GEORGIA INSTITUTE OF TECHNOLOGY
Engineering Experiment Station

PROJECT INITIATION

Date: Dec. 13, 1974

Project Title: Computer Instruction Development

Project No.: A-1699

Project Director: Mr. R. G. Pearl

Sponsor: USDA - Forest Service; Washington, D. C., 20250

Effective: 11-15-74 Estimated to run until 2-16-75

Contract Amendment: Contr. # 11-778 Amount: $ 15,000

Reports Required: Instructional Materials - no conventional reports required.

Sponsor Contact Person(s):

Mr. Sidney Haggard
Contracting Officer's Representative (COR)
USDA - Forest Service
1621 N. Kent St.; Rm. 101-RPE
Arlington, Virginia 22209
Phone: (703) 235-8670

Signed to RADAR Division

RES TO:

Project Director   Director   Assistant Director   GTRI   Division Chief(s)   Branch Heads   Service Groups   Patent Coordinator

- Photographic Laboratory
- Security, Property, Reports Coordinator
- EES Accounting
- EES Supply Services
- Library
- Office of Computing Services
- Project File
- Other

Sue Corbin
Bonnie Wettlaufer
GEORGIA INSTITUTE OF TECHNOLOGY
ENGINEERING EXPERIMENT STATION

PROJECT TERMINATION

Date: April 21, 1975

Project Title: Computer Instruction Development

Project No.: A-1099

Project Director: Mr. R. G. Pearl

Sponsor: USDA - Forest Service; Washington, D. C. 20250

Effective Termination Date: 3/19/75 (Contract Expiration)

Clearance of Accounting Charges: ASAP

Grant/Contract Closeout Actions Remaining: Final Invoice & Closing Documents as soon as all charges clear.

Assigned to: Radar Division

COPIES TO:
Project Director
Director, EES
Director, ORA/GTRI
Assistant Director
Division Chief
EES Accounting
Patent Coordinator

EES Supply Services
Photographic Laboratory
Security--Reports--Property Office
General Office Services
Library, Technical Reports Section
Office of Computing Services
Project File
Other: 
UNITED STATES FOREST SERVICE
COMPUTER USAGE SERIES

STUDY GUIDE

PREPARED FOR THE

COMPUTER SCIENCE GROUP
U. S. FOREST SERVICE
DEPARTMENT OF AGRICULTURE

MARCH 1975
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. INTRODUCTION</td>
<td>2</td>
</tr>
<tr>
<td>II. REFERENCES</td>
<td>4</td>
</tr>
<tr>
<td>III. TAPE 1: Purpose Outline Script and Examples</td>
<td>6</td>
</tr>
<tr>
<td>IV. TAPE 2: Purpose Outline Script and Examples</td>
<td>21</td>
</tr>
<tr>
<td>V. TAPE 3: Purpose Outline Script and Examples</td>
<td>35</td>
</tr>
<tr>
<td>VI. TAPE 4: Purpose Outline Script and Examples</td>
<td>49</td>
</tr>
</tbody>
</table>
INTRODUCTION

A series of video tapes has been produced for the Department of Agriculture Office of Automated Data Systems by the Radar Division of the Engineering Experiment Station at Georgia Institute of Technology. The purpose of these tapes is to demonstrate for potential users of the Ft. Collins Computer Center facilities the demand mode of operation on the UNIVAC 1108. The demonstrations cover those features of user-system interaction which are necessary for accomplishing useful work. No attempt has been made to teach programming.

Demand mode operation is characterized by alternation of commands from user to computer and responses from computer to user on an interactive basis. The user is generally in communication with the computer from a remote terminal by telephone line hookup.

The Ft. Collins Computer Center maintains system software which provides a powerful interface between the user and the machine. The U-1100 EXECUTIVE system has an extensive command set; compilers are available for several high-level programming languages—including FORTRAN, COBOL and BASIC; and a versatile TEXT EDITOR can be called on to aid in program and data file development. In addition, the demand user can choose to operate using the CONVERSATIONAL TIME SHARING system (CTS). Under CTS he has access to a simplified interactive command set, a conversational EDITOR, and a "desk calculator" mode of operation. In addition, CTS can be treated as an interpretive programming language and, as such, can be used to write and execute CTS subroutines.

This booklet is provided to aid viewers of these videotapes in understanding and recalling specific sections of the demonstrations. The tape
series consists of four half-hour lectures:

1. Demand mode operation under EXEC
2. TEXT EDITOR usage under EXEC
3. Introduction to CTS
4. Advanced CTS

There are four sections in this booklet—with one section devoted to each lecture. Each section contains a brief introduction, an outline of the session, a script (with lecture examples included), and a list of publications containing additional related information.
References

I. Two manuals have been prepared by Systems Engineering Branch, United States Department of Agriculture, Ft. Collins Computer Center, 3825 East Mulberry Street, Ft. Collins, Colorado 80521. Their titles and a brief list of their contents are:

1. UNIVAC 1108 Demand User's Manual
   1.1 Demand Processing
   1.2 Demand Symbionts
   1.3 ED Processor

2. FCCC User's Manual
   2.1 Center Contact List
   2.2 Equipment Configuration and Specifications
   2.3 Software Systems
   2.4 Center Services
   2.5 Customer Liaison
   2.6 Establishing Accounts
   2.7 Billing and Utilization Reporting
   2.8 Rates
   2.9 Information and Document Distribution
   2.10 Supplies
   2.11 Training

Copies of these manuals and others prepared by various regional offices may be obtained by calling

1. FCCC Customer Liaison Branch,
   phone: 303-484-2266 or

2. FOREST SERVICE Liaison Office,
   phone: 303-484-2246.
II. Sperry UNIVAC publications:

1. PRM: Programmers Reference Manual for the UNIVAC 1108,
   EXEC 8 Executive System, UP-4144
2. UNIVAC 1100 Series, EXEC 8 Hardware/Software Reference, UP-7824
3. UNIVAC 1108, EXEC 8 Reference Card
4. SPERRY UNIVAC 1100 Series Introduction to Time Sharing for
   CTS users
5. CTS, Conversational Time Sharing (UP-7940)
6. SPERRY UNIVAC, 1100 Series TIME SHARING User Guide (Preliminary
   issue, RSS-033)
7. Conversational Time Sharing (CTS) System Summary, UP-7946
TAPE 1: DEMAND MODE OPERATION UNDER EXEC 8

PURPOSE:

TAPE 1 provides the viewer with a brief introduction to the facilities and services of the Ft. Collins Computer Center operated by the Computer Sciences Group of the Department of Agriculture. Most of the half-hour session consists of a demonstration and lecture covering demand terminal operation from dial up to disconnect. A subset of the EXEC commands is included in the demonstration which permits the user to:

1. Establish an active run on the U-1108.
2. Execute a prestored program using either keyboard data entry or entry of data from files.
3. Establish program files.
4. Correct and add to text in file elements.
5. Produce an executable machine language program by compiling and collecting program elements.
6. Terminate the computer run and disconnect the telephone link.
TAPE 1: DEMAND MODE OPERATION UNDER EXEC 8

OUTLINE:

I. Introduction to Ft. Collins Computer Center
   1. Operation
   2. Facilities

II. Accessing the FCCC U-1108
   1. Establishing telephone connection
      1.1 Dialing and response
      1.2 MODEM
      1.3 Terminal switch settings
   2. Site ID
   3. @RUN command
      3.1 Coding the command
      3.2 Transmitting the command by the carriage return

III. Executing a prestored program
   1. @XQT command
   2. Data input
      2.1 Direct keyboard entry
      2.2 Stored data entry
         2.2.1 @ED,I command
         2.2.2 @PRT command
         2.2.3 @ADD command

IV. Preparing a new program under Exec
   1. Establishing a program file
      1.1 @CAT command
      1.2 @TTY command
1.2.1 Character delete

1.2.2 Line delete

2. System file utility routines (FURPUR)

2.1 Aids for manipulation of files and elements

2.2 @COPY,S OLD FILE.MAIN, MYFILE.MAIN

2.3 @PRT,T command

3. Compiling a routine

3.1 @FOR MYFILE.MAIN

3.2 Symbolic and relocatable elements

4. Initial insertion of new symbolic element in file

4.1 @ED,I MYFILE.SUB

4.2 Input of source code into element

5. Compilation, correction, recompilation of new element

5.1 @FOR,S MYFILE.SUB

5.2 @ED,U MYFILE,SUB

5.3 @FOR MYFILE.SUB

6. Collection of relocatables

6.1 Complete machine language executable program

6.2 @MAP ,MYFILE.PROG

6.3 IN MYFILE.

6.4 END

V. Execution of the new program

1. @XQT MYFILE.PROG

2. Data entry

VI. Terminating the active run

1. @FIN command
2. Disconnecting the line

   2.1 @TERM
   2.2 CNTRL/D

VII. Remarks

1. Additional U-1108 features treated in videotape series

   1.1 TEXT EDITOR
   1.2 CTS

2. Reference manuals

3. Where to call for additional help

   3.1 Regional office
   3.2 FCCC liaison group
You are watching the Central Processing Unit of a UNIVAC 1108 computer. This computer is operated by the Ft. Collins Computer Center in Ft. Collins, Colorado and is used by the United States Forest Service personnel from all over the country. Today you can learn how to use it. The Ft. Collins Computer Center is the principal site used by the US Forest Service and is operated by the Department of Agriculture, Office of Automated Data Systems.

Inside this building is the UNIVAC 1108, the computer you will be using. The most important part of this computer is the Central Processing Unit or CPU. This is where all computations and decisions take place. The UNIVAC has magnetic tape for long term storage and it has magnetic discs to provide rapid access to large amounts of data. Operators monitor the progress of the machine during its work, typing in instructions whenever necessary. If special problems arise, those using the machine can talk directly to the console operator although this is not often necessary.

In the past, computer users have had to bring their jobs to the computer location and return later to pick up their print out. Now you can talk directly to the UNIVAC by telephone. This mode of operation is called the Demand Mode. To use the Demand Mode, first dial the UNIVAC 1108 at Ft. Collins. The tone indicates when the computer has responded. When you hear it, place the hand set firmly into the special cradle to start the transmission. This device is called a MODEM. A MODEM can be connected to a variety of devices for different jobs.

Some devices, such as the remote line printers and card readers, are used in what is called the batch mode. In this mode, punched cards are processed by a card reader, and the answers are printed back later on a line printer. But today we will be working with devices that interact with the computer in the Demand Mode. The user and the computer can interact with each other. The Forest Service has several different kinds of computer terminals. This is one of them. As you can see, this terminal looks very much like a typewriter. In fact most of the keys are the same as those on a typewriter. There are some additional keys which we will use later. On any terminal it's important that the switches be set in the proper position. The switch settings for the terminal you will be using can be found in your instruction manual. For this particular computer terminal, these are the switches which you must set.

This terminal is connected to the MODEM we saw a moment ago and is ready to begin communicating with the UNIVAC. First assume that we have a particular problem to solve. The problem we will be using is to find the area inside a triangle. Lucky for us the computer program to solve this problem is already
stored on the computer. All we have to do is to find it and then run it to get our answer. Let's start by typing something into the computer and see what happens.

HELLO

Well, apparently we are not doing it right. What we need to do is refer to the study guide that comes with this tape and learn that we need to start by typing a SITE ID. Let's type the SITE ID and see what happens.

HELLOUVX945

We are waiting for the UNIVAC to recognize our SITE ID. It has done so and now it's asking us to enter a RUN card. We will type the RUN card, including all the accounting information necessary for the UNIVAC to recognize us, set up our active RUN, and later give us our bill.

@RUN TECH,115144662,PEARL-RON,5,100

Our original purpose was to solve the problem of finding the area of a triangle. Now we have given all the information that the computer needs; and we can begin to execute the program to solve the problem. Every program has to have a name. The program we will be executing is called PROG. We will start with the @XQT card.

EDITORIAL NOTE: It is better practice to first assign the desired file to the user's RUN by:

@ASG,AZ RON*OLDFILE.

When the U-1108 responds with READY, then proceed to execute the program.

We will type the qualifier and file name which identify the storage area occupied by PROG:

> @XQT RON*OLDFILE.PROG
DATA IGNORED - IN CONTROL MODE
>

This is what happens when the computer doesn't quite get the message right. Let's try it again:

> @XQT RON*OLDFILE.PROG
HI...MY NAME IS PROG...RUN
ENTER LENGTHS OF SIDES
>

A = 1.00
B = 1.00
C = 1.41

AREA = .50
FROG is now running and we can enter the lengths of sides of the triangle whose area we would like to find. For this program, it's important that we use a decimal point after each number, and that the numbers be separated by commas. We enter a carriage return to tell the program we have completed the input. And the program comes back with our original input as well as the answer.

```
> 1.1,1.1414
A = 1.00 UNITS
B = 1.00 UNITS
C = 1.41 UNITS
AREA = .50 SQUARE UNITS
```

Let's type in another set of data.

```
> 3.1,4.1.5.
A = 3.00 UNITS
B = 4.00 UNITS
C = 5.00 UNITS
AREA = 6.00 SQUARE UNITS
```

and again the program gives the answer.

Now suppose that we wanted to store a large amount of data describing triangles and run this program later with the stored data. This can be done using a processor called ED which is short for EDITOR. We will discuss EDITOR in detail in another session; but for now it's good to know some of its basic features. We will type ED. We are going to insert a new data element so we'll use an I option.

```
> @ED, I DATA
```

EDITORIAL NOTE: Since no qualifier*filename was supplied on the ED command, the element DATA is inserted into file PEARL-ron*TPFS. A similar temporary work file is set up by the EXEC for every active demand run.

Our FORTRAN program has terminated; and ED has asked us to input our first line of characters. Let's input several lines using the same data as before.
We're done with EDITOR now. We'll enter an EOF (end of file) to tell EDITOR that we have finished with this portion of the work.

> @EOF
LINES: 3 FIELDATA
>
We'll now use another @XQT command to execute our program again. When the program has started it will send the same message as before. The difference will be that in running the program this time, instead of typing in data from the keyboard, we will add data using an @ADD card.

> @XQT RON*OLDFILE.PROG

Apparently the program is not going to work. Let's try the @XQT card one more time.

@XQT RON*OLDFILE.PROG
HI...MY NAME IS PROG...RUN
ENTER LENGTHS OF SIDES
NORMAL EXIT. EXECUTION TIME: etc.
> HI...MY NAME IS PROG...RUN
ENTER LENGTHS OF SIDES
>
This time it worked; and we're ready to add to our data with our ADD card. Hmm, it not only worked, it terminated itself and then ran again. This is what happens when you try to jump the gun with the UNIVAC. It doesn't forget what you've said. It just doesn't pay attention right away. So what we have done is executed the program, terminated it with the new control card and then executed it again. In any event, now we are ready to begin adding the data. This is done with an ADD card and the word DATA. The program is now reading the input which we have just created and is typing its answers.
You have now seen all that is required to run an existing program on the UNIVAC. But you may be interested in being able to enter, store and execute your own programs to solve your particular problems. In order to do this we should know a little more about some of the system programs available under EXEC 8. The first program is the EXEC. The EXEC is the master program and it uses its commands to control other programs. Let's go through the example of writing a program to compute the area of the triangle. First we need a place to store our program. What we must do is ask the EXEC to give us some space (called a file) on the disc. We can call this file anything we want. So let's call it MYFILE. This is done with an @CAT card. We may first enter a qualifier.

```
> @CAT RON*MYFILE.
READY
```

EDITORIAL NOTE: This is the simplest form of the @CAT command. It has the disadvantage that a file so catalogued is temporary. To specify a permanent file, a device specification must be given. For example:

```
@CAT RON*MYFILE.,F40
```

will create a permanent file on disc.
The EXEC has successfully cataloged that file.

Now we need to use some of the other programs stored on the disc. But first let's use a special command that uses two @ signs to make it easier to use the computer. The first is @@TTY. This command allows us to correct mistakes made in typing either characters or lines which have not yet been transmitted by the carriage return. We will type a C and a comma and indicate we would like to use an arrow as a character delete. We will type an L and a comma and indicate that we would like to use the question mark for the line delete:

```
@@TTY C,←,L,?
>
*@@PROCESSING COMPLETE*
>
```

That processing is now complete.

One of the most useful programs on the UNIVAC is called FURPUR. This program, the file utility processor, understands several commands which can be used to manipulate files and pieces of files called elements. We will use the copy command of FURPUR to copy an existing element called MAIN from OLDFILE to MYFILE. This element contains program statements in symbolic form and is called a symbolic element. FURPUR will tell us when the copy is complete.

```
@COPY,S RON*OLDFILE.MAIN,RON*MYFILE.MAIN
FURPUR 0026-02/21-14:50
1 SYM
END COPY
```

Another convenient FURPUR command is the print command which is abbreviated PRT. When used with a T option the PRT command will print the table of contents of MYFILE.

```
@PRT,T RON*MYFILE.
RON*MYFILE
ELT MAIN (0)
END PRT
```

The response tells us there is one element called MAIN in MYFILE.
The next step is to translate the symbolic element to a FORTRAN type element. For this, we use the FORTRAN compiler which is called with the @FOR card.

```
@FOR RON*MYFILE,MAIN
FOR SL1F-02/21/75-14:52:02(0,)MAIN @EOF
*WAIT-LAST INPUT IGNORED*
@EOF
*WAIT-LAST INPUT IGNORED*
>@EOF
STORAGE USED: CODE(1) etc.
END FOR
```

FOR is now ready to begin processing. We will type an @EOF to indicate we have no further instructions for the FOR processor. The message you see tells us that the last input has been ignored. Not being one to be ignored, I'm going to type it in again. Let's try it one more time and see what happens.

EDITORIAL NOTE: The input was ignored because the EXEC was not ready to receive more input from this terminal. This was indicated by the fact that no solicit character had yet been sent to the terminal.

The compilation has completed. Now we will see (using the Prt command with the T option again) that we have two elements in this file.

```
@PRT,T RON*MYFILE.
>
FURPUR 0026-02/21-14:54
RON*MYFILE
ELT MAIN(0)
REL MAIN
END PRT
```

We have now finished the compilation of the MAIN program and we see a second element listed named MAIN but of TYPE REL (for relocatable).

Our next step will be to show how to use Ed again to input a new subroutine. The subroutine happens to be in FORTRAN although Ed doesn't really care. We use the I option for the input. We will call this subroutine SUB.

```
@ED,I RON*MYFILE.SUB
ED 14.1C-02/21-14:54-(,>)
INPUT
>11:>
```
EDITOR will now begin to run. And when it is ready, it will solicit input and give us the line number for the first line. We will now type in the subroutine. We finally made our first mistake. We'll use a left arrow (which we defined using the @TTY command) to eliminate that last character. Then we'll type a plus which is what we meant to type in the first place. I will purposely make a mistake here and leave off the second parenthesis to show what happens when a mistake is encountered by the FORTRAN compiler. This is our last card.

```
1I: > SUBROUTINE TRIANG(A,B,C,AREA)
2I: > S = (A+B+C)*0.5
3I: > AREA = SQRT(S*(S-A)*(S-B)*(S-C)
4I: > END
5I: >
```

We will now briefly go to the EDIT mode of EDITOR by typing a blank line. And we will go to the top of the file and print the subroutine.

```
5I: > [CARRIAGE RETURN ENTERED]
EDIT
4: >
!: >P!
SUBROUTINE TRIANG(A,B,C,AREA)
S = (A+B+C)*0.5
AREA = SQRT(S*(S-A)*(S-B)*(S-C)
END
SCAN:4
EOF:4
!: >
```

You will notice as we print the subroutine that the left arrow has taken out the mistake that we originally typed. Now we are ready to get out of EDITOR and to attempt to compile this program....We again use the FORTRAN compiler. This time we will use the S option to list the program as it's being compiled.

```
!: >@EOF
LINES:4 FIELDATA
>@FOR,S RON*MYFILE.SUB
FOR S11F-02/21/75-14:57:58 (0,) SUB
>@EOF
SUBROUTINE TRIANG ENTRY POINT etc.
STORAGE USED: CODE (1) etc.
;
00101 1* SUBROUTINE TRIANG(A,B,C,AREA)
00103 2* S = (A+B+C)*.05
```
An @EOF will begin the compilation. As you can see the FORTRAN Compiler has caught the error that we made and now we will use ED again to go back into the element and correct the error.

> @ED,U RON*MYFILE.SUB
ED 14.1C-02/21-14:59- (0,1)
EDIT
ϕ > GO 3
AREA = SQRT(S*(S - A)*
(S - B)*(S - C))
3: > C/C)/C)/
AREA = SQRT(S*(S - A)*
(S - B)*(S - C))
3: >

This time we use a U option to indicate an update command. We would like to go to the third statement; so we will type GO 3. The error here is the lack of a second right parenthesis at the end of the statement. We will use the CHANGE command to change a C parenthesis to a C and 2 parentheses. EDITOR will process this CHANGE and then will type a new version of the line. Now we are again ready to go to the FORTRAN compiler which we can do this time without having all of the cards to the program typed out. The new control card terminates EDITOR as well as beginning processing by the new program. Now we type @EOF and the compilation begins

3: > @FOR RON*MYFILE.SUB
LINES:4 FIELDATA
FOR S11F-02/21/75-15:00:22 (1,)
> @ EOF
STORAGE USED: CODE(1) etc.
END FOR
>

Compilation has been successful. We will do a PRT,T to demonstrate what is now in MYFILE. There should be two symbolic elements, one of them MAIN, the other
one SUB, and two relocatable elements also named MAIN and SUB.

```plaintext
@PRT,T RON*MYFILE.
FURPUR 0026-02/21-15:01
RON*MYFILE
ELT MAIN (0)
REL MAIN
ELT SUB (1)
REL SUB
END PRT
```

Now the program is almost ready to execute. The only remaining task is to use the MAP processor to pull all the relocatable elements together. The MAP routine will generate a new element called the absolute element which will be placed back in MYFILE. We will name this element PROG. We will now tell the MAP processor to go to the file called MYFILE. Finally, we will tell it that we are ready for it to begin pulling in all of the elements necessary to create what is called the absolute version of the program.

```plaintext
@MAP,RON*MYFILE.PROG
MAP 26.2-02/21-15:02
@IN MYFILE.
@EOF
BAD STATUS: etc.
: 
END OF COLLECTION
```

I can see a mistake that I made. I forgot to type in the qualifier which is Ron. I only typed in MYFILE. We have to start the MAP process over again. And now we type the input card again, this time using the qualifier, and then type in the @EOF.

```plaintext
@MAP ,RON*MYFILE.PROG
MAP
@IN RON*MYFILE.
@EOF
START = 012002 PROG SIZE(I/O) = 4651/2348
SYS$*RLIB$.LEVEL 71-1C
END OF COLLECTION - TIME 1.415 SECONDS
```

The collection process is now complete. We will type a PRT,T to show what is now in the file. I've almost typed in the wrong line. I forgot the T option. Select the shift-question mark and the UNIVAC will forget the entire line. And then we'll do it again.
Now the program is ready to be executed. For this we can use the @XQT command as before. The EXEC takes a copy of the absolute element, in this case named PROG, out of the file we've created on the disc. It puts this copy of PROG into core and it tells the central processing unit that PROG is now a working program which we can execute. PROG is now complete. We will enter some data in PROG. And we get the answers as we did before.

Now that we're through using our program we are ready to disconnect from the UNIVAC. While you may be tempted to just hang up, just as with a person you ought to say goodbye first. This is how it's done:

> @FIN

[ACCOUNTING INFORMATION RECEIVED]
TAPE 2: TEXT EDITOR UNDER EXEC 8

PURPOSE:

In this half hour, some of the most useful capabilities of the TEXT EDITOR which is available to the U-1108 user operating under EXEC 8 are introduced and demonstrated. The TEXT EDITOR can be used to insert elements into program files and to input, correct, print and otherwise manipulate text in program and datafiles. The examples presented illustrate use of the ED processor in both INPUT and EDIT modes.

The set of EDIT-mode commands which is demonstrated covers those most likely to be used. Detailed information on ED processor calls and commands can be found in the UNIVAC 1108 Demand User's Manual published by the FCCC. (See the Reference list included in this booklet.)
TAPE 2: TEXT EDITOR UNDER EXEC 8

OUTLINE:

I. Insertion of an element in a file
   1. @ED,I command
   2. File TPF$.
   3. INPUT mode
   4. Line numbers
   5. EDIT mode
   6. @EOF
      6.1 Terminates ED processing
      6.2 Transfers text from ED's workspace to specified file

II. Manipulation of text with EDIT mode commands
   1. TAB command
      1.1 TAB character
      1.2 SET tab stops
      1.3 TAB space input
   2. PRINT command
      2.1 PRINT!
      2.2 LNPRINT and LNPRINT!
      2.3 QUICK and LNQUICK
   3. SCALE command
   4. CHANGE command
      4.1 String delimiter
      4.2 Replacement string
      4.3 Line specification
5. LOCATE command
   5.1 To find first occurrence of string
   5.2 To find all occurrences of string

6. INSERT command
   6.1 IB
   6.2 INSERT

7. RETYPE command

8. INLINE command
   8.1 String deletion
   8.2 String insertion

9. DELETE command

10. MOVE command
During the next thirty minutes we will continue our discussion of the use of the Univac 1108 computer located in Ft. Collins, Colorado. As you saw in the first tape of this series, we will be discussing the EDITOR processor. The first tape showed some things that you can do with EDITOR such as text insertion. Now we will talk further about how you actually use the EDITOR and its many commands.

The first use of the EDITOR processor is for text insertion. That is you may simply sit at the terminal and type your input into the system. The text EDITOR will receive your input and place it in a file. We will try a short example of that technique. You call EDITOR with @ED,I. The I option says "text insertion" and TEST specifies where you would like it placed.

```
@ED,I TEST
ED 14.1C - 02/28 - 13:45 - (.0)
INPUT
1I: > THIS IS A LINE OF INPUT
2I: > ADFDFFFVBVFERTER34
3I: >
```

If you do not specify a file name but simply reference an element name as I have here, it is placed in file TPF$ which is temporary and will disappear at the end of your run. Note that EDITOR signed on in INPUT mode. That is, it is ready for the first line of input. In the previous tape we used the EDITOR to input a program. As you can see here, EDITOR does not care what type of input it has. It may simply have text, or it may be FORTRAN, COBOL, or other language text. In fact it may be any input that you like. It does not have to meaningful input; it may be simply a series of characters. EDITOR, you will note, is numbering each line— one, two, three starting with the first line. There is no line number contained within the line itself. The line numbers are always supplied counting from the beginning of the file.

At this point, let's suppose that we have made a mistake on a line.

```
3I: > THIS OS NOT ALINE
4I: >
```

You can correct mistakes in a line by switching to EDIT mode. To enter EDIT mode you simply type a blank line—that, is a carriage return. EDITOR will
respond by switching to the EDIT mode. If you are in EDIT mode and you type a blank line, EDITOR will switch to INPUT mode.

```
4I:>
EDIT
3:>PRINT
THIS IS NOT ALINE
3:>
```

You can see here that we are now in EDIT mode; and EDITOR is indicating that line 3 is available for us to EDIT. We may print the line that the pointer is currently pointing at. That line is the one that we inserted with the errors. If we wish to make corrections we can do so at this time. Or we may simply terminate EDITOR if all corrections have been made. And EDITOR will at this time place the text (that we have input) into the file. You terminate by typing @EOF:

```
3:@EOF
LINES:3 FIELDATA
```

You notice that ED terminated with 3 lines and indicated the character code that it saved the text in. Now, it was not until I typed @EOF that the actual text of the element was copied to the file that I specified. In all previous operations, the text was actually contained within a working storage space that EDITOR uses internally.

Let's start now by cataloging a file to be saved for future use. We must then assign the file to our run in order to be able to access it.

```
> @CAT JERRY,F40
READY
> @ASG,A JERRY
READY
```

If you forget to assign the file, EDITOR will assign it for you automatically. But this type of processing incurs additional system overhead and is not recommended. At this point we will be ready to enter text. So ED,I for input mode, the file name, and some location for the text.
Again you can see that EDITOR signs on in the INPUT mode and is ready to accept input. We will now enter several lines:

```
11:>THIS IS LINE 1
21:>THIS IS LINE 2
31:>THIS IS LINE 3
41:>THIS IS ANOTHER LINE
51:>THIS IS LINE 5
61:>THIS IS THE LAST LINE
71:>
```

Once we have entered the text that we would like, we may then go into EDIT mode to make corrections or to make additional entries into the file.

Now as I have typed here, all input lines started in column 1. Let's suppose that we had intended to put in some input, perhaps a FORTRAN program, where we wished to start in another column. We could do that by simply spacing to the appropriate column and then typing the text. But there is a much easier way in EDITOR using the TAB feature. To use the TAB feature you must first go into EDIT mode and type the command TAB and specify a character to be used as the TAB character. I frequently use as the TAB character the semicolon. Any character is suitable as long as you realize that particular character cannot occur naturally in the text you will be typing. Any occurrence of that character on input will be interpreted as the TAB character. Now you must set the TAB stops where you would like them to be by using the SET command.
We are now ready to input data using the TAB stops. So we will return to INPUT mode and type in a line and then return to EDIT mode in order to print it.

```
INPUT
7I:>THIS;IS;A;TABBED;LINE
8I:>
EDIT
7:>P
THIS IS A TABBED LINE
7:>
```

As you can see the line has been spaced out. Notice that I used four TAB characters in that line; but I only had three tabs set. If there is no TAB set for a specific TAB character, EDITOR simply replaces it with a space.

Now how do you list the file once you have it entered? You may use the PRINT command followed by an exclamation point if you intend the entire thing to be printed. Or you may use LNPRINT which prints the line with a line number in front; or LNPRINT followed by an exclamation point which prints the entire working space again.

```
7>:PRINT!
THIS IS LINE 1
THIS IS LINE 2
THIS IS LINE 3
THIS IS ANOTHER LINE
THIS IS LINE 5
THIS IS THE LAST LINE
THIS IS A TABBED LINE
SCAN:7
EOF:7
0:>LNPRINT
  1:THIS IS LINE 1
1:>LNP!
    1:THIS IS LINE 1

SCAN:7
EOF:7
0:>
```
You will notice that this printout comes complete with all the spaces included that you typed in. There are two additional printing commands called QUICK and LNQUICK. They are equivalent to PRINT and LNPRINT except that they do not print spaces; they will compress all multiple spaces to a single space as you can see in this example.

```
0: >LNQUICK!
1: THIS IS LINE 1
2: THIS IS LINE 2
3: THIS IS LINE 3
4: THIS IS ANOTHER LINE
5: THIS IS LINE 5
6: THIS IS THE LAST LINE
7: THIS IS A TABBED LINE
SCAN: 7
EOF: 7
0: >
```

Notice that the tabbed line at the end has had all of the spaces squeezed out.

If you are interested in which column a particular entry is typed, you may ask for a SCALE to be PRINTED. The SCALE command simply prints a row of numbers that you may use to compare a particular printed line with.

```
0: >SCALE
12345678901234567890123 [etc.]
0: >P 1
THIS IS LINE 1
1: >
```

As you can see from this example, you will be able to count the characters to see where the end of this line is located. You saw me use the number 1 to print one line. You may print any number of lines that you like in that manner. You may type:

```
1: >P 5
ILLEGAL COMMAND.
1: >P 5
THIS IS LINE 1
THIS IS LINE 2
THIS IS LINE 3
THIS IS ANOTHER LINE
THIS IS LINE 5
5: >
```

You must have a space after the P. It prints from where the line pointer is for 5 lines. You may get to the desired position with a GO TO that line or simply by typing the line number.
If I type the line number 3, EDITOR will print line 3 as you can see here.
And then if I type PRINT 2, it will print line 3 and the next line. You may
go backwards one line by simply typing minus 1. Or you may go forward in the
work space by typing plus and the number.

Once you get the pointer to the line you would like to work on, you may
then change the contents of that line in any way that you like using the CHANGE
command. The general format of the CHANGE command is as follows: CHANGE
(which may be abbreviated to C) followed by some delimiter character which is
not to be used in the string that is to be changed. If you want to change a
string with a slash in it, in the example that I am using, you must use some
delimiter other than slash. The first character that occurs on the line after
the word CHANGE is taken to be the delimiter. I am using slash in this case.
Type CHANGE, slash, the thing to be changed, followed by a slash, followed by
the replacement string that you would like to insert. Now this can be either
nothing or it can be a whole long string of characters. Then you must tell
EDITOR whether you would like it done in this line alone, or all lines. If you
put a number here it indicates the number of lines to be processed. You must
indicate whether or not you intend for ED to replace the first occurrence or
all occurrences of this string of characters in the line. If you intend to
replace the first occurrence on this line only you may simply terminate the
line.

And you can see the correction has been made as I indicated.

EDITORIAL NOTE: The CHANGE command gives unexpected results if the string to
be changed is not unique in the line. Jerry intended to replace the word IS,
but the EDITOR replaced the letters IS of the word THIS which it found first.
If you want to find a certain string within the file and are not sure of the location (for example, the sequence NOT IS which I inserted in line 5), you may go to the top of the file and then locate the character string NOT IS:

```
5:>0
0:>L 'NOT IS'
THIS NOT IS LINE 5
5:>
```

The pointer was again placed at the line we worked on previously. To find all occurrences of this string, you may go to the top and use the LC command:

```
5:>Ø
Ø:>LC 'NOT IS'
      5:THIS NOT IS LINE 5
EOF:7
Ø:>
```

This locates all occurrences. Note that it prints the line with the line number in front of it. When EDITOR is finished, the pointer is placed at the top of the file so that you are ready to do an additional locate or to go to that specific line.

Let's suppose we would like to insert an additional line just before the sequence NOT IS in line 5. We must first go to line 5 by entering the number 5. We would like to insert before line 5 a new line. Use the IB command, which is the abbreviation for INSERT BEFORE. Do that by typing IB, a single space, and then the new information that you would like contained on the line. And now let's print the entire thing to be sure that the line was really inserted at the correct location. Then we'll go back to line 5 and observe that the line number of line 5 has been changed.

```
Ø:>5
THIS NOT IS LINE 5
5:>IB THIS IS THE NEW LINE BEFORE
6:>P!
THIS IS LINE 1
THIS IS LINE 2
THIS IS LINE 3
THIS IS ANOTHER LINE
THIS IS THE NEW LINE BEFORE
THIS NOT IS LINE 5
THIS IS THE LAST LINE
THIS IS A TABBED LINE
SCAN:8
EOF:8
Ø:>5
THIS IS THE NEW LINE BEFORE
5:>
```
The lines are numbered sequentially from the top to the bottom of the file. And if you insert a line, all line numbers below that are changed. The line that we were previously working on is now numbered 6. If we would like to insert a new line after line 6, we may insert the line by simply typing INSERT (abbreviated I) and the line. And again we will print the entire space to see exactly what is contained in it:

```
6: I THIS IS THE NEW LINE AFTER
7: P!
```

```
6: THIS IS LINE 1
```

```
6: THIS NOT IS LINE 5
```

```
THIS IS THE NEW LINE AFTER
```

```
THIS IS THE LAST LINE
```

etc.

If you have made a mistake on a line and it is not convenient to use the CHANGE command, you may replace the line with a RETYPE command. To do this you go to the particular line (6 is the line we were previously working on). To change that line you may RETYPE using R for the abbreviation.

```
>: 6
```

```
THIS NOT IS LINE 5
```

```
6: R THIS IS NOW LINE 6
```

```
>:
```

And if you print that line you will see that line 6 has now been replaced by the new information that I have put in.

There are many other ways to make changes. We will talk about the INLINE command which allows you to make corrections within any individual line. To do this you go to the specific line again. We will locate the word TAB to find the last line and use the INLINE command abbreviated INL to make corrections on this line.

```
0: L TABBED
```

```
THIS IS A TABBED LINE
```

```
9: INL
```

This command works as follows: it prints the line for you, allows you to space over to the correct column and replace, insert or delete a string of characters. For example, I may type D and delete the string of characters contained between the D and the exclamation point.

```
+++THIS IS A TABBED LINE
```

```
+D!
```

```
THIS IS A TABBED LINE
```

As you can see the string of characters has been deleted. You may insert a character string in the line using the I instruction as I'll show here. Again you type INLINE. Note that every time an INLINE command is used the system will print the results to show you that the corrections were actually made. The same applies to the CHANGE command that we saw demonstrated earlier.

```
9:>INL
++++THIS IS A TABBED LINE
+> INOT!
THIS IS NOT A TABBED LINE
9:>
```

Let's suppose that we have some lines in the text that are superfluous to the problem that we are trying to solve. We would like to simply delete those lines to get them out of the work space. We may delete lines using the DELETE command. We will go to line 1 and delete three lines. Let's print the work space to see what we have.

```
9:>1
THIS IS LINE 1
1:>D 3
0:>P!
THIS IS ANOTHER LINE
THIS IS THE NEW LINE BEFORE
THIS IS NOW LINE 6
THIS IS THE NEW LINE AFTER
THIS IS THE LAST LINE
THIS IS NOT A TABBED LINE
SCAN:6
EOF:6
0:>
```

As you can see lines 1,2,3 have been deleted. Another feature of EDITOR is the ability to rearrange lines using the MOVE command. The MOVE command simply takes text from the current location and moves it to wherever the pointer is now located. You position the pointer where you would like to have the information. In this case I will position it to line 1. We will now MOVE line 3 to that position. You type MOVE, the number of the first line that is to be moved, which is line number 3, and the number of the last line to be moved. And ED indicates that the line has been moved. We will print the work space to see that it really has.

```
0:>1
THIS IS ANOTHER LINE
1:>MOVE 3,3
THIS IS NOW LINE 6
1:>P!
THIS IS ANOTHER LINE
THIS IS NOW LINE 6
THIS IS THE NEW LINE BEFORE
```
Notice that the line that said THIS IS NOW LINE 6 has been moved to the second line position. Again notice that the line numbers changed when I deleted the lines in the top of the file. What this means in your processing is that you should work forward down the file making any changes required and then work backwards from the bottom if you want to delete lines. This is not necessarily the most efficient way to use EDITOR; but it is the easiest way for the user. There is another command now that is very similar to MOVE called DITTO. It works exactly like MOVE except it does not delete the lines after it has copied them. We are at line 0; and we would like inserted at the top line 2.

\[ \emptyset:)\text{DITTO 2,2} \]
\[ \text{THIS IS NOW LINE 6} \]
\[ 1:)! \]
\[ \text{THIS IS NOW LINE 6} \]
\[ \text{THIS IS ANOTHER LINE} \]
\[ \text{THIS IS NOW LINE 6} \]
\[ \text{THIS IS THE NEW LINE BEFORE} \]
\[ \text{THIS IS THE NEW LINE AFTER} \]
\[ \text{THIS IS THE LAST LINE} \]
\[ \text{THIS IS NOT A TABBED LINE} \]
\[ \text{SCAN:7} \]
\[ \text{EOF:7} \]
\[ \emptyset:> \]

As you can see, a copy of the line that says THIS IS LINE 6 has been moved to the top of the file.

A question frequently asked on long files is what is the last line in the file. To answer this question you simply type LAST. And the system will position the pointer to the last line in the file. However, be aware that you cannot change or alter this line if you arrived at it in this manner. In order to change this line once you've discovered its number, you must type the line number itself before a change is permitted. In order to do additions to the bottom of the file you may use the command APPEND. This moves you to the bottom of the file and switches to INPUT mode. At this point you may type additional lines.

\[ \emptyset:)\text{LAST} \]
\[ \text{THIS IS NOT A TABBED LINE} \]
\[ 7:>7 \]
\[ \text{MISSING NUMERIC} \]
\[ 7:>7 \]
\[ 7:>P \]
There are two methods of getting out of EDITOR other than the @EOF that we have previously talked about. One of these is EXIT, and the other is OMIT. Let's suppose that you have been editing a file and have made some drastic changes which were in error. This is easy to do using the general form of the CHANGE command. The way out of that dilemma is to type OMIT. If you are in EDIT mode and you type OMIT, then no changes whatsoever will be made to the element. You are out of EDITOR at this point and back in the operating system.

The other way to get out of EDITOR is use the EXIT command which will terminate ED exactly the same as though an @EOF had been entered.

This concludes the session on EDITOR. For further information you should consult Chapter 18 of the Univac Publication No. UP-4144, Revision 3. This is the manual that discusses the EXEC operating system and all of the language programs used on the Univac 1100 series systems.
TAPE 3: INTRODUCTION TO CTS

PURPOSE:

This taped half-hour with the U-1108 operated by Ft. Collins Computer Center is intended to introduce the user to some of the features of operation under the Conversational Time Sharing system - CTS.

CTS has a set of commands which allows the demand user to interactively accomplish a variety of tasks leading to some useful result. These tasks fall into several categories:

1. Establishing program and data files.
2. Entering symbolic statements in one of the available programming languages into a file
3. Checking the statements entered for syntax errors (currently prescan units are available for FORTRAN AND BASIC)
4. Executing a finished program
5. Correcting or modifying existing files

In this lecture-demonstration a subset of the CTS commands is used which makes it possible to accomplish all of these tasks. CTS has many useful capabilities not covered here. Further information can be obtained from the CTS manuals mentioned in the Reference List of this booklet.
I. Establishing an active run on the FCCC U-1108
   1. Setting up a terminal to computer telephone link
   2. Entering the site ID
   3. The @RUN command
      3.1 Time limit specification
      3.2 Page limit specification
   4. Further information on demand mode in TAPE 1.

II. Entering CTS mode
   1. @CTS command
      1.1 Computer responds inappropriately
         1.1.1 Probable "garbage" character on line
         1.1.2 U-1108 did not recognize command
      1.2 Reenter @CTS command
         1.2.1 Command recognized
         1.2.2 CTS response line received at terminal
   2. CTS news file
   3. *HELP command
      3.1 The asterisk on CTS commands
         3.1.1 Optional unless *NUMBER in effect
         3.1.2 Arbitrary on *MANUAL command used to discontinue
         automatic numbering
      3.2 The teach function of HELP
      3.3 HELP commands - EXPLAIN BASIC
      3.4 EXIT from HELP
III. CTS mode commands

1. *BASIC
   1.1 Command abbreviation
   1.2 SYNTAX prescan solicit character

2. Executing a previously stored program, AVERAGE
   2.1 Calling up the program with *OLD
   2.2 *RUN
   2.3 Keyboard data entry

3. The CTS desk calculator mode
   3.1 Evaluating an expression
   3.2 TYPE to print the value

4. *PRINT ALL to list the program AVERAGE

5. Correcting the text
   5.1 *CHANGE
      5.1.1 Abbreviation C
      5.1.2 C / 2 / 1 / 150
   5.2 Rerun using *RUN

6. Generating a new program
   6.1 *NEW TRIANGLE
   6.2 *NUMBER
   6.3 The SYNTAX prescan unit
   6.4 *PRINT
   6.5 Input error correction
   6.6 *MANUAL

7. Executing the new program
   7.1 *RUN
   7.2 Keyboard data entry
   7.3 STOP to terminate BASIC program execution
8. Saving the new program

8.1 *SAVE

8.1.1 Program name
8.1.2 System SAVE location

8.2 *LIST SAVED

9. Creating a data file

9.1 *CLEAR

9.1.1 Discontinue syntax scanning
9.1.2 Change in solicitation character

9.2 *NEW MYDATA.

9.2.1 *NUMBER
9.2.2 Line-by-line data entry
9.2.3 *MANUAL

9.3 *SAVE into nonsystem generated location

9.3.1 Public or private file?
9.3.2 Read or write keys?
9.3.3 Device characteristics?

10. Recalling the TRIANGLE program

10.1 *OLD TRIANGLE

10.2 *RUN

10.3 Using the prestored data

10.3.1 EXEC command required
10.3.2 @ADD MYDATA.

10.3.3 STOP

IV. Terminating the RUN and the connection

1. @FIN to receive accounting information

2. @@TERM to disconnect the telephone link
TAPE 3: INTRODUCTION TO CTS

SCRIPT AND EXAMPLES:

We have just established the connection for the UNIVAC 1108 at Fort Collins, Colorado used by the U. S. Forest Service. Today we will be studying a system called the Conversational Time Sharing System or CTS for short. We hope to demonstrate to you the use of CTS and possibly answer some of the questions you may have about CTS. As with all demand terminal operations you must begin by entering your SITE ID:

UVX945

After recognizing the SITE ID, the U-1108 will respond with a line indicating the operating system currently in use:

*UNIVAC 1100 OPERATING SYSTEM VER. 31.244.17 FC(RSI)*

Next you must supply the accounting system on the 1108 some information in the form of a RUN command. An option permits you to indicate the maximum number of pages of output you intend to produce.

@RUN TECH,1151006602,SEGERS,5,100
DATE: 021275    TIME: 121918

EDITORIAL NOTE: TECH is supplied by the user to identify his particular RUN. The account number was assigned by the FCCC when this account was established. SEGERS identifies the user. The time and page limits selected are 5 minutes and 100 pages, respectively.

The system responds with a date and time. Up to this point we have done exactly the same thing that you would do to connect for any demand mode processing. For more information about this, you may refer to the first tape in this series on the use of the demand mode on the UNIVAC system.

Now we will connect to the Conversational Time Sharing system which runs under the UNIVAC 1108 EXEC 8 Operating System. To do this we will type:

@CTS
DATA IGNORED - IN CONTROL MODE

The indication you have in front of you is caused by the machine not recognizing the input data line as a control card, possibly due to some garbage character on the line. We will try again:

@CTS
CTS 3.1 12:26:24
TYPE *HELP IF YOU NEED HELP AT ANY TIME

This time the CTS control card was recognized and honored by the system.
CTS is presently signing on and at first looks to see if there is a NEWS file. If there is a NEWS file available, it will ask you at this time if you would like the NEWS. If you would, you may type YES to the question and it will print the NEWS for you on the terminal. The line - TYPE *HELP IF YOU NEED HELP AT ANY TIME - indicates that there is a HELP feature available to help you use CTS. The asterisk in front of a command, such as *HELP, is permissible in all cases in CTS but it is required in one. That one is the *MANUAL command that we will cover later. To demonstrate the use of the HELP verb, we will type:

```
->HELP
WHEN YOU NO LONGER NEED HELP, TYPE EXIT
TEACH? - TYPE YES OR TYPE A HELP COMMAND EXPLAIN BASIC
*BASIC - THIS COMMAND CALLS THE BASIC PRESCAN UNIT SO THAT
SUBSEQUENT PROGRAM STATEMENT INPUT WILL BE SCANNED FOR
SYNTACTIC CORRECTNESS.
MORE EXPLANATION? NO
MORE HELP? >EXIT
RETURN TO CTS MODE
->
```

The HELP processor will sign on and give you an indication that it has a TEACH function available to teach you how to use the HELP verb. If you need such teaching, you may answer YES, or you may simply enter a HELP COMMAND such as EXPLAIN BASIC. There is a CTS command BASIC; and as you can see from the indications here, the command BASIC is used to call the BASIC prescan unit to scan programs that are input in the BASIC language for syntactic correctness. If you would like additional explanation you may type YES. If you do not need additional information you may type NO. It then asks if you would like more HELP. If not, you may type EXIT and the program will return to CTS mode. Let's now call for the BASIC prescanner:

```
-->BAS
>>
```

As you observe from this example, we may abbreviate any commands to three characters and some commands may be abbreviated to less than three characters. Note the change in the input solicitation character. We now have two "->" symbols for solicitation as opposed to the "->". This indicates that the syntax scanner has been turned on.

Let's now get an old program by simply typing 0 and the name of a program - AVERAGE:

```
>>OLD AVERAGE
>>
```

The solicitation character indicates that our request has been complied with. We may RUN this program by typing:
The program inquires for the number of numbers to be averaged. For example purposes, I will choose 5. I should now enter the numbers one at a time. I will use the numbers 1.2, 3.4, 5.8, 10.0, and 1.3. But the program did not stop to ask for the fifth value:

```
>1.2
>3.4
>5.8
>10.0
THE AVERAGE OF YOUR 5 NUMBERS IS 4.423333
TIME : .031
>>
```

Therefore, I expect that the answer that it has given me is incorrect. Let's test, using the calculator mode, to find what the correct answer is. To use the calculator mode, you may simply ask it to:

```
>>TYPE 1.2+3.4+5.8+10.0+1.3
21.7
>>
```

The answer is 21.7. To find the average we now type:

```
>>TYPE 21.7/5
4.34
>>
```

The answer is 4.34, which is not the answer given by the program. Let's list the program to see if we can find what is wrong. We do this by saying:

```
>>PRINT ALL
110 PRINT
120 PRINT 'ENTER NUMBER OF NUMBERS TO AVERAGE'
130 INPUT N
140 PRINT 'ENTER THE ';N;' NUMBERS ONE AT A TIME'
150 FOR I = 2 TO N
160 INPUT A
170 LET B = A + B
180 NEXT I
190 LET A = B / N
200 PRINT
210 PRINT 'THE AVERAGE OF YOUR ';N;' NUMBERS IS ';A
220 STOP
230 END
>>
```
This prints the entire BASIC program as it currently exists in our work space. You notice that at line 150 of this program the loop statement appears to have an incorrect starting value since I should start from one to read in all of the values. We will correct the value by typing C, which is the abbreviation for CHANGE, a delimiter character, 2, an additional delimiter, 1, and then indicate which line the change is to be effective on, in particular, line 150.

```
> C /2/1/150
  150 FOR I = 1 TO N
>
```

Notice the line numbers in the front of each line of the program. The system now responds with the new value of that line and we may again run the program.

```
> RUN
OLD: AVERAGE
ENTER NUMBER OF NUMBERS TO BE AVERAGED
> 5
ENTER THE 5 NUMBERS ONE AT A TIME
> 1.2
  1.2
> 3.4
  3.4
> 5.8
  5.8
> 10.1
  10.1
> 1.3
  1.3
THE AVERAGE OF YOUR 5 NUMBERS IS 4.3399999
TIME: .033
```

Notice that it waited for the last number and printed the average, which is the answer that was given by the desk calculator mode in the previous test.

Now let's see how this whole system came about. We will start back at the top by creating a new program. To do this you indicate your wishes to have a new program by typing new and the name of the program.

```
> NEW TRIANGLE
>
```

The system is now acting very much like a stenographer. That is, it is taking the information from the terminal as I type it and placing it on a scratch pad. The information on this scratch pad will later be disposed of as you desire. To have the stenographer supply you with the line numbers instead of you telling the stenographer which line number you use the NUMBER command. There
are two parameters on the NUMBER command. The first parameter tells the line number to start with, the next parameter tells what the increment is to be from one number to the next.

>>NUMBER 100,10
100 >

NUMBER 100,10 is the default value and therefore you do not have to type the parameters if those are the values you want. We are now ready to input the program. We are at line 100. We will enter a PRINT statement first:

100 >PRINT
110 >PRINT ' ENTER LENGTHS OF SIDES SEPARATED BY COMMAS'

As you recall, the BASIC syntax scanner is turned on. If we make a mistake in typing in the input, the BASIC syntax scanner, which is very much like an English teacher, is sitting there checking for spelling and other errors in the language itself. It will stop the number generation long enough for you to correct any mistakes that you have made. It is now ready for the next line.

120 >INPUT A,B,C
130 >LET S = (A + B + C) * .5
140 >PRINT
150 >D = SQRT(S*(S - A)*S - B)*(S - C))
150 "THE STATEMENT CONTAINS UNMATCHED PARENTHESES"

We are now using the BASIC language but that is not a requirement. There are many languages available on the UNIVAC 1108. As you can see from the error there is unmatched parentheses in line 150. We may again use the CTS print statement, abbreviated P, to print the line that contains the error.

>>P
150*D = SQRT(S*(S - A)*S - B)*(S - C))
>>
***PARITY ERROR***

The parity indication that you see on your screen means that some character was sent down the data line that was not recognized by the 1108 and the last transmission, in this case a blank line, was ignored. The system is now ready for additional input. We can change the line we just entered by simply saying "CHANGE":

C *S
I'm sorry - I must delete the line using the CNTRL/X key. I forgot the delimiter.

```
C /*S/*)S/
```

Again it pointed out another error in the command - square root is spelled SQR.

```
150 [SQ] IS AN IMPROPER FUNCTION IN A 'LET' STATEMENT
150*D = SQT(S*(S-A)*(S-B)*(S-C))
```

I can again CHANGE:

```
>>C /T/R/
150 D = SQR(S*(S-A)*(S-B)*(S-C))
160 >
```

This time the syntax scanner let the thing pass; and the number mode came back with the next number and is now ready for input. We have now computed the area of a triangle given its sides and we must simply present the results:

```
160 >PRINT 'A = ';A
170 >PRINT 'B = ';B
180 >PRINT 'C = ';C
190 >PRINT 'AREA = ';D
```

We wish the program to loop, so we will input a GO TO 120 and this is the end of the program. We will type END and now we have come to the command MANUAL:

```
200 >GO TO 120
210 >END
220 >*MANUAL
```

In this case the asterisk is required to indicate that this is not additional data to be taken by the stenographer and placed in the files. This is a command to CTS to do some additional work. And in particular it is to return you to the non-numbering mode. We will now print all of the program to show that it has been correctly entered into the system.
To run this program you simply type RUN:

```
>>RUN
OLD: NAME
ENTER LENGTHS OF SIDES SEPARATED BY COMMAS
```

We are now ready to enter the data. We will use as a test case the 3, 4, 5 right triangle.

```
>3,4,5
A = 3
B = 4
C = 5
AREA = 6
```

As you can see, the area is 6. The program is now ready to accept additional input data, but since we have none we will simply type STOP:

```
>STOP
TIME : .029
```

This will terminate the execution of the program so that we may call on the file clerk section of CTS to actually SAVE the program so that we may refer to it later. To do this you type SAVE. It remembers the name that you gave it from the NEW command when you first started into CTS. A copy of the program is created by ditto type operation and saved in the public save area. To find out what you have access to you may LIST SAVED.
LIST SAVES
tech.
BASIC TRIANGLE SIZE 3
BASIC NAME SIZE 3
BASIC TRI SIZE 3
BASIC AVE SIZE 4

Notice the spelling error on SAVED. The system looks at the line and if the first three characters are acceptable, then that's enough information for it to process it. Therefore it is very forgiving as far as spelling errors on commands are concerned. You will notice the name TRIANGLE which is at the top of the list. That is the last program that has been added and it's the one that we just created. Now we wish to create a data file to be used by TRIANGLE. To work on data we must get rid of the BASIC prescanner. To do this we will type CLEAR.

>>CLEAR
->

Notice the solicitation character change. We are now out of prescan mode. We would like to create a new program called MYDATA which will be the data for this program. We type NEW MYDATA followed by a period since we wish for this to be a file and not an element within a program file. And we now wish for the system to number.

->NEW MYDATA.
->NUMBER

Again the system is taking the information and placing it line by line on its internal scratch pad. We enter the data one line at a time as we would like the program TRIANGLE to have it.

100 >4,4,4
110 >3,4,5
120 >2,2,2.828
130 >*MAN
->

We must now cause the numbering to stop with the manual command and notice the asterisk is required. We now have the program in the work space which we may SAVE. The system remembers from previously when we typed in the information
about the file name with the period. That told the system to remember that this is a file and not an element of a program file. Therefore, several pieces of information have to be specified.

```
->SAVE
IS THIS FILE TEMP, PUBLIC, OR PRIVATE? >PRIVATE
READ AND WRITE KEYS: >
DEVICE CHARACTERISTICS: >
*CRE,U MYDATA,F2
```

For example, is the file to be temporary, public or private? We will specify a private file. It asks if we would like keys. No response indicates that we would like no keys. Device characteristics: a carriage return indicates that we wish to specify default mass storage file. The system gives indication of the CREATE command that you could have used initially to create this file and then comes back indicating that it has finished the SAVE.

Now we need to get the program back.

```
->OLD TRIANGLE
->RUN
ENTER LENGTHS OF SIDES SEPARATED BY COMMAS
```

The program is now reloaded and we are ready to RUN and it is asking for the lengths of the sides. We can now ask the operating system or the Exec to ADD MYFILE. Notice this is the first Exec control card we have used in the entire session. Again, you are required to have the period on the filename.

```
>ADD MYFILE. [CNTRL/X entered]
@ADD MYDATA.
A = 4 UNITS
B = 4 UNITS
C = 4 UNITS
AREA = 6.9282032 SQUARE UNITS
e tc.
```

I'm sorry - the name of the file should be MYDATA. The operating system adds that file just as though you were typing the data in from the terminal; and you can see that the program is generating answers again, just as though I were typing the data. It's finished with all the data from the program and is back requesting additional data from the terminal. In order to stop the program, again I type STOP as before.

```
?>STOP
PROGRAM STOPPED.
TIME: .069
```

The program stops and we are now back in the Conversational Time Sharing mode ready to continue with our next task. To terminate from the entire system we type:
This will terminate our connection to the system and make the system ready to connect for the next user who may enter a RUN card or you may type:

@@TERM

to terminate this connection. This concludes the basic CTS presentation. For additional information on CTS please refer to the Advanced CTS Tape or to the UNIVAC CTS Manual.
PURPOSE:

In the preceding half hour session, enough features of the Conversational Time Sharing mode of operation on the U-1108 were demonstrated to enable a user to store and execute a program and solve a numerical problem. In this second half hour devoted to CTS, two additional features will be introduced and demonstrated which allow the user to make more efficient use of both his time and that of the computer. These features are referred to as desk calculator mode of operation and CTS subroutines:

1. CTS permits the immediate evaluation of numerical expressions for desk calculator type computations.

2. CTS is an interpretive programming language in which subroutines can be written for later execution. CTS commands, preceded by line numbers, can be entered as executable statements in these subroutines.
TAPE 4: ADVANCED CTS

OUTLINE:

I. Introduction to CTS
   1. TAPE 3 reference
   2. The work space, f, and SAVE file, F
      2.1 *LIST contents of f
      2.2 *LIST SAVED for table of contents of F
      2.3 The work space line pointer, P
         2.3.1 *PRINT L
         2.3.2 Line specification, L

II. Desk Calculator Mode
   1. Example calculation of hypotenuse of right triangle
      1.1 *TYPE command
      1.2 *SET command
         1.2.1 Setting variable to numerical values
         1.2.2 Setting variable to string value
   2. Evaluation of string expression stored by *SET

III. CTS Subroutines
   1. *DELETE ALL to clear workspace
   2. *SUBROUTINE CALC
      2.1 Names all of f "CALC"
      2.2 Specifies that CALC is a CTS subroutine
   3. Insertion of lines into CALC
   4. *SAVE subroutine
   5. Calling the subroutine
   6. Modifying CALC to permit looping
6.1 *QUERY
6.2 *REPLACE CALC
6.3 Subroutine loops
   6.3.1 *JUMP
6.3.2 Statement labels
6.3.3 Getting out of the loop
   6.3.3.1 @@X TIO
   6.3.3.2 *JUMP+
6.4 *RESEQUENCE line numbers
7. Modifying CALC to store output in f
   7.1 *DELETE ALL
   7.2 *GENERATE
   7.3 *JUMP 20
   7.4 Printing generated values
   7.5 *SAVE generated values into file element TPF$.DATA
7.6 Execution of old program, TRIANGLE, by CALC
7.7 Statement to indicate to user that control has returned to CALC
7.8 Difference between CTS desk calculator and subroutine command
   7.8.1 CTS commands entered with no line number are executed immediately
   7.8.2 CTS commands entered with a line number are stored in the work space for later execution
7.9 *MANUAL to disable automatic line numbering
7.10 *REPLACE CALC
8. Calling the completed subroutine CALC
   8.1 *CALL CALC
   8.2 Keyboard data entry
For the next thirty minutes we will be continuing our discussion of the Conversational Time Sharing system, CTS, as used on the UNIVAC 1108 at Ft. Collins, Colorado. In the introductory session, we learned how to insert and execute BASIC programs. We will now learn how to use the more advanced features of CTS which make user interface easy and convenient.

But first, we need some definitions. Earlier we talked about the work space. The work space is known in the documentation as lower case f and you see the current value of the work space listed on the screen:

```
->P A
100 PRINT
110 PRINT 'ENTER LENGTHS OF SIDES SEPARATED BY COMMAS'
120 INPUT A,B,C
130 LET S = (A+B+C)*.5
140 LET D = SQRT(S*(S-A)*(S-B)*(S-C))
150 PRINT
160 PRINT 'A = ';A;' UNITS'
170 PRINT 'B = ';B;' UNITS'
180 PRINT 'C = ';C;' UNITS'
190 PRINT 'AREA = ';D;' SQUARE UNITS'
200 GO TO 120
210 END
END OF FILE
```

When we used the SAVE command in the first session, we SAVED this work space into capital F (as it is referred to in the documentation) which is a part of the permanent file system. To look at the contents of big F, we may type LIST SAVED:

```
->LIST SAVED
MASTERFILE
SIZE TYPE NAME
4 BASIC AVE
4 BASIC TRIANGLE
```

This lists the names of the programs which are currently contained under your name in the master file space. Once we've talked about the master file and the work space, we now need to known how to point to the various pieces that are contained within the work space. To do this, we will talk about the line pointer—referred to in the documentation as capital P. The line pointer is simply a pointer within the CTS system program, that points to the current line you are working on. To print the current line you may use the print command.
As you can see the pointer is currently located at the top of the file. To go to the specific line of the file we may type go to 4.

Let's try GO TO 100 which was the first line of our file:

Editorial Note: The correct command in CTS is GO 100, rather than GO TO 100.

You can see here it has moved to line 100, and we may now print the value of that line which contains the word PRINT.
It is not necessary to use the GO command; you may simply PRINT a specific line.

->PRINT 120
120 INPUT A,B,C
->

All commands have the capability of having the line numbers inserted in them. You may specify a single line as done here, or you may specify a group of lines:

->PRINT 100,200
100 PRINT
110 PRINT 'ENTER LENGTHS OF SIDES SEPARATED BY COMMAS'
:  
200 GO TO 120
->

That says print all lines between 100 and 200, inclusive.

There is nothing to prevent you going forward or backward in the work space. For example, we may print 200 back to 180.

->PRINT 200,180
200 GO TO 120
190 PRINT 'AREA = ';D;' SQUARE UNITS'
180 PRINT 'C = ';C;' UNITS'
->

This leads directly to the capability of printing the file from the bottom up. To do this you type PRINT, the exclamation character which is used to indicate the bottom of the file, a minus to indicate that you wish to go backward, and the number of lines you would like to go backward.

->PRINT!-4
210 END
200 GO TO 120
190 PRINT 'AREA = ';D;' SQUARE UNITS'
180 PRINT 'C = ';C;' UNITS'
->

PRINT 100 + 3, will print lines 100 and the following three lines regardless of their line numbers.

->PRINT 100+3
100 PRINT
110 PRINT 'ENTER LENGTHS OF SIDES SEPARATED BY COMMAS'
120 INPUT A,B,C,
130 LET S = (A+B+C)*.5
->
PRINT 120 minus 1, will print line 120 and the preceding line.

```
->PRINT 120 - 1
120 INPUT A, B, C
110 PRINT 'ENTER LENGTHS OF SIDES SEPARATED BY COMMAS'
->
```

I'd like to now talk about desk calculator mode under CTS. The example that I will be using throughout this session concerns computing the length of the hypotenuse of a right triangle. Initially we will use the TYPE command:

```
->TYPE SQR(3**2 + 4**2)
5.
->
```

The correct answer is 5 since this is a 3, 4, 5 right triangle. If we had a number of values to compute, this particular mode of operation would require a lot of typing. So we will use on additional feature called the SET statement. To SET variable values, you type SET, name of the variable and the sign and the value. You may now have the program TYPE the value of A.

```
->SET A = 3
->TYPE A
(8) VARIABLE AX IS UNDEFINED
->
```

I seem to have received a garbage character on the line. We will try again.

```
->TYPE A
3
->
```

And the variable A is now SET to 3. Similarly, we may SET B equal to 4.

```
->SET B = 4
->
```

Now we will set a new variable, E, equal to a string value. You may now TYPE the value of E.

```
->SET E = 'SQR(A**2 + B**2)'
->TYPE E
SQR(A**2 + B**2)
->
```

And as you can see it is the string expression we entered earlier. To evaluate that expression, you enclose the string variable name in percent signs. You may change the percent signs to parenthesis or any other characters you desire, using a star command as documented in the manual. We may now type the evaluation of E and as you can see the result is 5.
We may change the value of A or B. SET A equal to 1 for example; then type the evaluation of $E$ again.

```
->SET A = 1
->TYPE %E%
4.1231856256176685
```

You can see that it has computed the new value of the hypotenuse of this new right triangle.

Again this is a lot of typing for a small amount of work produced. So we will introduce the concept of the CTS subroutine. You may enter information into the work space and have CTS process it as though you had typed it in from the console.

```
->CLEAR
->NEW CALC
```

First you must clear the work space, and then we will initialize a new subroutine called CALC. We will enter into this subroutine at line 100. And at line 110, since this is a subroutine, we will have it RETURN to CTS.

```
->100 TYPE 'THE ANSWER IS' SQR(A**2+B**2)
->110 RETURN
```
We now save this subroutine.

->SAVE
->

We may now call CALC.

->CALL CALC
THE ANSWER IS 4.1231856256176685
->

As you can see it used the previous values of A and B that we used in the TYPE statement. To keep from having to use the SET statement to change the values of A and B, we will insert some additional lines in our subroutine to QUERY the terminal for data. At line 90 we will enter QUERY--abbreviated QU, the variable whose value we wish returned, and the question to ask of the user.

->90 QU A WHAT IS THE VALUE OF A
->

Similarly at line 95, we will QUERY for the value of B.

->95 QU B WHAT IS THE VALUE OF B
->

The subroutine now looks like this.

->P A
 90 QU A WHAT IS THE VALUE OF A
 95 QU B WHAT IS THE VALUE OF B
 100 TYPE 'THE ANSWER IS 'SQR(A**2+B**2)
 110 RETURN
END OF FILE
->

We REPLACE the subroutine and call the subroutine. It asks for the value of A. We will type in 3. It asks for the value of B. We will type in 4.

->REPLACE
->CALL CALC
WHAT IS THE VALUE OF A >3
WHAT IS THE VALUE OF B >4
THE ANSWER IS 5.
->

It prints the answer as 5, which is the length of the hypotenuse. If we wish to do this for other values of A and B we may simply call the subroutine again.

In order to keep from having to type that much, we will add a loop to the subroutine to get it to do the same operation again and again. The top of the
loop will be at line 90 which is the QUERY statement so we will use the CHANGE command to change the Q in the beginning of QUERY to 10 (which is a line number), a space, and then the letter Q. And we must indicate line 90 since the current line pointer is at the top of the file, not at line 90.

```
->C /Q/ 10 Q/90
90 10 QU A WHAT IS THE VALUE OF A
->
```

We have inadvertently added an extra space at the beginning of the line; and since all labels must begin in the first column we'll change the initial space to a non-existent value. As you can see it has slid the statement number over to the first column.

```
->C / //
90 10 QU A WHAT IS THE VALUE OF A
->
```

Now at line 105 we will enter the looping item called a JUMP statement. The command is JUMP, space, and then the location to jump to—in particular, label 10.

```
->105 JUMP 10
->
```

The subroutine now looks like this.

```
->P A
90 10 QU A WHAT IS THE VALUE OF A
95 QU B WHAT IS THE VALUE OF B
100 TYPE 'THE ANSWER IS 'SQR(A**2+B**2)
105 JUMP 10
110 RETURN
END OF FILE
->
```

We will REPLACE this in the save area, and then call it.

```
->REPLACE
->CALL CALC
WHAT IS THE VALUE OF A >3
WHAT IS THE VALUE OF B >4
THE ANSWER IS 5.
WHAT IS THE VALUE OF A >1
WHAT IS THE VALUE OF B >2
THE ANSWER IS 2.2368679774997897
WHAT IS THE VALUE OF A >
```

The value of A is 3 in our example, B is 4, the answer is 5; and as you can see it has come back to ask for the value of A again. We can give A equals 1 and B equals 2; and we observe that CALC has come back to ask for the value of A...
again. And that appears to be an infinite loop. We have not made any provision to stop the subroutine from looping and therefore it may continue indefinitely. One means of stopping any program on the UNIVAC is the @@X command. @X, space, and then the character C is to interrupt the program; and optionally the letter O at this point would cause any output to be terminated also. Again we appear to have a garbage character. Notice that the executive solicit character came back but that CTS is still waiting for an input line before it will be interrupted. So we must enter a blank line. CTS recognizes the interrupt and returns to command mode. To keep this from happening again, at line 92 we will enter a jump plus which is equivalent to a RETURN. We will only do this if the value of A equals 0.

->92 JUMP + IF A=0
->

Now let's look at the subroutine again to be sure we are aware of exactly what's going to happen.

->P A
90 10 QU A WHAT IS THE VALUE OF A
92 JUMP + IF A=0
95 QU B WHAT IS THE VALUE OF B
100 TYPE 'THE ANSWER IS 'SQR(A**2+B**2)
105 JUMP 10
110 RETURN
END OF FILE
->

As you can see at line 92, now as soon as it clears the value of A it will RETURN if A is 0. We now must REPLACE CALC.

->REPLACE CALC
->CALL CALC
WHAT IS THE VALUE OF A >3
WHAT IS THE VALUE OF B >4
THE ANSWER IS 5.
WHAT IS THE VALUE OF A >Ø
->

The value of A is 3; the value of B is 4; the answer is 5. We will give it a value of 0; and CALC returns immediately to command mode.

Now as you saw in the last listing, the subroutine line numbers are becoming rather messy. So we will resequence these values so that they are consecutive—starting at 100, in steps of ten.

->RES
->
Next thing we would like to do is to be able to save these values instead of having them printed at the terminal. To save these you use a GENERATE command, to GENERATE new lines in the work space. We will have difficulty doing that, since our program is in the work space. We must first have the subroutine CLEAR the work space. So at line number 50, we will input a DELETE ALL.

->50 DELETE ALL
->65 G Ø
->

When this statement is executed, it will clear the work space. At line 65 we will move the line pointer to location 0, that's at the top of the file. We would now like to input the GENERATE immediately after the TYPE statement; but since we have lost line position when we resequenced, we must locate the TYPE statement.

->L TYPE
130 TYPE 'THE ANSWER IS 'SQR(A**2 + B**2)
->

The TYPE statement is at line 130. We must force the computation now to take place; so at line 131 we will input a SET statement: SET C equal to the evaluation of E. At line 132 we will input the GENERATE statement to generate the values.

->131 SET C = %E%
->132 GEN '%A%,%B%,%C%'
->

The way the GENERATE statement works is you say generate and then a quote mark and then a string of characters that you would like generated. What I would like generated is the evaluation of A (which is the first side that we entered), a comma, the evaluation of B, a comma, and the evaluation of C. Be sure to close the quote marks. Now then, we would like to locate the JUMP that is immediately after the input value so that we do not RETURN when we get the value of A. We wish to do some other processes, so let's JUMP in the negative direction from where we are. Notice the minus sign after the word JUMP. This indicates backward processing.

->L JUMP -
110 JUMP + IF A = Ø
->

It located the JUMP at line 110 which is JUMP plus if A is Ø. What we wish to do at this point is to JUMP to label 20. So we will change the plus to a 20.

->C /+/20/
110 JUMP 20 IF A = Ø
->
Note that there is no line number necessary since we are at the proper location. We will go to the bottom of the file now at line 1000, ask for numbers to be generated automatically by the system starting at 1000 in increments of 10, and at line 1000 we will put label 20, and ask it to PRINT the values that we have generated in the work space.

```
>NUM 1000
1000 >20 P A
1010 >SAVE TPF$.DATA
1020 >
```

We will then ask it to SAVE the current work space in the file TPF$. This is a temporary program file which is assigned by the system to every run it starts. The element name we'll use is DATA. We will now have the subroutine CALL the program TRIANGLE. We'll do an OLD TRIANGLE. And then TYPE information to the user to indicate that the program is about to be started.

```
1020 >OLD TRIANGLE
1030 >TYPE 'DATA GENERATED PROGRAM STARTED'
1040 >
```

We would now like for CALC to RUN the program. And then we would like to find out if, after all of this running, we really return to the subroutine. So we will have the subroutine type a value of a string that simply says "if this types we made it back."

```
1040 >RUN
1050 >TYPE 'IF THIS TYPES WE MADE IT BACK'
1060 >
```

Notice that none of these commands are being processed as I typed them in. The reason for that is that there are statement numbers in front of each command and they are simply being stored in the work space. The subroutine will be complete when this finishes. We will have nothing else that we need to do; so we ask it to return. Now we must go into the MANUAL mode. Remember that you need the star on the MANUAL command. We will now list the program to see what it looks like—using the PRINT ALL command.

```
1060 >RETURN
1070 >*MAN
->
->P A
50 DELETE ALL
65 C 0
100 10 QU A WHAT IS THE VALUE OF A
110 JUMP 20 IF A = 0
120 QU B WHAT IS THE VALUE OF B
130 TYPE 'THE ANSWER IS 'SQR(A**2+B**2)
131 SET C = %E%
132 GEN '%A%,%B%,%C%'
140 JUMP 10
150 RETURN
1000 20 P A
1010 SAVE TPF$.DATA
```
1020 OLD TRIANGLE
1030 TYPE 'DATA GENERATED PROGRAM STARTED'
1040 RUN
1050 TYPE 'IF THIS TYPES WE MADE IT BACK'
1060 RETURN
END OF FILE

We will now REPLACE the subroutine and CALL it.

->REP
->CALL CALC

CALC asks for the value of A, we enter 1; value of B, 2. It prints the answer and generates it in the work space. We input the values of 3 and 4. It asks for the value of A again. We will enter 0 to process it.

WHAT IS THE VALUE OF A > 0
0
1 1,2,2.368679774997897
2 3,,4,,2.368679774997897
COMPILER TYPE: BASIC, R
DATA GENERATED PROGRAM STARTED
ENTER LENGTHS OF SIDES SEPARATED BY COMMAS

Editorial Note: The second pair of values (3 and 4) were apparently accidentally followed by commas on input. This confused the subroutine which then returned the same value as before for the hypotenuse without admitting it didn't know what it was doing.

It lists the work space exactly as the data were generated; and it starts the BASIC program TRIANGLE. You will notice that it told you the compiler type of the program that was referenced by the OLD in the subroutine. In order to get the data to the program, you must now use the Exec statement:

>ADD TPF$.DATA
1,2,2.2368679774997897
A = 1 UNITS
B = 2 UNITS
C = 2.236868 UNITS
AREA = .99999995 SQUARE UNITS
3,,4,,2.2368679774997897
A = 3 UNITS
B = 4 UNITS
C = 2.236868 UNITS
AREA = 3.3166246 SQUARE UNITS
>
And as you can see the program is printing the value of the results. It comes back and requests additional information. Since we have none, we will type STOP. Notice from the TYPE statement that we made it back to the program. And indeed we came back into the input mode ready to input additional data, since this is the last thing that happened when we left the subroutine.

>STOP
STOP
  PROGRAM STOPPED.
  TIME: .048
  IF THIS TYPES WE MADE IT BACK
READY
10>

Star MANUAL will return you to the MANUAL mode. If we look the content of the work space, you will see that the program is now in the work space, and not the data that we generated.

10 >*MAN
*MAN
>LIST
LIST
100 PRINT
110 PRINT 'ENTER LENGTHS etc.
:
210 END
END OF FILE
>

As you can see CTS has many convenient features. There are others which we haven't covered, but may be useful to you. You can learn of these features, or review what you've seen demonstrated, in the UNIVAC CTS Programmers Reference Manual. Or call the people at Ft. Collins, they will be glad to help you.