IS TOM JONES
NAME IS DUKE OF EARL
```

The GOTO instruction tells the computer to go to the place indicated by the label, in this case, BEGIN. The computer then starts following the instructions at that point. You can see that our program is endless. After one name is printed, it again asks for another name to print. There is effectively no way out of this loop except, perhaps, pressing the STOP key (which halts the computer) or pulling the plug. If you decide to run this program, the best way to halt it would be to press the KEYBOARD and DISPLAY keys simultaneously, which halts your program and turns control back to the DATABUS interpreter program. Another way to halt it would be to tap the Restart key which rewinds the back tape and reloads the operating system, which we'll cover in the last chapter, so don't worry about it now.

While endless or unconditional loops are useful in some circumstances, conditional loops are more practical. A conditional loop or conditional GOTO has limits, or goals, which must be met before the "jump" can be made. If the conditions are not met, then the computer ignores the instruction and goes on to the next one.

One of the classic computer operations is the counting loop. In this, the programmer defines a limit, and the computer keeps on counting until the limit is reached.

To determine when the limit has been reached, an operation called a compare is used. The computer compares the number against the highest allowable number, the limit, and determines if they match. After each compare, the computer can tell if the number is less than, equal to, or greater than the number compared against.

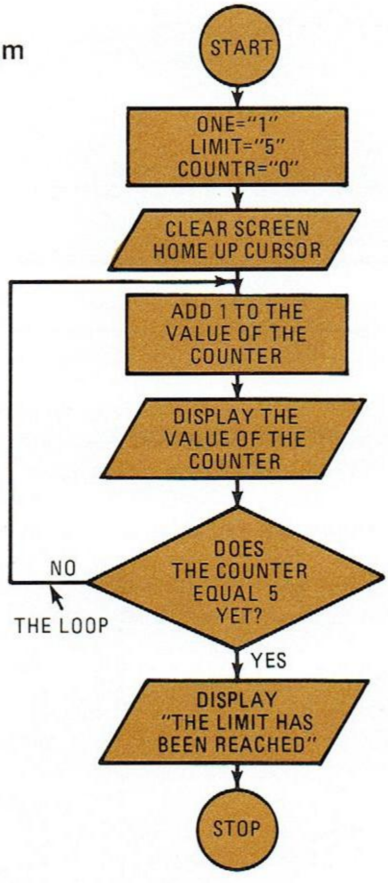
Although other, more complex compare operations are possible in DATABUS (See Chapter 10 – Advanced Databus), we'll confine our discussion to two types: COMPARE and MATCH.

Recall that we have used two types of data to store under labels: general mixed characters and numerics only. The DIM and INIT statements allow any type of characters to be stored under them while FORM takes only numerals. Much the same happens with the compare instruction. MATCH will compare two labels containing general text (DIM and INIT labels). COMPARE will only work with numeric labels (FORM labels).

The following example illustrates the use of a COMPARE instruction with a conditional GOTO. Note that the GOTO is ignored unless the condition is met.

FLOWCHART

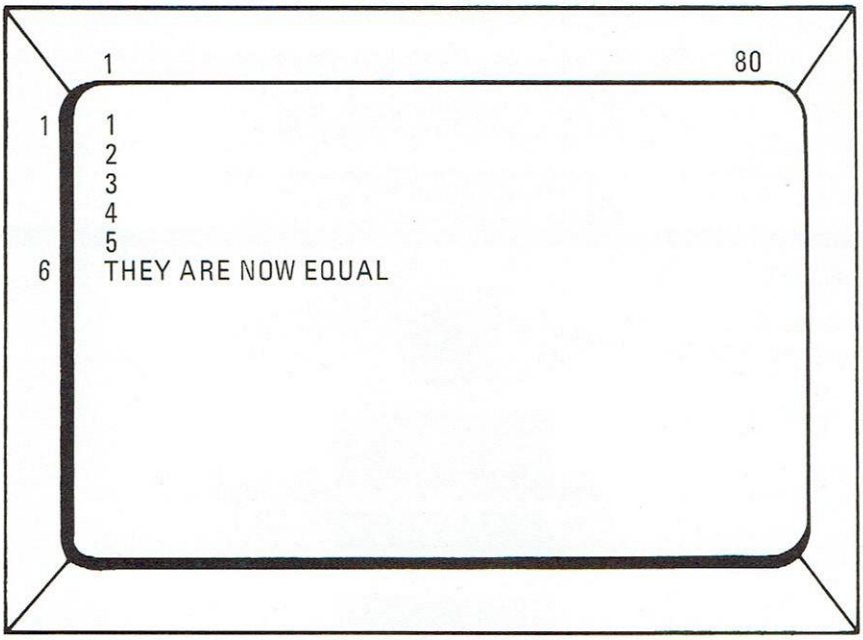
Counter and limit checking program


START
ONE="1"
LIMIT="5"
COUNTR="0"
CLEAR SCREEN
HOME UP CURSOR
ADD 1 TO THE VALUE OF THE COUNTER
DISPLAY THE VALUE OF THE COUNTER
DOES THE COUNTER EQUAL 5 YET?
NO
THE LOOP
YES
DISPLAY "THE LIMIT HAS BEEN REACHED"
STOP

LOOPS PROGRAM

```
ONE         FORM "1"
LIMIT       FORM "5"
COUNTR      FORM "0"
            DISPLAY *V1, *H1, *EF
ADDR        ADD ONE TO COUNTR
            DISPLAY COUNTR
            COMPARE LIMIT TO COUNTR
            GOTO ADDR IF NOT EQUAL
            DISPLAY "THEY ARE NOW EQUAL"
            STOP
```

DATAPOINT 2200 DISPLAY SCREEN



```
1
2
3
4
5
THEY ARE NOW EQUAL
```

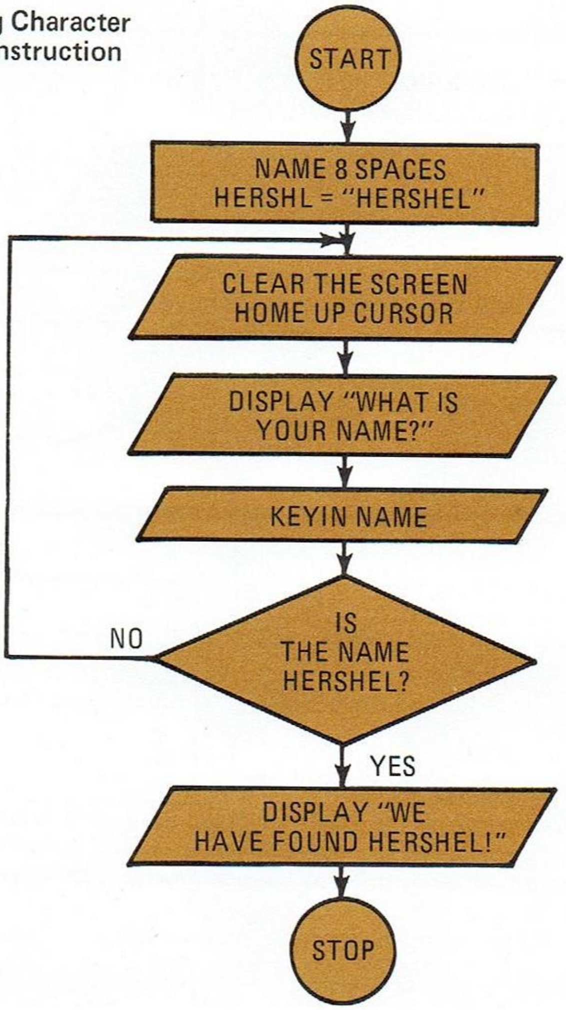
In this program, we started out with the label, COUNTR at zero and kept increasing it by the value of 1 until the LIMIT **is** equal to COUNTR. Next, if the two are **not** equal, the computer is asked to go back to another label, ADDR, and keep going. After five loops around ADDR, the value of five is attained, and program finds the two **are** equal and ignores the GOTO command. The message appears, and the program ends. Remember that the COMPARE works with numerics only.

While numerics are nice, you might want to use characters in the form of a name or answer. We could create an imaginary situation where people passing by are asked to type in their name and ask the computer to find someone named Hershel.

This use of general mixed characters involves use of the MATCH instruction. This instruction can compare the contents of two character DIM or INIT labels where the COMPARE instruction worked only with numerals.

FLOWCHART

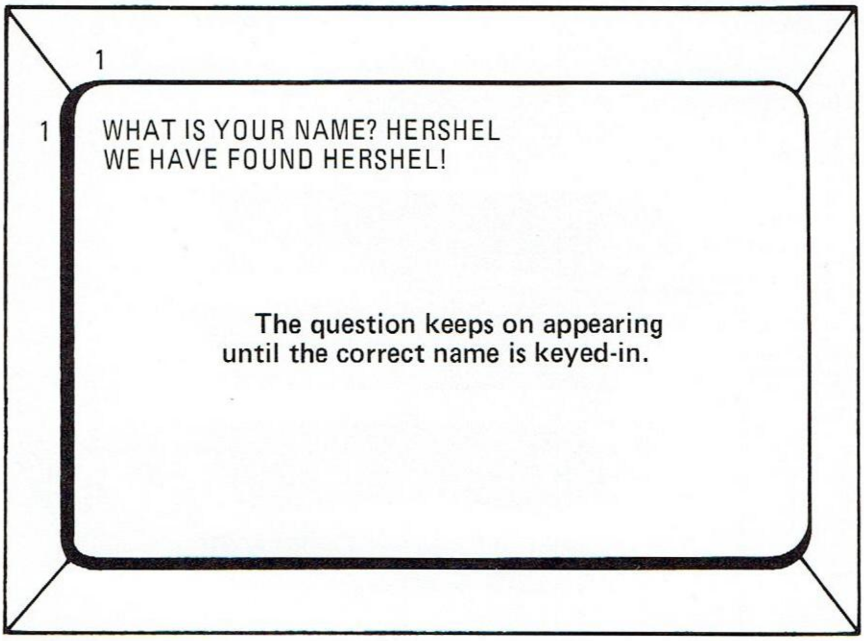
Program Using Character MATCH Instruction



PROGRAM

```
NAME        DIM 8
HERSHL      INIT "HERSHEL"
NUNAME      KEYIN *H1, *V1, *EF, "WHAT IS YOUR NAME?", NAME
            MATCH NAME TO HERSHL
            GOTO NUNAME IF NOT EQUAL
            DISPLAY "WE HAVE FOUND HERSHEL!"
            STOP
```

DATAPOINT 2200 DISPLAY SCREEN



```
WHAT IS YOUR NAME? HERSHEL
WE HAVE FOUND HERSHEL!
```

The question keeps on appearing until the correct name is keyed-in.

Note that we used the MATCH instruction. This takes the contents of the two labels and matches the first letter to the first, second to the second, and so on. If all the letters match, then the labels are declared equal and the next GOTO instruction can make use of this condition.

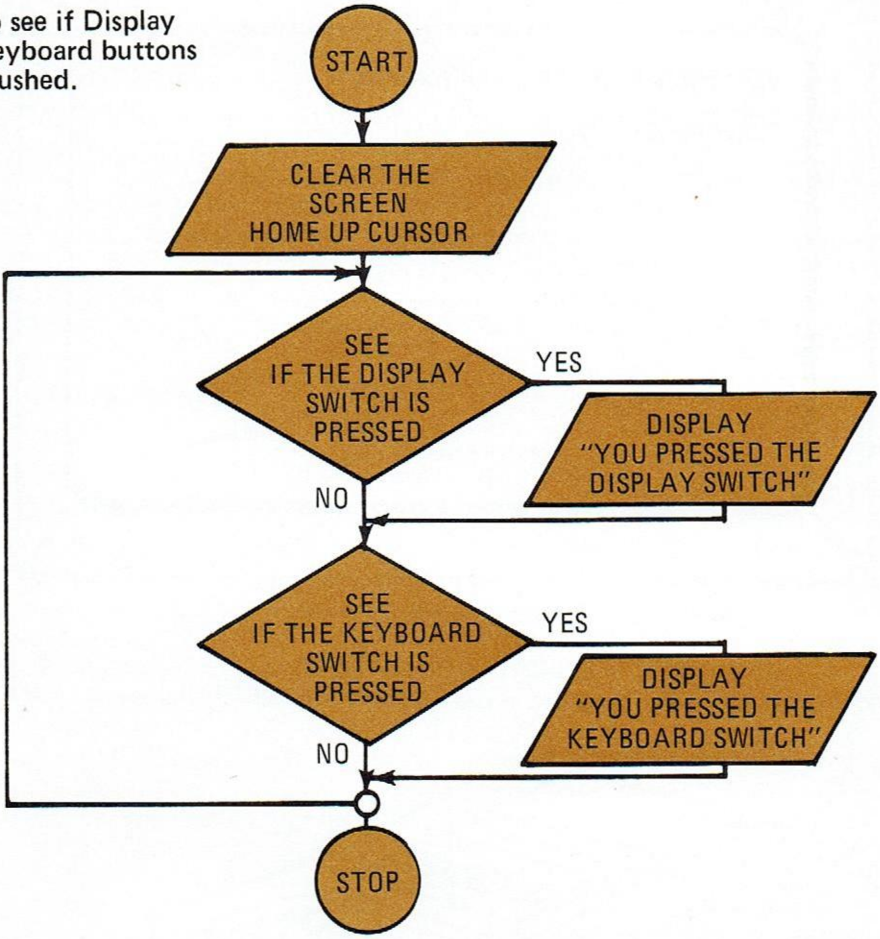
Use Of Keyboard And Display Buttons

In the lower right hand corner of the 2200's keyboard there are two push buttons named Keyboard and Display. These names are purely arbitrary as the functions have no direct relation to either the screen or keyboard. The buttons provide a handy means, however, of signaling the computer that you wish to do something.

The buttons can be used for almost any purpose, such as calling up a new screen format or performing some other task. An "equal" condition is created when one of the buttons is pushed if the instruction DSENSE or KSENSE precedes the conditional instruction. These two instructions might be interpreted as "sense the Keyboard switch" or "sense the Display switch". Sense means, in this case, to see if someone has pushed the button. The followıng program will give you an idea of how these work.

FLOWCHART

To see if Display or Keyboard buttons are pushed.

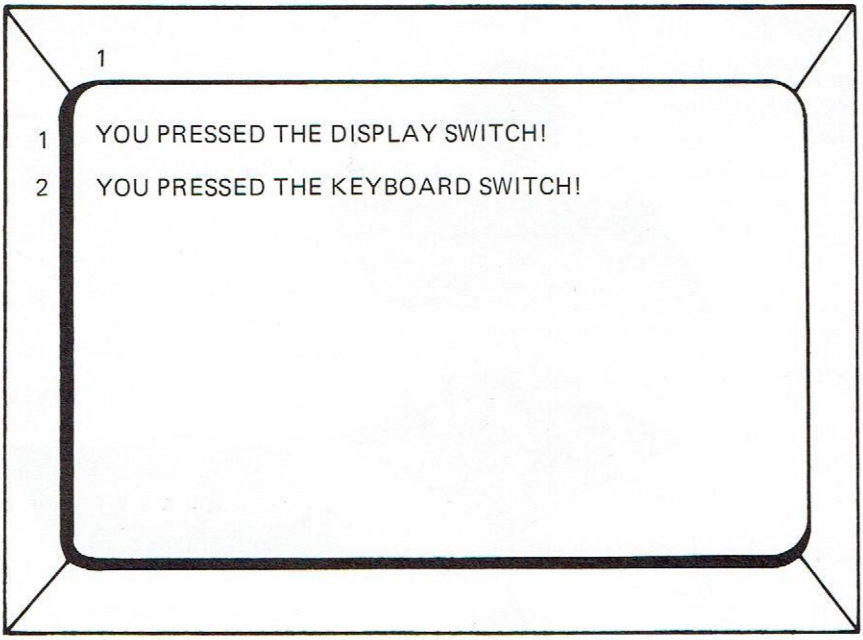

START
CLEAR THE SCREEN HOME UP CURSOR
SEE IF THE DISPLAY SWITCH IS PRESSED
YES
DISPLAY "YOU PRESSED THE DISPLAY SWITCH"
NO
SEE IF THE KEYBOARD SWITCH IS PRESSED
YES
DISPLAY "YOU PRESSED THE KEYBOARD SWITCH"
NO
STOP

PROGRAM

```
          DISPLAY *V1, *H1, *EF;
DSBUTN    DSENSE
          GOTO KBBUTN IF NOT EQUAL
          DISPLAY "YOU PRESSED THE DISPLAY SWITCH!"
KBBUTN    KSENSE
          GOTO DSBUTN IF NOT EQUAL
          DISPLAY "YOU PRESSED THE KEYBOARD SWITCH!"
          GOTO DSBUTN
          STOP
```

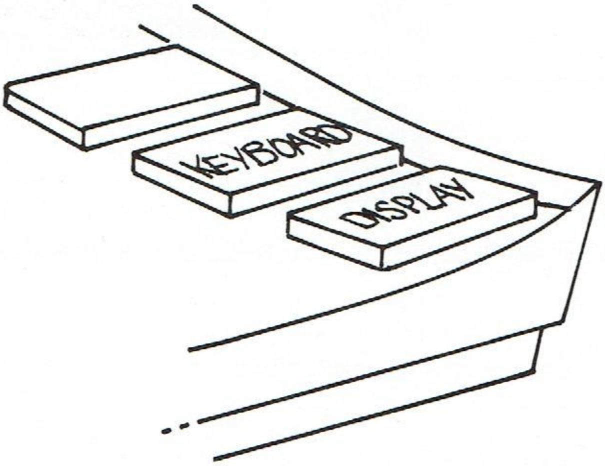
LOOPS

DATAPOINT 2200 DISPLAY SCREEN



```
       1
  1    YOU PRESSED THE DISPLAY SWITCH!
  2    YOU PRESSED THE KEYBOARD SWITCH!
```

LOCATION OF SWITCHES



The use of the switches can be left pretty much up to your imagination. It's much easier to have the operator push these buttons than to type in a certain word and then do a MATCH instruction to see what was typed in.

Beeping and Clicking

The 2200 can make quite a racket upon command by use of two DATABUS instructions, Beep and Click. It can give the operator an audio clue as to what's happening.

Beep elicits about a second's worth of audio tone from a hidden speaker inside the 2200. This tone could be used for audibly indicating the end of a job or an error.

Click, likewise, elicits a click. All KEYIN operations are automatically accompanied by this click to give the operator an audio assurance that the computer is accepting the incoming data, but you might want to use it for other functions also.

Programming Beep and Click is easy. Just write Beep or Click.

```
BEEPER     BEEP
           CLICK
           GOTO BEEPER
           STOP
```

This is a good program to use if you like to be alone. You can drive everyone screaming out of the office, as the machine will alternately Beep and Click forever, although you might be kind enough to program in a halt via the Keyboard and Display switches.

Problems:

1. Write a program that examines a person's name and age. Set the program up to see if a man named George, age 28, is using the Keyboard.
2. Add the steps necessary for the program to CLICK when the correct name and age are typed in.

```
GEORGE   INIT "GEORGE"
NAME     DIM 6
N28      FORM "28"
AGE      FORM 2
Q1       KEYIN  *V1,*H1,*EF,"NAME, PLEASE?" ,NAME
         MATCH NAME TO GEORGE
         GOTO Q1 IF NOT EQUAL
         KEYIN "WHAT IS YOUR AGE, GEORGE?", AGE
         COMPARE AGE TO N28
         GOTO FINDER IF EQUAL
         DISPLAY *V5,*H1, "YOU'RE THE WRONG GEORGE – SORRY!"
         GOTO Q1
FINDER   DISPLAY *V5,*H1, "GEORGE, YOU'LL BE OVER THE HILL IN 2 YEARS!"
         CLICK
         BEEP
         STOP
```

Chapter 8
Databus Lesson 6
Cassette Tape Operations

Having a built-in storage medium such as the cassette tapes provides the Datapoint 2200 user with the capability to generate and maintain a group of records (sometimes called a data base) within the Datapoint itself. In some situations the operations used in program generation involve simply "pulling" various programs off the rear cassette tape and using them. However, in most business applications, only one or two programs are routinely stored and used, and the remainder of the cassette tape space is used for storage of data, either for local use or for transmission via communications channels to another site.

With Databus, we can easily record and retrieve data without regard for the usual complexities of programming a mechanical storage device. The language takes care of all the intricate detail automatically.

The front tape deck, the one closest to the keyboard is deck number two, and the rear, number one. Although you know that during Databus operation, the Interpretive Tape is in the rear deck, you still may record information on it. The program has enough sense to find out where the program area of tape number one ends, and uses the remainder of the tape for storage.

Use of the tapes involves only several logical steps and adherence to two or three rules.

First, the tape should be rewound, in order to assure a common beginning place. Make sure that the clear portion of the tapetrack leader is not showing, by manually moving the tape with a pencil in the sprocket holes. It **is** possible for the tape to be rewound all the way to the wrong end, as there is clear leader at each end of the tape.

Second, after writing all your data be sure to tell the tape that this is the end of the data. An WEOF (Write End of File or Data) mark should be written.

Third, "TRAPS" should be set when reading data such that the End-of-File mark can be detected to notify the program that there's no more data.

After each WRITE instruction, the tape chosen will advance and the data will be recorded in a serial fashion, label after label. You'll need to remember the sequence of labels because they'll be needed in reading back the data. A group of recorded labels is often called a "Record", while a group of Records is known as a "File".

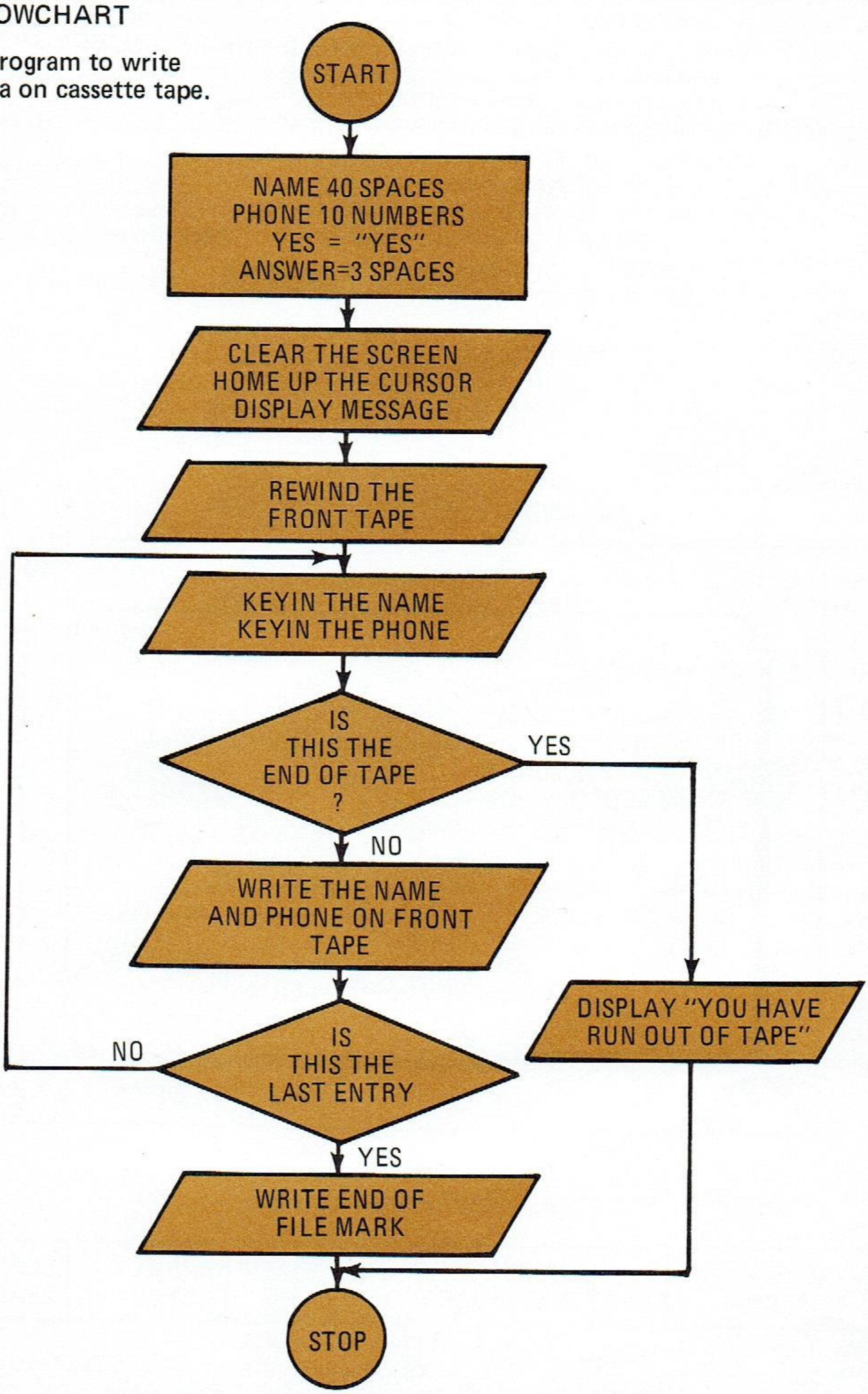
Before we do an example, try to think of these operations as comparable to recording on an audio tape data such as your name, address, age, and telephone number, and all your friends' names, addresses, and phone numbers. You would get the first name and details, press the record button, and say Harry Aardvark, 149 Maple, Sod House, Idaho. Now, release the button and look up the next name. Later, when you're looking for an address, you would listen to the tape until the sought-for name came up and take note of the information.

In effect, we do exactly the same operation with the Datapoint 2200 except that the recording method uses digital signals rather than audio signals.

Enough of the analogies – let's write a name and address program. First, a program to write the names on tape and then one to search for a particular name:

FLOWCHART

Program to write data on cassette tape.

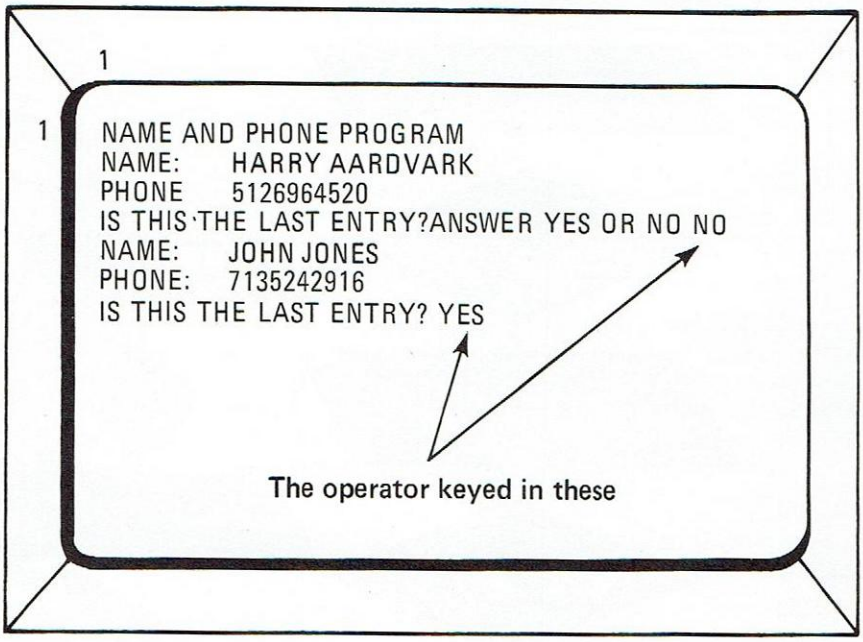


CASSETTE TAPE OPERATIONS

PROGRAM

```
NAME        DIM 40
PHONE       FORM 11
ANSWER      DIM 3
YES         INIT "YES"
            DISPLAY  *V1,*H1,*EF,"NAME AND PHONE PROGRAM"
            PREPARE 2
            TRAP ENDTAP IF EOT2
WRITER      KEYIN "NAME: ",NAME
            KEYIN "PHONE: ",PHONE
            WRITE 2,NAME,PHONE
            KEYIN "IS THIS THE LAST ENTRY?: ":
            "ANSWER YES OR NO",ANSWER
            MATCH ANSWER TO YES
            GOTO FINISH IF EQUAL
            GOTO WRITER
ENDTAP      DISPLAY "YOU'RE OUT OF TAPE"
            BKSP 2
FINISH      WEOF 2
            BEEP
            STOP
```

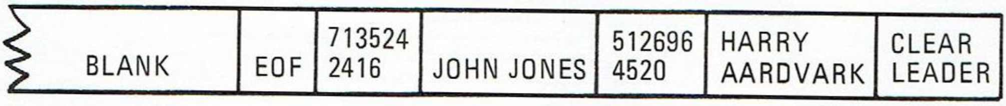
DATAPOINT 2200 DISPLAY SCREEN



```
NAME AND PHONE PROGRAM
NAME:     HARRY AARDVARK
PHONE     5126964520
IS THIS·THE LAST ENTRY?ANSWER YES OR NO NO
NAME:     JOHN JONES
PHONE:    7135242916
IS THIS THE LAST ENTRY? YES
```

The operator keyed in these

TAPE FORMAT



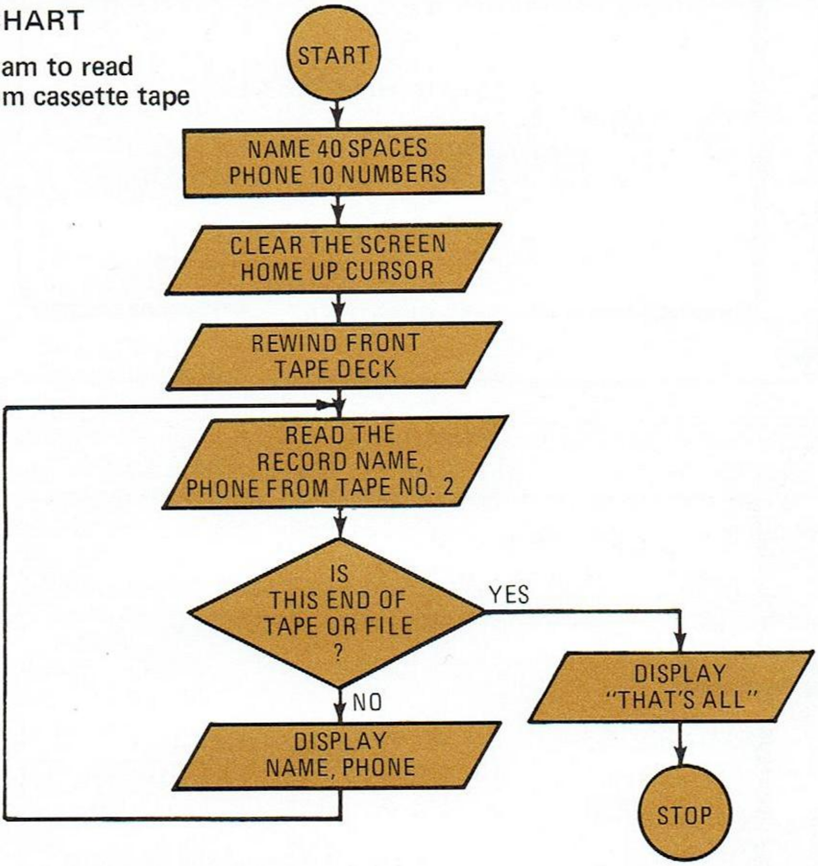
BLANK	EOF	713524 2416	JOHN JONES	512696 4520	HARRY AARDVARK	CLEAR LEADER

Notice how the TRAP instruction worked. Nothing happened until the machine found out the end of tape had arrived. Then the TRAP instruction is allowed to operate. It would take quite a Christmas card list to fill one side of a cassette. That's about 1500 names per side, and more could be stored if we made the records longer and more efficient. The instruction, BKSP 2 moves the tape backward one record length. In this program if the end of the tape was encountered, the tape was backspaced one record and an end-of-file was written.

Now, let's write a program to read the data off the tape.

FLOWCHART

Program to read data from cassette tape

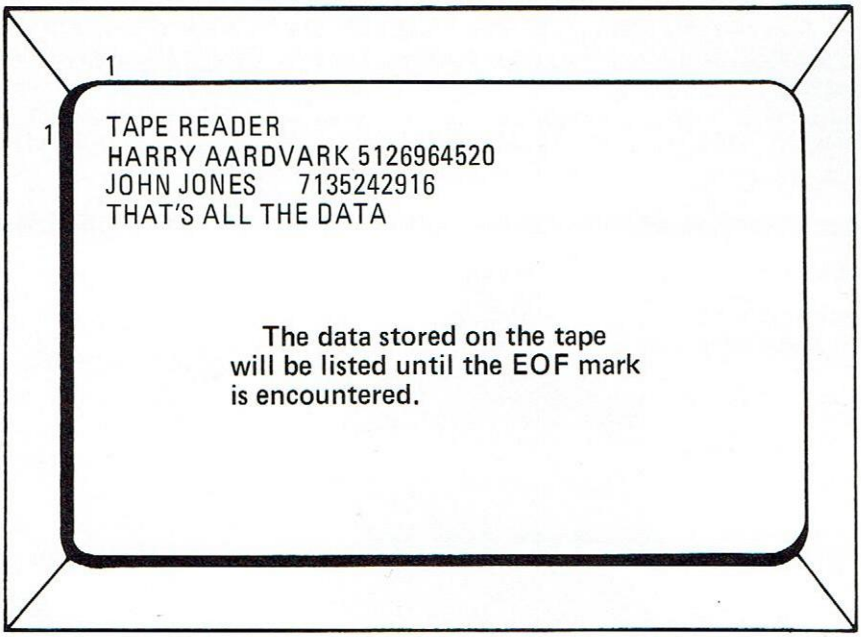


PROGRAM

```
NAME       DIM 40
PHONE      FORM 10
           DISPLAY *V1, *H1, *EF, "TAPE READER"
           REWIND 2
           TRAP DONE IF EOF2
           TRAP DONE IF EOT2
READER     READ 2, NAME, PHONE
           DISPLAY NAME, PHONE
           GOTO READER
DONE       DISPLAY "THAT'S ALL THE DATA"
           STOP
```

CASSETTE TAPE OPERATIONS

DATAPOINT 2200 DISPLAY SCREEN



```
TAPE READER
HARRY AARDVARK 5126964520
JOHN JONES     7135242916
THAT'S ALL THE DATA
```

The data stored on the tape will be listed until the EOF mark is encountered.

In WRITE and READ operations, you get back what you put in, in the same order. Always check for the End-of-File mark as the tape is read so there won't be any chance of reading old data that's lingering on the tape. That chance is remote because of the inherent error-checking powers of DATABUS, but we need to know when to finish, so don't forget to check.

If you plan to do a great deal of tape work, the DATABUS Reference manual goes into far more detail than this booklet, and we suggest you order a copy.

Chapter Summary

READ 1 or 2, Labels	Allows data to be read from previously – recorded DATABUS file.
REWIND 1 or 2	Tape unit is rewound and positioned to read.
BKSP 1 or 2	Tape is backspaced one record
PREPARE 1 or 2	Tape deck specified is rewound and prepared for subsequent writing data.
WEOF 1 or 2	This instruction writes an end-of File mark on the tape to indicate end of data.
WRITE 1 or 2, Labels	Allows data in labels to be written on tape.

Problems

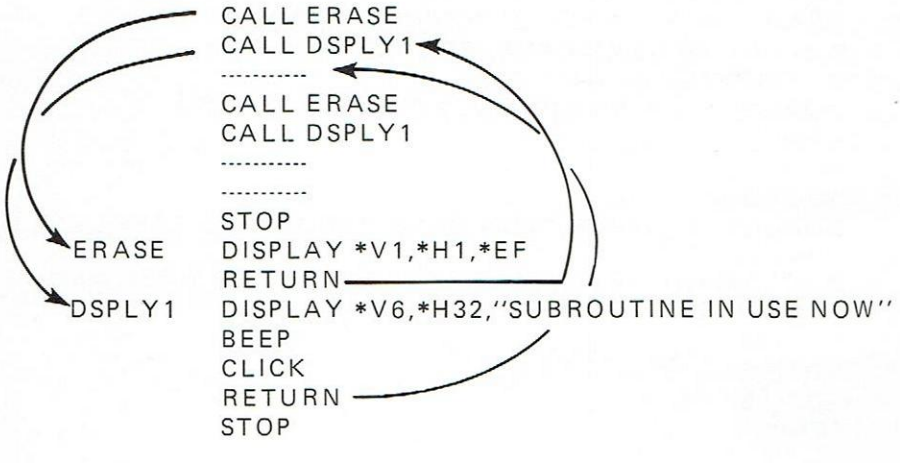
1. Combine tape examples to write data on tape and read it back in the same program.
2. In problem one, break the phone number into three areas so that the number is written as three labels, area code, exchange, and number, 512-696-4520. Modify DISPLAY instructions to include the hyphen between numbers.

```
NAME      DIM 40
AREACD    FORM 3
EXCHNG    FORM 3
NUMBER    FORM 4
NUTAPE    PREPARE 2
          TRAP ENDER IF EOT2
WRITER    KEYIN *V1, *H1, *EF, "NAME? ", NAME, *H60, "PHONE? ", AREACO,"–":
          EXCHNG,"–", NUMBER
          WRITE 2, NAME, AREACD, EXCHNG, NUMBER
          DSENSE
          GOTO PART 2 IF EQUAL
          GOTO WRITER
PART 2    WEOF 2
          DISPLAY *V1, *H1, *EF, "THE TAPE WILL NOW BE READ"
          REWIND 2
          TRAP FINISH IF EOF2
READER    READ 2, NAME, AREACD, EXCHNG, NUMBER
          DISPLAY *H1, *V1, NAME, AREACD, NUMBER
          GOTO READER
FINISH    DISPLAY*H1,*V12, "END OF DATA ON TAPE"
          STOP
ENDER     BKSP 2
          WEOF 2
          DISPLAY *H1, *V12, "YOU HAVE RUN OUT OF TAPE LAST RECORD ":
          "WAS LOST"
          DISPLAY *H1, *V12, "PRESS KEYBOARD KEY WHEN NEW TAPE IS IN ":
          " FRONT DECK"
SENSER    KSENSE
          GOTO NUTAPE IF EQUAL
          GOTO SENSER
          STOP
```

Chapter 9
Databus Lesson 7
Using Subroutines

Often a user finds that a particular sequence, or grouping, of instructions crops up quite frequently in the programs he writes. For instance, in a data entry application a user may frequently want to erase the screen, display a form, beep and related chores. To have to write out this identical sequence of instructions each time is burdensome and time consuming. Fortunately there exists a handy technique for elimination of this drudgery. This involves the labeling of these standard sequences that reappear in program writing with distinctive names and making them available upon request to the program writer. These standard sequences of instructions are known as subroutines. A user can call for a subroutine when required, have the repetitive task at hand accomplished, then get back to the mainstream of the program. To do this we make use of the CALL and RETURN instructions.

To demonstrate this useful feature of DATABUS, let's assume that you're writing a program that is quite lengthy and which requires you to erase and place various messages upon the screen. The example has no actual application but it will serve to give you the hang of the subroutine approach. In the following example, note that the CALL instruction jumps to the instruction containing the label and keeps working on that line of instructions until it encounters the RETURN instruction. At that point, the computer jumps back to the instruction immediately following the CALL instruction. The dotted lines indicate instructions that are not pertinent to the example.



```
        CALL ERASE
        CALL DSPLY1
        ----------
        CALL ERASE
        CALL DSPLY1
        ----------
        ----------
        STOP
ERASE   DISPLAY *V1,*H1,*EF
        RETURN
DSPLY1  DISPLAY *V6,*H32,"SUBROUTINE IN USE NOW"
        BEEP
        CLICK
        RETURN
        STOP
```

The arrows show the "leaping around" the computer goes through in finding and executing the subroutines. You can CALL other subroutines even though you're in one already. CALLS can be "nested" eight deep. That is, you can say CALL eight times before saying RETURN. If you nest the subroutines more than eight times, the computer will lose track of what's going on and weird things will happen.

Naturally, the advantage of this subroutine feature lies in your being able to write shorter programs and avoid writer's cramp.

Another advantage is that shorter programs occupy less space in computer memory than longer programs. (Memory economy assumes that you use the subroutine more than once or the space saving benefit won't hold true.)

LINKING UP WITH OTHER DATABUS PROGRAMS

At Datapoint's home facility, 2200's running DATABUS are used for many business data processing applications. In some cases, the programs that do such jobs as printing price sheets often occupy the entire memory space, and it is desirable to have one program fetch another as soon as it has finished. This eliminates the need for an operator to wait around and see if a part of a job has finished, so that they can proceed with the next part.

To do this operation, called "chaining," we make use of the CHAIN instruction. Look at the example below:

```
PGM2    INIT "PGM2"
        -----------
        -----------
        -----------
        -----------
        -----------
        CHAIN PGM2
        STOP
```

The last step in the program (other than STOP) instructs the computer to go out and find the program named "PGM2." If you wrote your program and used a chain instruction, you would see the rear tape move as the computer found the DATABUS program the chain instruction referred to and loaded it.

You can even carry forward information in labels as you chain from one program to another. Let's suppose that your first program asked for today's date via a KEYIN instruction and you wanted to use this date in all other programs you were going to run. Note that asterisks are used to define what to carry over for use of the next program.

```
        Program No. 1                           Program No. 2

MONTH   DIM *2                          MONTH   DIM *2
DAY     DIM *2                          DAY     DIM *2
YEAR    DIM *2                          YEAR    DIM *2
PGM2    INIT "PGM2"                             -----------
        KEYIN MONTH,DAY,YEAR                    -----------
        -----------                             -----------
        -----------                             -----------
        -----------                             -----------
        CHAIN PGM2                              -----------
        STOP                                    STOP
```

LINKING

Using the asterisks and arranging the labels in identical order in the beginning of the program you will find that Program No. 2 has the data that was asked for (KEYIN) in Program No. 1.

Be sure to observe the following rules when carrying labels from one program to another using CHAIN:

1. The program being chained to must be a DATABUS program and be cataloged under the same name on the rear tape. (See Chapter 11.)
2. The name of the program must be defined in a label using an INIT or loaded into a DIM during the first program's operation.
3. Make sure the labels are in the beginning of the program, in the same order, and are DIM'ed or INIT'ed to the same value if you want to carry label contents ahead.
4. Use the asterisk in the labels to denote that the computer is to carry this instruction along to the next program.

Chapter Summary

CALL label	Transfers operation to instruction indicated by label.
RETURN Label	Transfers operation to instruction immediately following last CALL instruction.
CHAIN Label	Locates, loads, and runs named DATABUS program.
Label DIM *n	Asterisk allows label data to be carried from one program to another.
Label INIT *n	
Label FORM *n	

Chapter 10
Databus Lesson 8
Advanced Databus Techniques

In Chapter 1 of this booklet, we noted that its contents would not cover all of the capabilities of DATABUS. We have covered the fundamental concepts in depth, but there are several additional features that merit further study and will reinforce your ability to construct a DATABUS program. The DATABUS Reference Manual, of course, contains the complete story on all the features of DATABUS.

Character Of String Operations

Recall the DIM and INIT instructions which allowed us handy storage places for mixed alphabetic and numeric data. In some applications, the programmer might wish to examine the contents of this data. For example, if an operator had entered a part number from the keyboard, the programmer might have to dissect the part number to see if the third character was an "A" or a "Z" or whatever.

This type of operation involves "string" operations. A string being a group of characters; i.e., ABZEKE123 could be a string, and "GOODFELLOW" another.

DATABUS is well-equipped for these operations. While space prohibits a detailed explanation, we will briefly outline these features in case you are asked to examine in detail the contents of a label.

Suppose a program contained a label thusly

```
NAME        DIM 40
            STOP
```

and the operator keys in "Jack Jones." The label now contains a "string" and some other things you weren't aware of before.

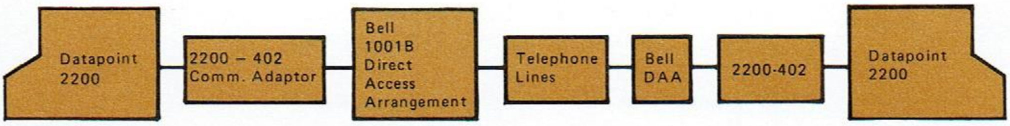
LENGTH	FORM-POINTER	JACK JONES	END-OF-TEXT

The length tells us how much data the label contains, and we can adjust the formpointer to point to any of the characters in the string. Using this technique, there are a whole group of instructions that allow examination of this data. The Reference Manual should be your guide for further study in this area.

Communications

Many Datapoint systems are installed with the end purpose of transmitting cassette data via telephone lines to another distant 2200 which will record the data upon its own cassette or perhaps on an industry- compatible magnetic tape. In conventional mini-computer systems the communications programming is usually written in binary machine language and can be very complex. To ease this chore, a version of DATABUS was written to include English-language instructions capable of transmitting and receiving data.

This version, DATABUS 3, allows a 2200 equipped with a 1200 baud modem of Datapoint manufacture to communicate over standard telephone lines.



ADVANCED DATABUS TECHNIQUES

The Cassette tape data could be generated by any of the DATABUS family of languages. We could use, for instance, our DATABUS 2 program that recorded names and phone numbers on tape to generate the data file.

We can then take a DATABUS 3 program and transmit this data to another terminal as the cassette information is compatible between the two versions of DATABUS. Why didn't we simply write the whole job in DATABUS 3 since it contained the communications instructions needed? It's possible but bear in mind that these two versions of DATABUS vary in some key features. To put all these features into one version of the language would require very large machine memory. In DATABUS 3 you will find considerably less arithmetic power than in DATABUS 2. So, if you aren't using much arithmetic, the entire job could be written in DATABUS 3.

As we've noted, considerable communications power is incorporated into DATABUS 3. We'll list some of the features and their instructions.

AUTOMATIC DIALING	NMBER1 INIT "1 *512-696-4520" DIAL NMBER1 (asterisk provides 2 second pause in dialing sequence)
AUTOMATIC ANSWERING	RING
DATA TRANSMISSION AND RECEPTION	SEND (labels), RECEIVE (labels)
AUTOMATIC ERROR CONTROL AND RETRANSMISSION	PARITY
DUAL CODE CAPABILITY	ASCII, EBCDIC

If you plan on a specific communications application with your Datapoint System, send for a copy of the Databus Reference Manual and Communications Reference Manual.

SUMMARY OF THE CASSETTE TAPE OPERATING SYSTEM DATABUS LANGUAGES

DATABUS 1 High Level Language	This powerful high-level language provides full arithmetic, file handling, keyboard input, screen display formatting, and printer formatting capability.
DATABUS 2 Expanded String Commands	This version of Databus provides all the features of Databus 1 plus full character string handling capabilities. Although user space is slightly smaller than Databus 1, this version has a most versatile command structure.
DATABUS 3 Communications Oriented	Known as the communications-oriented version, Databus 3 contains commands to operate the 2200-402 (1200 baud modem) enabling transmission and reception of cassette or other data. This version has all the capabilities of Databus 2 except full arithmetic. Databus provides in addition, capability to use two 9 channel magnetic tape units. Programs using other versions of Databus may load data on cassettes for transmission by Databus 3, as all Databus programs are cassette-format compatible.

DATABUS 4

Small Memory Requirement

Designed for Datapoints with less than 8K of memory, this version will run in a minimum of 4K memory and up. An 8K machine is required to compile the interpretive code and this may be then loaded into a 4K or larger machine.
Databus 4 provides file handling, keyboard, CRT display, and cassette tape I/O as well as limited string and numeric operations.

DATABUS 6

Keypunch Replacement

Often called the "super keypunch emulator," Databus 6 programs will operate in a 2K Datapoint and offers a highly sophisticated keypunch replacement.
The programs provide high-level operations of punching, editing, verifying and transmitting data, via a communication channel. 6 control cards may be used.
The system is programmed via "control cards" by the operator and requires no compiling.

The Disk-based language, DOS (Disk operator System) Databus

A disk memory added to the Datapoint 2200 increases the computer's capability until it becomes hard to differentiate between the throughput of the Datapoint compared to much higher priced conventional business computers.

The disk adds a large amount of high-speed memory to the system. This memory can be used for storage of data and programs. The DOS (Disk-operating System) DATABUS language allows the user to take full advantage of the disk's power without complex programming. The DOS Databus language also provides a completely dynamic and open-ended file storage capability. A user can create, expand, delete or move the disk files without regard to space or linking problems.

DATASHARE (Multi-user Databus)

To expand the capabilities of the DOS Databus, a version is available to allow one Datapoint 2200 (Version II) and a Disk to service up to eight users, each with a display terminal, such as a Datapoint 3300 or Datapoint 3360, or a teletype. This version of Databus effectively expands the power of a 2200 eight times, giving each user the power of his own 2200 and Disk.

With this system, each user can run his own Databus program and access private or public data files on the disk. Two users may simultaneously use the same file.

A typical business application would involve one user running a payroll, another updating an inventory and accounts receivable or payable being run. Each application program may use the entire 16K or memory of the Datapoint.

The eight terminals may be local or remote and have attached terminal printers. A high-speed system printer is also available at the Datapoint 2200.

Chapter 11

Getting a Datapoint 2200 To Run The Program

If you've gotten this far, you probably have a program scratched out on the back of an envelope that you would like to try, or perhaps you might first try one of the examples in the booklet.

In any case, arm yourself with the following items:

1. Datapoint 2200 Version I or II with at least 8K of memory.
2. Some kind of Datapoint printer (not absolutely essential, but very handy).
3. A complete set of DATABUS 2 tapes. (At the time of this writing, tape numbers 274 and 276 are the current ones. Your local sales office will know if there are more recent releases.) There are seven versions of DATABUS – you need DATABUS 2.
   C00274 – DATABUS Program Generation System
      a. DBEDIT – DATABUS Editor
      b. DB2CMP – DATABUS 2 Compiler
      c. DB2CC – DATABUS 2 Compiler Configurator
   C00276 – DATABUS 2 Interpretive System
      a. DB2INT – DATABUS 2 Interpreter
      b. DB21C – DATABUS 2 Interpreter Configurator
      c. MASTER – the Cataloger
4. At least one blank tape (called, in the trade, a scratch tape).

The sequence of events used to take the program from a concept to an actual running computer program is fairly involved. Don't let the number of the programs listed above scare you off, though. Read and follow the directions carefully – success will be yours. At your next cocktail party, you can look coolly at a Director of Data Processing in attendance and modestly say, "Sure, I've written a little software."

Before fame and glory are yours, however, we have to get the program in the machine and running. The following table lists the sequence of events.

1. Type in the program and load it on the front cassette tape. (Editor)
2. Convert your program into computer code. (Compiler)
3. Catalog your program on Tape No. 276. (Interpretive System)
4. Run it. (Good Luck)
5. Doesn't run the way you would like? Fix it. (Editor) Back to Step No. 2.

Step 1
Databus Editor Operations

Sit in front of the 2200. Relax. Observe that there are two cassette tape decks on top of the machine. Pull the black handles toward you and the cassette tape carriers jump up. Get the feel of loading and unloading the cassettes. Insert the cassette back end first – that is, with the side exposing a small section of tape toward you. Push the carrier down and slide the black handle to the rear. You should feel and hear a nice, solid clunk as the cassette goes in place.

You can use both sides of a cassette, so mark which side you're going to use on the blank cassette so you won't later put it in upside down. The DATABUS tapes have stickers indicating which tape it is. These stickers go face up into the 2200.

Take your practice tape out and turn the machine on. The switch has been hidden from view along the right-hand lower edge.

Find the DATABUS 2 Program Generator Tape (the one with DBEDIT on it) and load it in the rear deck. Press the RESTART key – after the tape stops clicking and moving, the screen should come up with something like this:

```
READY                    CTOS 3.1
```

Type in CAT (short for CATALOG) and press the ENTER key. All CTOS (Cassette Tape Operating System) operations are initiated when the Enter key is pressed. This mark means hit the Enter key.↵

Now you will see:

```
CAT↵
DB2CC DBEDIT   DB2CMP
READY
```

This now assures us that the programs we're going to need are on tape.

The program we are interested in at this point is the Editor, DBEDIT, and we will ask the computer to find this program and run it by typing the following:

```
RUN  DBEDIT↵
```

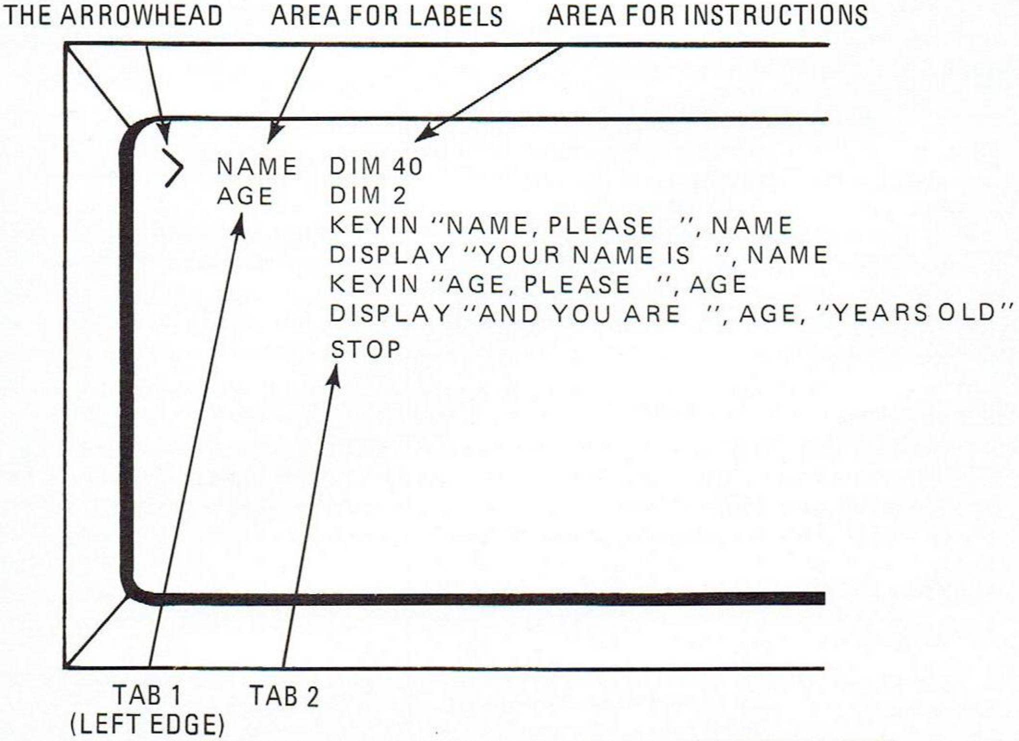
The back tape will move to find this program and load it into memory. After loading, a message indicating that the DBEDIT program is running will appear and ask you if you want to edit a new or old tape or if you want to duplicate a tape. Note that all instructions to the DBEDIT program are prefaced with a colon (:).

Place the blank tape in the front deck and close the carrier. Now, type:NEW↵ This tells the Editor program that the tape you placed in the front deck can be considered blank. Later on,if you have a program in the front deck you want to save and you type:NEW↵you've lost it. That's what :OLD↵is for, but we'll get to that later.

After the back tape has stopped moving, the screen will clear, and you will see the blinking cursor waiting for your next more.

Your move, of course, is to get your program onto the front tape. Let's get the feel of the DATABUS Editor first. By the way, the Editor program has nothing to do with green eye shades and cub reporters. Editor, in the computer business, is a program allowing text entry (your program) and the later modification of the text, in case you blow it.

To make life easier, the DBEDIT program has preset tabs for labels and instructions. Tap the space bar and note that the cursor jumps right about 10 spaces. Most instructions don't have labels, so this provides a convenient means of skipping that area. The drawing below shows what your program should look like as you type it in. Be sure to use only upper case (Caps) letters for the instructions.



```
>  NAME   DIM 40
   AGE    DIM 2
          KEYIN "NAME, PLEASE  ", NAME
          DISPLAY "YOUR NAME IS  ", NAME
          KEYIN "AGE, PLEASE  ", AGE
          DISPLAY "AND YOU ARE  ", AGE, "YEARS OLD"
          STOP
```

REMEMBER TO TAP THE ENTER KEY AFTER EACH LINE!

COMPILING THE PROGRAM

After completing a line the Enter key is tapped and the next line begun.

What if you make a mistake? If you make it while typing, the Backspace key will rub out the offending character and the Cancel key will obliterate the entire line.

Now that you've finished typing in the program, let's suppose that several errors mysteriously escaped your attention. This will give us a chance to show some of the power of the Editor program.

Notice the small arrowhead (>) in the left-most column. Press the Keyboard and Display keys one at a time and watch the arrowhead move up and down. By fiddling with these two keys, you can point out any line of code on the screen. So fiddle around and point to the line with the error.

Now, with the cursor in the left-most position at the bottom of the screen, you can type in the Editor commands to get at the error. All Editor commands are prefaced with a colon. Unless you do this colon thing, the Editor will assume it's just another line of the program.

:DEL (Delete) Blots out the entire indicated line and lets you try again.

:INS (Insert) Opens up a space between two lines so that another line may be squeezed in.

:MOD (Modify) This command lets you change individual characters or a group of characters. Suppose in the line you accidentally typed DSPLAY and didn't notice it. Point the arrowhead at the line and type :MOD DSPLAY <DISPLAY. The line will now contain the proper word.

```
DSPLAY < DISPLAY
   ↑    ↑   ↑
 (OLD)  |(NEW)
  (REPLACES)
```

All this will be somewhat hazy until you have some experience hammering away at the keyboard. One last step – when you've got everything so it looks good, stop. Contemplate the divine mysteries. And then type the last and most important Editor command.

:END (End of Program) Indicates to the Editor that the programmer is done and to write a complete, perfect copy of his program to the front tape.

Write this instruction all over your notes and ask the secretary to remind you to type :END. If you don't, and skip to the next step, the 2200 will laughingly toss your program to the electronic winds, and you'll have to painstakingly type in the program again.

After :END has been typed in, the tapes will move and after some time CTOS 3.1 READY message will reappear. Don't get anxious – wait until the message comes back.

Step 2 – Compiling The Program

When the CTOS 3.1 READY (or whatever number of CTOS you have) comes up, type in RUN DB2CC). This program, the DATABUS 2 Compiler Configurator, enables you to specify what type of printer, if any, is attached and how much memory is in the 2200. You only have to do this once unless you change the memory or printer.

Object Machine size: 8, 12, 16 K -type in 8 if you aren't sure.

Compiler machine size: 8, 12, 16 K - type in 8 if you aren't sure.

Printer – Remote or Local:

a. 2200/Printer is Local
b. 2200/Line Printer is Local
c. 3300/Thermal Printer is Remote, 300 Baud speed
d. Teletypewriter is Remote, 110 Baud speed

e. Anything hooked up through a Communications Adaptor is Remote. Take characters-per second-speed, multiply that times 10, and you have the Baud rate or speed in bits/second, in most cases.

f. If you don't have a printer, type L anyway. Remember in later operations not to ask for a print-out, or the machine will halt!

Once this is done, the tapes will move again and CTOS 3.1 will come back.

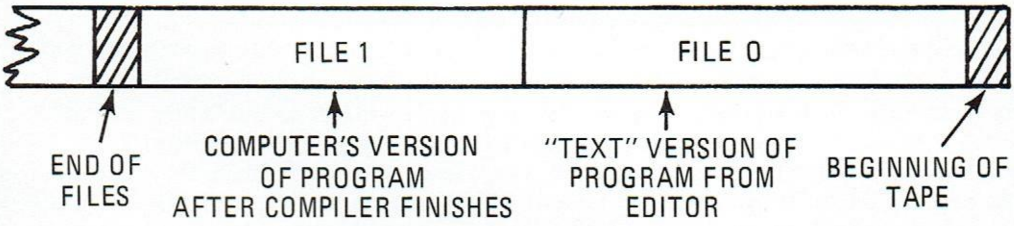
Now we'll convert your program from text to computer code, or, as programmers would say, compile it. The compiling operation adds a computer-readable version of the text on the front deck. In effect, there will now be two files on the front tape after compiler operation

The text version you produced with the DBEDIT (Editor) program and the computer's version of that text (object code) produced by the compiler.

Now we'll convert your program from text to computer code, or, as programmers would say, compile it. Type RUN DB2CMP and more questions will appear.

```
DISPLAY?:    YES
PRINT?:      YES (Only if you have a printer attached)
CODE?:       NO
```

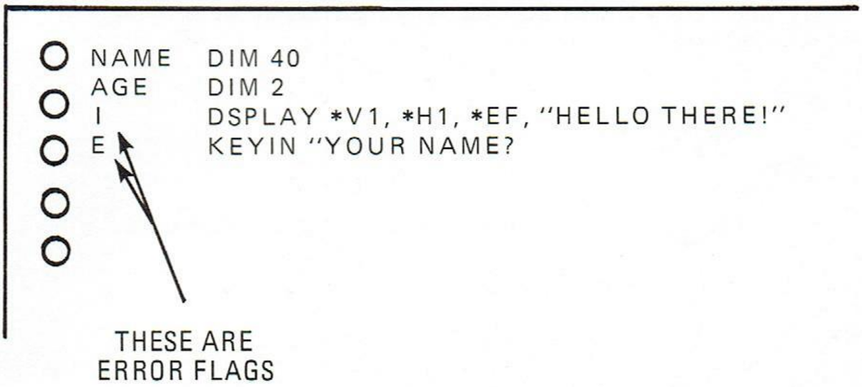
Now stand back and watch as your program is compiled. Compiling adds another shorter section to the tape. The tape will look something like the drawing below after the operation is complete.


FILE 1
FILE 0
END OF FILES
COMPUTER'S VERSION OF PROGRAM AFTER COMPILER FINISHES
"TEXT" VERSION OF PROGRAM FROM EDITOR
BEGINNING OF TAPE

While the compiler works, it displays and prints-out your program. At this time, it may also point a disapproving finger at errors by adding small "Flags" to the extreme left of the screen or printer.

Here's an example of typical programming error:

ERROR FLAG LOCATIONS VIA SCREEN OR PRINT-OUT



```
NAME   DIM 40
AGE    DIM 2
I      DSPLAY *V1, *H1, *EF, "HELLO THERE!"
E      KEYIN "YOUR NAME?
```

CATALOGING THE PROGRAM

DISPLAY IS SPELLED WRONG AND KEYIN lacks the last set of quotation marks. The complete list of errors is below:

1. D: The D flag means DOUBLE DEFINITION. It is flagged if a label or variable has been defined to more than one value during compilation. In that case, it has the first value.
2. I: The I flag means INSTRUCTION MNEMONIC UNKNOWN. The instruction was not an acceptable instruction.
3. E: The E flag means that an error has occurred in the operand field of a statement or some unrecognizable character appeared in the wrong place.
4. U: The U flag means UNDEFINED SYMBOL. It is used whenever a symbol is referenced and is not defined.

If your program has error flags, the faults must be corrected before going any further. Go back to Step 1 and run Editor. Be sure to say :OLD when it asks, or the text program on the front tape (your program) will be erased. Once the tapes have stopped moving, press the Keyboard and Display keys simultaneously to see the contents of the tape (your program) roll by on the screen.

Step 3 – Cataloging The Program

No errors? Good. Now remove the Program Generator Tape and insert the DATABUS Interpretive tape. Hit the Restart key and wait for CTOS 3.1 READY to come up. Type in RUN DB21C↵ the Interpreter Configurator, and fill in with the questions just like you did with the Compiler Configurator.

After that's done, CTOS 3.1 will come back. The Compiler version made a copy of our program that can be transferred from the front tape to the back tape and placed in the catalog. Up to 14 programs may be cataloged on the back tape.

Dream up a name for your program following the same rules for labels. RUNNER might be a good name for an example. Now type IN RUNNER↵and the tapes will do quite a bit of moving back and forth. Be patient. CTOS 3.1 will reappear when the cataloging finishes.

Let's review that sequence again. Before we transfer the program we can find out the current contents of the tape by asking for a catalog:

```
CAT↵
DB21C  DB21NT  MASTER
READY
```

Now, by using the IN command we can load in our program

```
IN RUNNER↵
READY
```

Now the catalog will show our program

```
CAT↵
DB21C  DB2INT  MASTER RUNNER
```

There's proof it's now in the catalog on the back tape deck.

Step 4 — Running The Program

Remove your tape from the front deck. If the program you wrote calls for writing on the front deck, find another blank tape and load it in. You wouldn't want to overwrite your program tape.

CTOS should still be on the screen and the cursor blinking at you. Type RUN DB2INT and finish your coffee while the 2200 runs out on the tape and finds the interpreter. The message:

```
DATABUS INTERPRETER 2.1

 Program Name:  RUNNER
```

You type this in where it's indicated

will show. Type in your program's name, in this case, RUNNER, and a few seconds later the results of your labors will be known. Your program will run. When it encounters the stop instruction the DB2INT program will reload. Congratulations.

Step 5 — Fixing The Program

Most programs require some degree of tinkering with before the user becomes completely satisfied with the screen format, etc.

Replace the blank tape in the front deck with the tape with your program on it and begin at step 1. Be sure to type: OLD when the Editor comes up.

Depressing the Keyboard and Display keys simultaneously will display the contents of your program tape, and you can watch as it scrolls by on the screen. Lifting the keys will cause the search to stop and the portion of the program displayed on the screen can be modified, or deleted as desired.

General Hints

1. Try an example program in the book to get the hang of all steps.
2. Make sure the printer is turned on if you're going to use it.
3. If you have an unexplanable problem, give your Datapoint Systems Engineer or salesman a call and they'll try to advise.

Appendix
Datapoint Business Language 2 (Databus 2) Definitions

condition — The result of an arithmetic operation: OVERFLOW, LESS, EQUAL ZERO, EOS.

character string — Any string of alphanumeric characters.

event — The occurence of end-of-file, end-of-tape, a data type error, or a program chain failure: EOF1, EOF2, EOTI, EOT1, FORM1, FORM2, CFAIL.

list — A list of variables or controls appearing in an input/output type of instruction.

label — A name assigned to a statement.

nvar — A label assigned to a directive defining a numeric string variable.

svar — A label assigned to a directive defining a character string variable.

sval — A label assigned to a directive defining a character string variable, or a quoted alphanumeric character, or a number.
The number may be octal or decimal as long as it is between 0 and 12710.

unit — A number defining a tape deck. 1 (defining the rear deck), 1 (defining the front deck).

456.23 — Refers to any decimal number.

See DATABUS reference manual for further information

DATABUS 2 INSTRUCTIONS

TRAP	(label)	IF	(event)
GOTO	(label)		
GOTO	(label)	IF	(condition)
GOTO	(label)	IF NOT	(condition)
CALL	(label)		
CALL	(label)	IF NOT	(condition)
CALL	(label)	IF NOT	(condition)
RETURN			
RETURN	IF	(condition)	
RETURN	IF NOT	(condition)	
STOP			
STOP	IF	(condition)	
STOP	IF NOT	(condition)	
CHAIN	(svar)		
BRANCH	(nvar)	OF	(label list)

STRING

CMATCH	(sval)	TO	(sval)		
CMOVE	(sval)	TO	(svar)		
MATCH	(svar)	TO	(svar)		
MOVE	(svar)	TO	(svar)		
MOVE	(svar)	TO	(svar)		
MOVE	(nvar)	TO	(svar)		
APPEND	(svar)	TO	(svar)		
RESET	(svar)	TO	(svar)		
RESET	(svar)				
BUMP	(svar)				
ENDSET	(svar)				
TYPE	(svar)				
CHAIN	(svar)				
EXTEND	(svar)				
CLEAR	(svar)				
LOAD	(svar)	FROM	(nvar)	OF	(svar list)
STORE	(svar)	INTO	(nvar)	OF	(svar list)

ARITHMETIC

ADD	(sval)	TO	(sval)
SUB	(sval)	FROM	(sval)
MULT	(sval)	BY	(sval)
DIV	(sval)	INTO	(sval)

Datapoint

DATAPOINT CORPORATION
9725 Datapoint Drive
San Antonio, Texas 78284
(512) 696-4520
Sales Offices in Principal Cities

0374-5

PRINTED IN U.S.A.