The QUALOG User’s Manual (Preliminary Version)

Anne Shelly  
*Syracuse University*

Ernest Sibert  
*Syracuse University, eesibert@syr.edu*

Follow this and additional works at: [https://surface.syr.edu/eecs Techreports](https://surface.syr.edu/eecs_Techreports)

Part of the Chemistry Commons

Recommended Citation  
[https://surface.syr.edu/eecs_Techreports/37](https://surface.syr.edu/eecs_Techreports/37)

This Report is brought to you for free and open access by the College of Engineering and Computer Science at SURFACE. It has been accepted for inclusion in Electrical Engineering and Computer Science Technical Reports by an authorized administrator of SURFACE. For more information, please contact surface@syr.edu.
The QUALOG User's Manual
(Preliminary Version)

Anne Shelly
School of Education

Ernest Sibert
School of Computer and Information Science

November 1983
The QUALOG User's Manual

(Preliminary version)

Anne Shelly
School of Education

Ernest Sibert
School of Computer and Information Science

November 1983

Syracuse University
Syracuse, New York 13210
Acknowledgement

The authors wish to thank Mel Shelly for the suggestions which he has contributed to the writing of this manual.

Copyright 1983, A. Shelly and E. Sibert
Errata

The last paragraph on page 2-4 should read:

BEGIN creates ten files in your disk area. These are
SWITCH.INI, SETUP.MIC, PNUMB.LGC, PRTCPT.LGC, REMSTS.LSP, DONEXT
PNUMB.MIC, CODE.MIC, BYTEST.MIC, FINISH.MIC

The examples near the top of page 5-11 should read:
> *pcodes ^$main,.<subcode>; <any of the 4 choices listed above>
> *pcodes ^$main, eval; Cook
Contents

Chapter 1  INTRODUCTION TO QUALCG

Chapter 2  INITIATING A STUDY
  File Names
  Organizing Data Files - an Example
  Participant and Setting Labels
  Performing the Initialization
  Files Associated with a Study
  Subdividing your Disk Area

Chapter 3  PREPARING DATA FILES WITH SCS
  Titles and Page Headings
  Paragraphing
  Unformatted Material
  Special Characters
  Choosing the Page Size
  Transferring Files from Other Computer Systems

Chapter 4  FORMATTING DATA FILES WITH FNUME
  How to run FNUME
  Space Requirements for Running FNUME
  Printing Files and Disk Maintenance
  Format of the FNUME File

Chapter 5  ORGANIZING DATA WITH THE CODE PROGRAM
  THE CODING FUNCTION
    Coding Data Files
      The "c" Command - Entering Code Information
      The "rc" Command - Replacing Code Information
      The "dc" Command - Deleting Code Information
      Other Commands used in Coding
    Retrieving Coding Information
      The "fatcodes" Command
      The "pcodes" and "pcodesn" Commands
      The "wrtext" Command
      Other Commands used in Retrieving Coding Information
  THE MEMORANDUM-WRITING FUNCTION
    Writing memos
    Looking at Memo and Data Files
    Copying from Memo and Data Files
    Summary of Commands used in Writing Memos

  Ending the Coding Session
    The "stop" Command
    The "endday" Command
    The "kill" Command
CHAPTER 1

INTRODUCTION TO QUALOG

In qualitative research, data analysis is the stage of organizing, interpreting, synthesizing, conceptualizing, and reporting information collected over a period of time and by various means. The researcher organizes the information into "manageable units", searches for recurring topics and patterns of words or phrases, decides on a focus, and refines the thesis or theme so that it accounts for supporting and negative evidence (Bogdan and Biklen, 1982). The major tasks of data analysis include organizing the data, developing categories around patterns or recurring pieces of information, discovering and testing the strength of relationships among categories, and developing a theme or thesis. Using a logic programming system, we are developing and refining a collection of computer programs to facilitate the mechanical tasks of formatting, coding, and hypothesis testing which are part of the analysis stage.

Logic programming is a relatively new method of writing computer programs (Kowalski, 1974) based on earlier work in mechanized theorem-proving (Robinson, 1965). Unlike conventional programs, which specify the sequence of steps the machine should follow to obtain the desired result, logic programs specify the conditions the result should satisfy, and leave the details to the computer. The notation of logic programs is that of mathematical logic, more specifically a subset of the first-order predicate calculus, adapted somewhat for use in the computer. Logic programs are "executed" (one also says "interpreted") by a theorem-proving program which establishes the existence of a result satisfying the specified conditions, and yields the result itself as a by-product of the proof.

LOGLISP (Robinson and Sibert, 1981), the programming system in which QUALOG is written, integrates logic programming with LISP (McCarthy, 1960; McCarthy et al., 1965; Meehan, 1979), a well-known and highly regarded language intended primarily for symbolic (as opposed to numeric) computation. The combination provides a very rich and flexible programming environment which seems very well suited to the requirements of qualitative data analysis:

1. LOGLISP allows the researcher to work directly with symbolic information, including prose text, with no necessity for numeric encoding;

2. the researcher easily enters and records arbitrary information (e.g., interview transcripts, field notes, observer's memoranda, codes, line numbers) without extensive advance planning or special programming;
3. LCGLISP provides extremely flexible means for retrieving such information, for studying relationships in the information, and for testing hypotheses about the relationships.

The notation used in the LOGIC portion of LCGLISP is based on the notation of LISP. We write expressions somewhat like those found in ordinary algebra, but which may refer to words, phrases, or dates, as well as to numbers. An expression is written as a list, enclosed in parentheses, whose entries are themselves expressions. For example,

(15 Oct 83)

is an expression; it is the way we represent a date in this system. As in algebra, certain conventions are established regarding the representation of variables. Unlike algebra, which uses letters (e.g., "x" or "y") to represent variables, LCGLISP allows the researcher to establish almost any convention. We follow the rule that any name beginning with "$" is a variable. For example, "$date" stands for "any date", "$subcode" stands for "any subcode", and the expression ($d Nov 82) stands for "any day in November, 1982".

As we explain the QUALOG programs, we will use terms relating to LCGLISP specifically, or logic programming in general.

**fact** - an unconditional statement containing no variables; also referred to as a "datum"
- e.g., The number 10 follows the number 9.
- Lines 6 through 18 of this file are coded "students rdg".

**rule** - a statement which contains variables and/or is conditional
- e.g., Any number following another number is the larger number.
- One instance of coded text is adjacent to another instance of coded text if the last line of one instance immediately precedes the first line of the other.

**assertion** - a statement claimed to be true about the specified problem; facts and rules are assertions

QUALOG uses the logic programming facilities of LCGLISP in several ways. Most importantly, the coding information is stored as a collection of facts, which leads to great flexibility in retrieving this information. Thus, we can use logic variables to express "code patterns" to concisely specify subsets of codes (e.g., a particular main code with any subcode). QUALOG also keeps routine records: which files have been coded, what codes have been introduced in the
study, etc.; these all are stored as a collection of facts.

We use the deductive capability of LOGLISP:

1. to select relevant cases in various operations (e.g., to determine which main codes appear most frequently across a variety of data, to replace occurrences of a main code when it appears with one subcode, but not when it appears with others),

2. to investigate relationships involving coding information, the data itself, and information associated with the data files,

3. to examine formally stated hypotheses about these relationships.

In qualitative analysis, the researcher spends many hours interpreting data, conceptualizing and synthesizing the important issues, and subsequently developing the major focus or theme. The mechanical tasks of coding data, of sorting or retrieving information by categories, and of scanning text for word or phrase patterns account for many of the hours spent in analysis. QUALOG is the set of computer programs we are using and developing which allows the researcher to work through these mechanical tasks more thoroughly and faster, thus enhancing the conceptual work which is the central concern of the researcher.

QUALOG is perhaps best viewed as a highly sophisticated, mechanical assistant. It can record large quantities of information and can retrieve that information in a variety of formats. It will not perform the conceptual work of the researcher. QUALOG will not make decisions, it will not determine the codes to be associated with the data, it will not identify relationships between codes. It will record decisions, record associations of codes with data, and test relationships against the coding information and the data. QUALOG cannot establish working hypotheses to explain particular aspects of the data, but if the researcher develops a formal statement of such hypotheses, QUALOG can exhaustively search the coding information and the data for confirming and disconfirming instances.

This is a preliminary manual describing QUALOG as currently implemented at Syracuse University, and includes some details which are specific to the S.U. Computer Center. Some of the programs discussed here are still being refined; one major program is still being developed and is not described at all.
Since this is a reference manual, we suggest that you read through the entire manual before studying particular sections. We assume that the reader has a basic acquaintance with the DEC-10 computer, on which QUALOG runs, but we explain several of the DEC-10 monitor commands most likely to be used with QUALOG.
CHAPTER 2

INITIATING A STUDY

Before you start putting data into the computer, you need to establish a few conventions about names for data files and to decide about certain formatting issues, then tell the QUALOG programs about your choices. The programs are quite flexible, but you must be reasonably consistent, and some matters have to be settled before you start using the programs.

Throughout the manual, we will be giving you detailed directions about talking to the QUALOG programs. Be sure to follow our command formats closely. Take note of the convention we are using to give you directions for writing commands; we use it consistently throughout this manual. When we show you generalized examples, we will use angle brackets (< >) to enclose phrases that describe the information you should type. When we give specific examples, we will not use the angle brackets. Rare exceptions to this convention may occur, such as typing protection codes (e.g., <477>); in this case, you should type the angle brackets and what's inside them.

2.1 FILE NAMES

A file name in the DEC-10 computer has the form

<file name>.<extension>

where <file name> is any sequence of one to six letters or digits, and <extension> is any sequence of one to three letters or digits. Examples are COOK.T1, CODE.LSP, FNUMF.MIC, CUMFGS.I2.

Your qualitative data will be stored in the computer as a collection of files. In a typical study, a file might contain the transcript of one interview, one day's field notes, or some other appropriate piece of text. A file can contain any reasonable amount of data, from a few pages to perhaps two hundred pages of (single-spaced) text. Amounts larger than this should be divided into two or more files. There will, in fact, be two versions of each data file -- an unformatted file which you prepare with the editor SCS, and a formatted file with special line numbers which will be used for coding and analysis. We further organize the files by grouping them according to participants (or objects) of the study, and also according to the nature of the data (interview, observation notes, etc.). The names chosen for files will reflect this organization.
2.2 ORGANIZING DATA FILES - AN EXAMPLE

To illustrate we shall use a small demonstration study of secondary school teachers and their personal models of reading. The study involves two participants, Jim Cock and Jill Cummings. For each participant we choose a file name which will be used for all data files for that participant. Having six characters at most, we use COCK and CUMMGS for file names. In this study, the data are of two types: interview transcripts and observations. The (unformatted) SOS files for interviews we give extensions T1, T2, ..., while the corresponding formatted files will have extensions I1, I2, ... (think of talk and interview). Similarly we use O1, O2, ... for the unformatted observations and CO, C2, ... for the formatted observations (lock and observation). The first letter of the extension for the formatted file has special significance; it identifies the type of file for the programs. The programs determine the number of title lines and the number of heading lines (on the second and subsequent pages) from this letter. You can use any letters except C and Q, and can specify either two or three lines for titles on the first page of the file. Normally the second and subsequent pages have one or two heading lines, with the page number at the upper right and one or two running heads at the upper left, though the programs can accommodate files with no headings, or with three or more heading lines. In any case, you choose the identifying letters, decide about titles and headings, then inform the programs about these choices as part of the study initialization.

2.3 PARTICIPANT AND SETTING LABELS

The first title line of a data file in this particular study will include the name of the participant and the setting in which that participant was observed, in this case a school. To illustrate, the COCK files will all begin with

Jim Cock - Newtown High School

The programs will print this information in various situations, and we wish to use labels adapted to this study, like "teacher" and "school". Consequently, you also get to specify participant and setting labels as part of study initialization. In another study you might want "employee" and "division", or "document" and "author".
INITIATING A STUDY

2.4 PERFORMING THE INITIALIZATION

You should begin your work in a disk area containing no files from another QUALOG study. We shall explain later how to subdivide an area to store files from a number of studies. To begin, give the command

./DSKB:BEGIN[733,21,QUAL]

This will start a program which does two things: 1) it creates various files in your disk area, 2) it asks you for a participant label, for a setting label, and for format information for your data files. Since BEGIN is used only once per study, it types fairly detailed instructions as it proceeds. What follows is a complete example showing how we used BEGIN to initialize a study. The parts you type are underlined here; they won't be underlined when you use the program.

BEGIN-STUDY)

QUALOG

Study Initialization

Specify the words you want to use to label the participants (or objects) of the study and their settings.

Example:
Participant label: Teacher
Setting label: School

Participant label: Teacher
Setting label: School

How many types of data files will you have? 2

What are these types (e.g. interview)?
File type 1: interview
File type 2: observation

Data-file types are identified by the first letter of the file-name extension, which must be the same for every file of a particular type, e.g., interview files might all have names like ffffff.Ixx. This may be any letter except C or Q. What letters will you use to identify data-file types?

First letter of extension for interview files: 1
First letter of extension for observation files:  C

Files of the same type should be consistent in having the same number of title lines on the first page and the same number of heading lines on the second and subsequent pages. The number of title lines must be either 2 or 3. The number of heading lines may be 0 or more, but is normally either 1 or 2.

How many title lines for interview files?  3
How many heading lines for interview files?  2

How many title lines for observation files?  3
How many heading lines for observation files?  2

Summary of data-file types:

<table>
<thead>
<tr>
<th>Type</th>
<th>Identifying letter</th>
<th>Title lines</th>
<th>Heading lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. interview</td>
<td>I</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2. observation</td>
<td>C</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Correct (y or n)?  y

If you answer "n" instead of "y" the program will ask whether you wish to add a type, change an entry, or delete a type. Respond with one of the letters "A", "C", or "D", then answer the question asking which line you wish to change by typing the line number. If you are adding or changing an entry, the program will then ask you to (re)type the line in question. You respond by typing, e.g.,

observation C 3 1

if you want to change line 2 to specify one heading line for observation files.

At this point your study is initialized and you may proceed.

2.5 FILES ASSOCIATED WITH A STUDY

BEGIN creates nine files in your disk area. These are

SWITCH.INI, SETUP.MIC, FNUMB.LGC, PRICPT.LGC, MESTST.LSF, DONEXT
FNUME.MIC, CODE.MIC, HYTEST.MIC
The first two contain information for the DEC-10 monitor which gives you easy access to the QUALOG programs (and ICGLISE), and some information about your terminal. The next three files have information about your study which will be updated as you enter coding information, write memoranda, etc. DCNEXT is needed for technical reasons. The remaining files tell the computer how to run the corresponding QUALOG programs. These files can ordinarily be left just as they are, but in a few cases you might wish to alter the terminal settings in SETUP.MIC.

If you want to change SETUP.MIC, you first need to change its protection code. Give the monitor command

```
.PBNT SETUP.MIC<455>
```
typed just as shown, including the angle brackets. Now you can use SOS to edit the file. The terminal settings appear in line 100:

```
00100  .TTY LC NG GAG NC ALTMOD PAGE WIDTH 80 NG BTCCMP
```

The only things you should change are "NO GAG" and "WIDTH 80". "NO GAG" allows other users to send messages to you (using the monitor SEND command) while you're working on the computer. If you don't want to allow such messages, take out the "NO" before "GAG". "WIDTH 80" says that your terminal can print 80 characters on each line. If you normally work on a hard-copy terminal with wide "computer paper", change 80 to 132. The rest of the file should be left as is. After you have edited SETUP.MIC, change the protection back to <655> with another "PBNT" command.

### 2.6 Subdividing Your Disk Area

If you want to conduct several studies at one time in the same computer account you need to divide your disk area using what are called "sub-file directories" in the DEC-10 system. The files for a particular study, then, will all be in the same "sub-area", and will not be confused with files for another study. To create a sub-file directory, you first pick a name (a file name, actually) for the sub-file directory, then follow the steps shown below. For this example we're using STUDY2 as the name of the sub-file directory. Again, the part you type is underlined here; it won't be when you do this on the computer.

```
.SCS STUDY2.SFD
Input: STUDY2.SFD
00100  $ {press the "escape" key}
```
*FS
[DSKG:STUDY2.SFD]

.PROT STUDY2.SFD<755>
Files renamed:
DSKG:STUDY2.SFD

.R SFTSRC

*CF [STUDY2] /NOSCAN

*C {hold down "control", press "C"}

If your account number is, say, [2233,12], then at this point any files you create go into the "sub-area" [2233,12,STUDY2]. You can initialize the study and begin to create data files, then proceed with coding as though this were an entirely separate account. To switch from one sub-area to another follow the steps beginning with "R SFTSRC". To log in directly to a sub-area use

.LLOGIN [2233,12,STUDY2]

The brackets are necessary in this case.
CHAPTER 3
PREPARING DATA FILES WITH SOS

SOS is a widely used file editor for the DEC-10, the one most often used to prepare data files for analysis with QUALCG. For details regarding the use of SOS you should consult the reference manual (SOS manual). In this chapter we shall explain the common ways of formatting data files and some of the special considerations needed to be sure that the formatting program, RUNOFF, produces properly formatted output. For further details about the formatting commands for RUNOFF see the manual (RUNOFF manual) and related papers.

3.1 TITLES AND PAGE HEADINGS

We usually arrange data files with three title lines on the first page and two header lines on each subsequent page. The commands to accomplish this are:

00100 .ap .fig 3 .ps 58.65
00200 .c:Participant Name - Setting
00300 .c:Interviewer _ #1 - Interview date
00400 .c:Interviewer - Interviewer Name
00500 .title Participant Name - Setting
00600 .subtitle Interview _ #1 - Interview date

Any line beginning with a period is a command line for RUNOFF, not part of your data. The line numbers provided by SOS are used only to edit this original file. They have no particular relation to the line numbers which will later appear in the formatted data file, and which are used to reference the text for coding and analysis. In the first line, the command ".ap" tells RUNOFF to use its "automatic paragraphing" mode, ".fig 3" skips three lines at the top of the page (this is optional), while ".ps 58.65" specifies a page size of 58 lines (standard) by 65 characters per line. We shall say more about these shortly.

The next three lines give the three title lines, each of which is centered on the page, as indicated by ".c:" (no space after the semi-colon). The first two title lines are used by the programs and must have the format indicated. For the first line, a name, a hyphen surrounded by spaces, then another name. The second line has one word, a space, the characters ".: #" followed immediately by a number (perhaps two or more digits), a hyphen set off with spaces, then a date in the form: November 28, 1983. The second half of lines 200 and 300 can be omitted if you wish, in which case you should also omit
the hyphen. The third title line can have any form you like, it will simply be reproduced by the programs.

The lines numbered 00500 and 00600 specify the headings to be printed at the upper left of the second and subsequent pages. The first heading is given by the command "title ...", the second line by "subtitle ...". There should be a space after "title" and after "subtitle". RUNOFF will put the page number at the upper right of each page (e.g. "Page 2"). We customarily repeat the first and second title lines, but you could do something different, or omit the headings entirely. Note that in any case there will be at least one heading line, which will be the page numbers at the upper right corner.

You may prefer to start the three title lines at the left margin, rather than centering them. If you want this arrangement, start your file like the following example:

00100 .ap.fiq 3.ps 58,65.nf
00200 Participant Name - Setting
00300 Interview _#1 - Interview date
00400 Interviewer - Interviewer Name
00500 .title Participant Name - Setting
00600 .subtitle Interview _#1 - Interview date
00700 .f

Either form is acceptable to the QUALCG programs.

3.2 PARAGRAPHING

".ap" is the "auto-parapraphing" command. This command will give you conventional paragraphs, indented five spaces and separated by one blank line. Indicate the beginning of a paragraph by starting a new line, then indenting with one or more spaces, or one "tab". Don't type a blank line in the SOS file. Be careful not to start a line with a space or tab unless you really mean to start a new paragraph.

When typing interviews you will want a different format for paragraphs, with the interviewer's (or participant's) name projecting to the left of the text at the beginning of each paragraph. To accomplish this, add one more command line just before you begin the text:

00700 .lm 5.p -5
This sets the left margin five spaces in from the left edge of the paper, and sets the paragraph "indent" to five spaces left of the margin. We still indent in typing the SOS file, starting each paragraph with the appropriate name, followed by a colon. To illustrate, an SCS file with

```
00800 Ernie: Isn't it a little odd to type the
00500   file this way, when what we want is quite different?
01000 Anne: Perhaps, but remember that this file is
01100   really just a large set of instructions to RUNCPP,
01200   consisting mainly of text. What you see in the SOS
01300   file is not exactly what you get in the end.
01400   What you get is better.
```

When formatted this would produce:

```
Ernie: Isn't it a little odd to type the file this way, when what we want is quite different?

Anne: Perhaps, but remember that this file is really just a large set of instructions to RUNCPP, consisting mainly of text. What you see in the SOS file is not exactly what you get in the end. What you get is better.
```

Sometimes you might want to break a long response into two or more paragraphs. Don't indent in the SCS file, you'll get a strange looking paragraph with its first line projecting to the left. Instead, insert a command ".s 1" or a line by itself, then start typing the new paragraph on the next line without indenting. The ".s 1" tells RUNOFF to skip one line, which makes a very reasonable "internal" paragraph.

You can do fancier things for such paragraphs if you think it worth the bother. To illustrate, a command line

```
02900 .s 1.i 5.tp 4
```

would skip one line, indent the next line five spaces (to the right), and test the number of lines left on the page, moving to the next page if this were not at least four lines.
3.3 UNFORMATTED MATERIAL

The techniques outlined so far should be adequate for interview transcripts and for most ordinary text. Notice that the programs make the decisions about where to start a new line, and insert extra spaces to justify the lines. In some cases you may not want the computer to do this sort of thing. To prevent RUNOFF from formatting the text, insert a command line

```
03600 .nf
```

This "no fill" command tells RUNOFF to reproduce subsequent lines, except command lines, just the way they are typed. RUNOFF still recognizes commands after ".nf", so if you want to skip a line in the data insert a command ".s 1" (".s 2" for two lines, etc.). A blank line in the SOS file would just be ignored. If you type a line which is too long to fit in the formatted file (determined by the ".ps" command at the beginning) RUNOFF will break the overlength line into two lines in the final file, which is usually the best thing to do.

To switch back to the normal formatting, insert a command line

```
10700 .f
```

This "fill" command restores filling and justifying of lines in the usual manner. Paragraphs will be treated just as they were before the ".nf" command.

3.4 SPECIAL CHARACTERS

RUNOFF attaches special significance to a few characters when they appear in the text, as opposed to command lines. These characters are

```
_ & ^ \ # : .
```

You can ignore the (slightly) special nature of the period and colon most of the time.

To begin with, the underscore character tells RUNOFF to ignore the special meaning of the immediately following character. If, for example, you want the character "#" to appear in your file, type "._#" in the SOS file. This is just what we did in the second title line at the beginning of the file. If you want an underscore character to appear in the file (by itself, not underneath another character) type "__".
If you want a portion of the text underlined in the final data file instruct RUNOFF to do this as shown in the following example:

11600 The collection "Literary Essays," by Mark Twain, 11700 is not only vastly entertaining but has some 11800 excellent advice for writers. "Penimore Cooper's 11900 Literary Offenses," with its famous (should we say 12000 notorious?) "nineteen rules governing literary art," 12100 is a masterpiece of the "genre."  

The pair "" tells RUNOFF to start underlining, while "" tells it to stop. Spaces between words will not be underlined, but notice that we were careful to stop underlining before the punctuation marks. When formatted this produces

The collection Literary Essays, by Mark Twain, is not only vastly entertaining but has some excellent advice for writers. "Penimore Cooper's Literary Offenses," with its famous (should we say notorious?) "nineteen rules governing literary art," is a masterpiece of the genre.

Underlining should probably be used sparingly. It works fine when the files are printed, but many video terminals cannot display underlined text properly.

The sharp sign "#" tells RUNOFF to insert a space, but that this space is not to be taken as a point at which to break the line, nor can it be expanded with additional spaces in order to justify the line. Thus, if you type "Prof.#Jones" in your SOS file, the title and name will always appear on the same line, with exactly one space after the period.

Period and colon are special in that RUNOFF inserts an extra space after these when they occur at the end of a word. In the middle of a word, say "$195.99", they are treated like ordinary characters. You can suppress this special treatment using the underscore character. To illustrate, typing "Ms._Smith" in the SOS file will produce "Ms. Smith" in the formatted file with no extra space, though "Smith" might turn out to be the first word of a line. Use the underscore also if you want a line of text to begin with a period, so the line will not be taken as commands for RUNOFF.
3.5 CHOOSING THE PAGE SIZE

The command ".ps 58,65" we put at the beginning of the SGS file specifies a page size of 58 lines by 65 spaces per line. Using 58 lines, the standard value, for headings and text leaves four lines each (2/3") for top and bottom margins, which is usually quite satisfactory. The choice of 65 characters per line is also quite reasonable, but several considerations might lead you to select a different value.

The points which influence the choice of line length are

1. The FNUMB program will add four-digit line numbers separated from the text by one space, so the lines in the final data file will be five characters longer than the number given in the ".ps" command.

2. Most printers print ten characters per inch. If you use paper 8-1/2" wide you can have up to about 80 characters (total) per line. With 14-1/2" "computer paper" you can have 132 characters per line.

3. Most video terminals display 24 lines of 80 characters each on the screen. Unfortunately, many of these terminals will skip a line after displaying a line that uses all 80 characters, so you can use only 79 characters per line.

Since we are using the computer to keep track of the coding information, we print the data single-spaced with regular margins. We end up using less paper, and, perhaps more importantly, we display more data on the terminal screen at one time. You may, on the other hand, want to do your initial coding by marking up a printed copy of the data file, so you'll want wider margins.

We recommend a line length for the ".ps" command ranging between 60 and 74 characters per line. If you are preparing data files entirely in the "no fill" mode you probably should use 74, even though most lines will not be so long. This way the lines come out the way you typed them if at all possible.
3.6 TRANSFERRING FILES FROM OTHER COMPUTER SYSTEMS

Data files prepared on another computer system must be moved to the DEC-10 if you want to use them with QUALC. Such files will almost certainly require some editing (with SOS) to incorporate title lines and formatting commands. Files are usually transferred between computers on magnetic tape, but other media can be used.

If you are moving files from another DEC-10 use a tape written by the BACKUP program (BACKUP manual) in INTERCHANGE mode. Files prepared under CMS can be transferred on standard IBM tapes read by CHANGE on the DEC-10. Consult the AID office at the S.U. Computing Center for assistance. The AID office can also help if you have files on floppy disks from a microcomputer.

Once you have transferred files to the DEC-10, the following steps will ensure that they are in a suitable form for QUALC. We illustrate with a file named TCHR.T1. You type the parts that are underlined, the machine does the rest.

```
.SCS TCHR.T1
Edit: DSKG:TCHR.T1
*PS

[DSKG:TCHR.T1]

.COPY = TCHR.T1/C

.SCS
Edit: DSKG:TCHR.T1
*i10;10
00010 .ap-fig 3.ps 58,65
00020 .c:Teacher Name - School Name
00030 .c:Interview #1 - Interview date
00040 .c:Interviewer = Name
00050 .title Teacher Name - School Name
00060 .subtitle Interview #1 - Interview date
00070 $ (press "escape" key)  
*P

[DSKG:TCHR.T1]
```

The first three lines in this example constitute an edit of your data file. This editing session removes the SOS line numbers (ES stands for "exit, strip"). Next is the COPY command, which deletes trailing blanks (switch "/C"). Files written on tape from CMS usually
have extra blanks so that every line has exactly 80 characters. These are just a nuisance on the DEC-10, so we get rid of them. The final editing session brings the file into the form required for the QUALLOG programs. In the example above, we show you the lines giving command and heading information; you can any other changes in the file at this time, too. After this, you leave the SCS line numbers on, in case further editing is necessary.
CHAPTER 4

FORMATTING DATA FILES WITH FNUMB

FNUMB is a program which combines operations from RUNOFF and from LOGLISP. Using RUNOFF, the SOS data file is formatted; SOS line numbers are removed and formatting commands are implemented. Using LOGLISP, the formatted data file is numbered with four-digit line numbers. The program numbers each line of text, beginning with number 0001 for the first line and numbering each subsequent line of the formatted file (excluding headings) to the end of the data file. As a result of this program, you can easily reference any section of text by specifying the file identification (e.g., COOK.T1) and the line numbers.

4.1 HOW TO RUN FNUMB

To run FNUMB, you type the following command at monitor level:

```
./FNUMB <file name>,<SOS file extension>,<formatted file extension>
```

For example, if I want to run my SOS data file, COOK.T1, through FNUMB, I type the command:

```
./FNUMB COOK.T1,II
```

I'm telling the program to take the file COOK.T1, to format the file, to attach the four-digit line numbers to the line of the text, and to call this new formatted file COOK.II.

Note the difference between the SOS file and the FNUMB file for the COOK data. The SOS file, COOK.T1, begins:

```
00100 .Fig 6.ps 58,65
00200 .c;Jim Cook - Newtown High School
00300 .c;Interview _#1 - March 5, 1982
00400 .c;Interviewer - Anne Shelly
00500 .title Jim Cook - Newtown High School
00600 .subtitle Interview _#1 - March 5, 1982
00700 .lm 5.ap.p -5
00800 .s 4
```
Anne: Tell me what your schedule is this morning.
Jim: Okay, I have American history non-regents class of what we consider average kids coming up.
Anne: What grade?
Jim: Eleventh grade. Then I go to another 11th grade American history, regents level class, right after that. Then I have a brief thirty minute break - we have some overlapping periods. Then I go to a psychology class which is a senior elective and that's followed by another American history regents class grade eleven. So all I have is grade 11 and one 12th grade elective in psychology.

The FNUMB file, COOK.11, begins:

Jim Cook - Newtown High School
Interview #1 - March 5, 1982
Interviewer - Anne Shelly

Anne: Tell me what your schedule is this morning.
Jim: Okay, I have American history non-regents class of what we consider average kids coming up.
Anne: What grade?
Jim: Eleventh grade. Then I go to another 11th grade American history, regents level class, right after that. Then I have a brief thirty minute break - we have some overlapping periods. Then I go to a psychology class which is a senior elective and that's followed by another American history regents class grade eleven. So all I have is grade 11 and one 12th grade elective in psychology.

You may notice, as you scan through your new formatted file, that there are errors in the text (e.g., spelling, typos) or in the format (e.g., blank lines where you don't want them, no spaces where you do want them) that may make the file difficult to use. If it's important to you to go back and make changes, you must do so on the original data file (the SOS file). As you may recall, when you add to or change your SOS file, the machine creates a back-up file, which is the
file as it was before you made the additions and/or changes. After you have made the necessary changes, simply run this edited SGS file through PNUMB, using the same extension as you used previously. The file will be formatted and line numbered, and will automatically replace the old formatted file.

4.2 SPACE REQUIREMENTS FOR RUNNING PNUMB

Before running the PNUMB program on any of your SGS files, be sure you have available in your disk area at least three times the number of blocks of the SGS file to be run. As PNUMB runs, it creates and/or uses several different files: the original SGS file, a RUNOFF file with no line numbers, a RUNOFF file with line numbers, and one additional file with headers and trailers. If, as the PNUMB program is running, there is not enough block space to create these files, the program will crash, resulting in no formatted, line-numbered file.

There are three ways to avoid this situation; two require you to anticipate the problem and to act accordingly ahead of the situation, and one requires no pre-planning, but can have undesirable effects.

1. The least-favored, short-term approach is checking your block space reserves and dumping files in your disk area if you need more blocks than you have available. To check your block space reserves, type the command (at monitor level):

   .r quosl

   You'll get a message back that looks like this:

   User: 3313,11
   str used left:(in) (out) (sys)
   DSKA 1668 2332 2332 92146

   This shows that you are using 1668 blocks and that you have 2332 blocks available. To find out the number of blocks used by the SGS file to be formatted, type the command (at monitor level):

   .dir <file name>.<file extension> [e.g., .dir COOK.T1]

   You'll get the message (for example):

   COOK T1 65 <477> 23-SEE-82
This shows that the Cook file is using 65 blocks. Multiply that by three and figure that the Cook file will take a little less than 200 blocks to run through the FNUMB program. As you can see above, you have well over that amount left in your disk area, so the program should run with no problems due to space requirements. However, if you are short of block space, you will need to delete some files in order to run the program successfully. Unless you have a lot of back-up files you don't mind dumping, this can be a fairly painful experience, hence the reason for dubbing this the least-favored method of dealing with the space issue.

2. If you anticipate potential block space problems, you can (and probably should for long-term storage reasons anyway) purchase a magnetic tape at the Computer Center. If you find yourself short of block space, you can transfer files to the tape (refer to Section 2.3 below for directions or refer to the BACKUP manual), then delete them from your disk area. This is a less painful experience than in the first method because you can always retrieve the deleted files from your tape at a later time.

3. A second way to anticipate this issue is to call the computer accounts office for a "logged in" quota with extra blocks. This way you have extra block space available while you are logged in your account, but your actual block storage space remains the same. For example,

```
User: 377,22
Str used left:(in) (out) (sys)
DSKC 3804 6196 1196 91990
```

This person is using (i.e., storing data on) 3804 blocks, has 1196 blocks available (i.e., at the time s/he logs out, s/he can use only 1196 more blocks), and has 6196 blocks available during the time s/he is logged in. As you can see, with this method, you don't need to worry about availability of block space while you're working. The caution here is simple; you will still have to delete files if, as you log out, you have used more than (in this example) 1196 blocks. Thus, it's probably a good idea to have the magnetic tape (described in #2 above) as a back-up to this method.
4.3 PRINTING FILES AND DISK MAINTENANCE

After FNUMB creates a formatted, line-numbered file with which you can work, the SOS files are no longer important. You can move them off your disk to tape through BACKUP (see BACKUP reference manual for more detail), by typing the following commands:

```
.MOUNT MTB: BKUF/REEL:<your reel number>/VID:<"your name - time estimate">/WENAEI
```
(For example, if Anne types this command to mount her tape, it looks like this: .MOUNT MTB: BKUP/REEL:135560/VID:"SHELBY-15MIN"/WENAEI)
(You will get a "...waiting" message.)
```
^C, ^C (Type C twice while holding down the "control" key.)
```
```
.MOUNT/C
```
(You will get a message telling you whether or not your request to have the tape mounted was received. When your tape is mounted, you will receive another message telling you so.)
```
.R BACKUP
.TAPE BKUP
.DENSITY 6250
.NCINTERCHANGE
.REWIND
.EXT
.SSNAME <type the name, usually the date, under which this batch of files will be saved>
```
```
SAVE
(The machine will type a message back to you, pause because it's in the process of copying your files from disk to tape, then will type DONE to indicate that the "save" is complete.)
```
```
.REWIND
.PFINT DSK:<file name>..<file extension> [e.g., TAPF:LOG] (Use this command only if you wish to print a list of files on the tape.)
```
```
.REWIND
.UNLOAD
.EXIT
```
```
.DISMOUNT BKUP/REMOVE
```

After you have copied all the files on your disk to tape, you can delete any files you do not anticipate using.

To print copies of your formatted files, you need to make several decisions:

1. number of copies - if you don't specify, you'll get one; if you do, you'll need to specify how many; e.g., copies: 3
2. type of paper - if you don’t specify, you’ll get form 0, which is the 8-1/2” x 15”, off-white, green-lined computer paper. If you do, you might specify form 15, which is 8-1/2” x 11”, plain white. For form options, type the command HELP OFORMS at monitor level; e.g., forms:15

3. limit - if you want to reduce the wait time in the queue, you may want to specify a page limit. You do this by checking the length of the file (RUNOFF will tell you how many pages the file is as the FNUMB program runs), then adding enough pages to cover the number of copies you want plus another five to ten pages to cover headers and trailers. If you don’t specify a limit, the computer will do so automatically, but it figures high, so you may have a longer wait for your printout.

As an example, I want to print two copies of the formatted Cook file. RUNOFF indicated the file was 16 pages long, so I’ll type the command:

.print COOK.11/forms:15/copies:2/limit:40

Notice that there are no spaces after I start typing the file and the specifications.

4.4 FORMAT OF THE FNUMB FILE

1. The first page of your printout will have the file name in large letters/numbers. Below the file name is computer system information including the command you gave to print the file.

2. The second page of the printout will have one number in the upper left corner; the rest of the page will be blank. This number indicates the number of characters in your data file. This number makes it easier for the CODE program to find specified pieces of text more quickly.

3. The third page of the printout will be the first page of your data. The two- or three-line title you specified in the SCS file will be centered at the top of the page, followed by the body of your text. The four-digit line numbers run down the left side of the page.
4. Subsequent pages contain the text. At the top of each page will be page numbers and the two-line heading you specified in the S0S file. As on the first page of text (the third page of the file), the four-digit line numbers run down the left side of the page.

5. The page following the text begins with the line (PROCEDURE Txpq). Below this statement you will see a list of LCGLISP assertions. They indicate the position in the file at which every fiftieth line of text begins. As noted in #2 above, this accounting makes it easier for the CCDE program to find specified pieces of text more quickly.

6. The last page of the printout contains computer system information.
CHAPTER 5

ORGANIZING DATA WITH THE CODE PROGRAM

CODE is a set of LOGLISP programs designed to sort, organize, and display data in a variety of ways. There are two major components of CODE: the coding function and the memorandum-writing function. In this chapter, the coding function is divided into two sections: coding data files and retrieving coding information.

THE CODING FUNCTION

The coding function allows you to assign codes (i.e., main codes with subcode divisions, if desired) to text sections of the data files. In this system, a code is an arbitrary prose phrase, usually fairly short, which suggests the actual coding category. The program automatically records data file information such as participant name, type of data file (e.g., interview), date on the file, location of data collection, researcher, and number of times you have worked on coding this particular file. The program also automatically creates, then maintains two files per participant in which the coding information is stored: <participant id>.COD contains all the coded text information, and <participant id>.CDB contains data file reference information. Through a series of commands, you can access this coding information in a variety of formats from a single data file, from all files for one participant, or across all data files (i.e., the entire study).

5.1 CODING DATA FILES

To begin entering code information, type the command (at monitor level):

```
./CODE
```

It may take several minutes before the program begins, depending on the amount of computer use at that time. Several messages will go by on the screen; you need to wait until you see the message:

Session date: 12 Nov 82 (or whatever the date is)

File to be coded*
You type the name and extension (separated by a hyphen) of the file you wish to code (e.g., COOK-I1). The machine will type back a file description, which is the information about the file contained in the first two or three lines of the file (i.e., the title lines). For example, if I type COOK-I1, the machine will type back:

New teacher: COOK

File id: COOK-I1
Teacher: Jim Cook
Type: Interview No. 1

File name: COOK.I1
School: Newtown High School
Date: March 5, 1982
Interviewer - Anne Shelly

Notice in this example that the program is describing this participant (in this case, a teacher) as new; the CODE program has not seen this name before. At this point, the program creates two new files in the disk area: COOK.CDR, which contains the file description information, including a record of the number of times you have worked on coding particular files, and COOK.CCD, which contains all the text and code assignments. The next time this file is entered in the CODE program, the file description will have a slightly different format:

File id: COOK-I1
Coding session 2
Teacher: Jim Cook
Type: Interview No. 1

File name: COOK.I1
School: Newtown High School
Date: March 5, 1982
Interviewer - Anne Shelly

In either case, after you get the file description information, you will get the prompt (>). This means the program is ready to record coding information about this data file. In future discussions in this manual, we will refer to this data file as "the current data file".

5.1.1 The "c" Command - Entering Code Information

To enter coded text information, you will use the "code" command. This command is designated simply by the letter "c". The form of the command is as follows:

>*>c <first line of text> <last line of text> <coding category>

Be sure to put a space between the "c" and the first line, between the first and last line of text, and between the last line of text and the coding category. You may use subcode divisions in addition to the "main" code. If you choose to use subcode divisions, you separate
them from the main code by a comma. Thus, in the example:

*>c 14 56 reading activity, oral

I am telling the machine to assign lines 14 to 56 of this file to the coding category "reading activity", which has a subdivision of "oral reading". If you want to code one piece of text multiply, you can do so on the same line; simply separate the codes by a semi-colon. Thus, in the example:

*>c 14 56 reading activity, oral; reading materials, text

I am telling the machine to assign lines 14 to 56 of this file the coding categories "reading activity" with the subdivision of "oral reading" and "reading materials" with the subdivision of "textbook".

You may use complete words/phrases or abbreviations of words/phrases as your codes. The CODE program does not constrain your use of letters, hyphens, or anything between quotation marks. You can use digits if they are not the first character in the word or the first character of any word in the phrase. So any of the following examples are acceptable:

day 2
VJ-Day
don't know
tricks of the trade
student says "Why me?"
ten-14-82
Oct-14-82

As you enter coding information, the program will be keeping track of the codes. If you enter a code the program has not seen before, it will ask the question:

New main code: <code you entered>, correct? (y or n) *

You must answer one of five ways:

1) "y" for yes, and the program will ask you to type a six-letter/digit file id. You'll need to reduce your code to this abbreviation, and you will access your code memo files (explained in the section below on memos) with it. Once you enter this abbreviation, the program records this code and will recognize it in the future. You do not use this abbreviation to enter codes with the "c" command; this is a file name and you'll use it only to access the memo file. When you use this code again, be sure to use the same words/phrases so the program won't think you're using a new code.
2) "m" for no, and the program will ask you to type the correct code.

3) "codes", and the program will type a list of the codes you have already used, with their file ids.

4) "help", for "I'm not sure what I'm doing anymore and I want to check my options", and the program will type a list of the commands.

5) "stop", for "I want out of this", and the program will forget about this coding statement and give you a new prompt (>*).

You will go through this procedure for main codes and for subcodes, if you use them. When you type "codes" in response to a question about subcodes, the program will list the subcodes you have used. You will not be asked to reduce subcodes to a six letter/digit abbreviation; that applies only to main codes (see explanation in the memo section below).

5.1.2 The "rc" Command - Replacing Code Information

To replace coded text information, you will use the "replace code" command. This command is designated by "rc". The form of the command is as follows:

>rc <first line> <last line> <coding category>; <new category>

or

>rc <first line> <last line> <coding category>; 
    <new first line> <new last line> <new category>

Thus, in the example:

>rc 14 56 reading activity, oral; reading rationale, oral

I am telling the program to replace the coding category "reading activity, oral" for lines 14 through 56 by the category "reading rationale, oral". The command

>rc 75 85 tchr expectation; 61 85 tchr expectation

tells the program to change the code "tchr expectation" for lines 75 through 85 to be the code for lines 61 through 85 instead.
You may want to replace a code every place you have used it. Use the command:

\[ \text{rc } \$a \ $b \ <\text{coding category}>; <\text{new category}> \]

The variables "$a" and "$b" are LGLISP variables which stand for any line numbers. With this command, the program will change every instance where the old coding category was used in the current file, regardless of the line numbers. "rc" works only on the current file, so you will have to enter each file, and use this command.

With any of these forms of the "rc" command, the program types back a message indicating how many items have been changed. You can keep track of when changes were made by using the "pcodesn" command (see explanation in the section below on retrieving coding information).

If you introduce a new code with the "rc" command, the program will ask the same questions as noted in the "c" command section above. As in the "c" command, after you confirm the code and give the six-letter/digit abbreviation, the program will record the new code.

If you are replacing codes with new codes, you may decide you've made a mistake. You can stop the procedure by typing "stop" or "kill" when you are asked to confirm the new code. "Stop" allows you to end a single-instance change; "kill" allows you to end a multiple-instance change.

5.1.3 The "dc" Command - Deleting Code Information

To delete code information, you will use the "delete" command. This command is designated by "dc". The form of the command is as follows:

\[ \text{dc } <\text{first line}> <\text{last line}> <\text{coding category}> \]

Be careful with this command; the coded instances really will be deleted, and there will be no record that will allow you to undo the deletion. You can delete all coded instances in the current file by a command similar to the "rc" command:

\[ \text{dc } \$a \ $b \ <\text{coding category}> \]

Be even more careful with this command; you will wipe out every coded instance of a particular coding category in the current file.
5.1.4 Other Commands Used In Coding

Any of these commands may be given at the prompt (>):

codes types a list of all the main codes for the current data file

endday stops the coding session, permanently records the coding information from that session; used to terminate the last session of the day

file types the description of the current file

h types a list of the commands you can use in the CODE program; you also get the list if you type "help"

h <command> [e.g., h codes] types a brief description of the specified command

kill terminates the coding session without recording information entered since the start of the session OR since the last "next" or "w" command; this may be a way to recover from an otherwise deadly decision to use the "dc" command. This command puts you back into LOGLISP; type "C, then "X to return to monitor level

main codes types a list of main codes and their file ids

next stops coding the current file and asks you to identify the next file to be coded

p If you just type "p", the program will print the text of the next 12 lines from the current data file. If you specify the lines you want printed [e.g. p 15 20], it will print the specified lines. If you type one line number it will print just that line from the data file. If you type, for example, p 3!6, it will print six lines of text starting from line three.

stop stops the coding session temporarily, assuming you are coming back later in the day. Coding information entered during the session is permanently recorded. This command puts you back into LOGLISP; type "C, then "X to return to monitor level

sub-codes types a list of subcodes

w permanently records coding information entered to that
point in the coding session, then returns you to the prompt (>*). Useful if you're worried that the machine might "crash"

5.2 RETRIEVING CODING INFORMATION

At any time, as you are working in the CODE program, you may want to retrieve some of the code information. In this section, we discuss the different formats you may use to access this information. If you are working at the terminal, you simply type the commands, and the program will produce the requested information. If you want a permanent record of this information, you have three options:

1. You can work on a hard-copy terminal. When you finish, simply take the output with you: that's your record.

2. You can work on a video terminal and create a log of your entire coding session. If you start the log after you enter the CODE program, you will have a record of everything that passes by you on the screen during that coding session. To start a coding log, enter the CODE program, then type the command

   >*(DSKLOG \(<file>.<extension>\))

   for example, >*(DSKLOG (CODING1.LOG))

   You will get a message telling you that the coding log has begun. If you want to end the log before you end the coding session, type the command

   (LOGOUT)

   You will get a message telling you that the log has ended. If you don't end the coding log yourself, the program will end it and create the file for it in your disk space when you end your session in the CODE program. Be sure to send a print command off before you log out; the program does not automatically print the coding log file.

3. Through LOGLISP, you can create a file for any specific command. After you are in the CODE program, type the command

   >*(WRITEANY \(<file>.<ext>\) 79 (THE T (Cmd <command you want recorded>)))
For example, if I want a record of all my main codes, I type the command

```plaintext
**{WRITEANY (MAIN.CDS) 79 (THE I (Cmd main codes))}
```

The program will create a file in my disk area called `MAIN.CDS`, which will contain all information produced from the command "main codes". After you have entered the (WRITEANY) command, you will have a file of the information from that command, but it won't come by you on the terminal. If you want to see this information on the terminal, type the command itself after the prompt (**).

One word of caution here; don't use the file extensions "$.COD" or "$.CDB" when you create file names. These extensions are used to keep coding information on each participant. If you inadvertently duplicate file names using these extensions, you may lose important files. This won't happen if you avoid using either of these extensions.

When you enter the CODE program, you respond to the prompt "File to be coded?". You have two options: you can type a file name or you can type "none". If you want to code a particular file, you will type in a file name. If you don't want to code a particular file, but you want to retrieve coding information, you type "none". Then, at the next prompt (**), you type the command

```plaintext
**{DSKIN (<participant name>.COD)<(participant name>.CDB)}
```

For example, if I want to retrieve some coding information on the Cook data, but I don't want to code any particular file, I type "none" to the prompt, "File to be coded?"; then I type the command

```plaintext
**{DSKIN (COOK.COD) (COOK.CDB)}
```

I am telling the program to load all the coding information for this particular participant, so that I can retrieve it. The program automatically loads these two files whenever you type a file name in response to the prompt "File to be coded?". If you want to retrieve coding information about more than one participant, simply type additional (DSKIN) commands, one for each participant for whom you want coding information. Thus, the commands

```plaintext
**{DSKIN (COOK.CCD) (COOK.CDB)}
**{DSKIN (CUMMGS.CCD) (CUMMGS.CDB)}
```
tell the program to load the coding information files for participants Cook and Cummings. After you have specified a file or have given (DSKIN) commands, you can retrieve coding information in a variety of formats.

5.2.1 The "fatcodes" Command

To find out how large your coding categories are, you will use the "fatcodes" command. This command is designated one of four ways:

fatcodes - types a list of main codes used in the current file with the number of text items assigned to each one

fatcodes <file id>-<extension> - types a list of main codes used in the specified file with the number of text items assigned to each one

fatcodes <participant id> - types a list of main codes used in any data files for the specified participant with the number of text items assigned to each one

fatcodes all - types a list of main codes used in any data files for all participants whose files are accessible (i.e., through "File to be coded*" or (DSKIN) command) with the number of text items assigned to each one

As you can see from the command descriptions, you can determine how large your coding categories are within one file, across all files for a particular participant, or across all files in your study. The fatcodes command only lists main codes; sub-codes are not broken out separately with this command.

5.2.2 The "pcodes" And "pcodesn" Commands

To get a summary of the coding information you have entered, you will use the "pcodes" command. This command is designated one of several ways:

pcodes - types lines and assigned codes (i.e., main codes and subcodes) for the current data file. The list will be alphabetical by main codes, and clustered, first by main codes, then by subcodes within main codes.
pcodes; <file id>-<extension> - types lines and assigned codes for specified file other than current data file

pcodes; <participant id> - types lines and assigned codes for all data files of specified participant

pcodes; all - types lines and assigned codes for all data files (i.e., the entire study)

The unusual placement of spaces and semi-colons in this command is purposeful. This form is a reminder to you that you can also specify code patterns with this command. A code pattern is any main code/subcode combination. Code patterns can be expressed particularly (e.g., specific words/phrases), or they can be expressed generally (i.e., using variables). In this discussion of pcodes, we will explain particular expressions (i.e., those you have assigned), and two variations involving general expressions. Refer to the section on advanced retrieval techniques for a discussion of more complex code pattern variations used with this (and other) commands.

If you want to specify code patterns in the pcodes command, you type in the code pattern before the semi-colon. The form of the command is

>*pcodes <main code>,<subcode>; <any of the 4 choices listed above>

In this example,

>*pcodes tchr act, eval; Cook-I2

I am asking the program to find all lines in the Cook-I2 data file that were assigned the code "tchr act, eval" (which stands for the coding category "teacher activity, evaluation").

If you want to specify a particular main code, regardless of the subcodes, use the command

>*pcodes <main code>\$sub; <any of the 4 choices listed above>

In this example,

>*pcodes tchr act\$sub; Cook

I am asking the program to find all lines in any Cook data file that were assigned the main code "tchr act" (which stands for the coding category "teacher activity"), regardless of any subcode assignments. The program will print cut a list indicating the file id (e.g.,
COCK-I2), the line numbers, the main code you specified, and the subcodes, if any.

If you want to specify a particular subcode, regardless of the main codes, use the command

```plaintext
*pcode $main, <subcode>; <any of the 4 choices listed above>
```

In this example,

```plaintext
*pcode $main, eval; Cook
```

I am asking the program to find all lines in any Cook data file that were assigned the subcode "eval" (which stands for the subcode category "evaluation"), regardless of any main code assignments.

The "pcodesn" Command

This command has the same forms and gives the same information as the "pcodes" command, with one additional piece. As the information is printed from this command, you will see a number or a set of numbers enclosed in parentheses. The single number or the number on the far right within the parentheses indicates the coding session in which these lines were coded. If you have a series of numbers, reading from right to left, you have a record of subsequent sessions in which these lines were recoded. This command is designed to give you a fast reference for coding change occurrences. If you want to track back through these changes in greater detail, you can move to your memo files; particularly MODCOD.CMM, which is a record of all modifications of code information, and <file id>.CMM, which is a record of your memos regarding specific coding categories. For a more detailed discussion of these files, see the section on the memo-writing function later in this chapter.

5.2.3 The "wrtext" Command

To write specific portions of text from your data files, you will use the "wrtext" command. The form of this command is similar to the "pcodes" commands:

```plaintext
**wrtext <code>; <current data file id>
```

```plaintext
**wrtext <code>; <specified data file other than current data file>
```

```plaintext
**wrtext <code>; <participant id>
```
Unlike the "pcodes" command, with "wrtext", you must specify the current data file in the command if you want text from that file. The "wrtext" command insists on the command word (wrtext), the code (main code and subcode), and the file(s) from which the text comes. You can specify more than one file by typing the file id for each data file of interest. In this example

>\*wrtext rdq act; Cook-I1 Cook-I4 Cook-I5

I am asking the program to print actual text from the three Cook data files that has been coded "rdq act".

In response to this command, the program will print a heading containing the file id, the beginning and ending line numbers of the text, and the code you specified. The text as it appears in the data file comes next. After all instances of specified coded text have been typed, the command response ends with a message telling you how many items were written.

If you want a printout of responses to "wrtext" commands, you will use the "wrtext file" command. This command has the form

>\*wrtext <file id>-<extension>; <code>; <any of the 4 choices listed above>

In the example

>\*wrtext RDGACT-TXT; rdq act; Cook

I am telling the program to produce a file containing all the text from any of the Cook data files that have been coded "rdg act". The program will create this file and store it in your disk area. Be sure to send a print command before you log out; the program does not automatically print a copy of the file. You can delete this file from your disk space after you have your printed copy, or you can leave it on disk. But, if at a later time, you create another such file with the "wrtext file" command, the newly created file will replace the old file in your disk area. We suggest you dump these files from your disk area after you get your printout. It's easy to create these files with the "wrtext file" command, so there's no sense paying storage charges for them in your disk area.
5.2.4 Other Commands Used In Retrieving Coding Information

Any of these commands may be given at the prompt (>*):

codes types a list of all the main codes for the current data file

endday stops the coding session, permanently records the coding information from that session; used to terminate the last session of the day

file types the description of the current file

h types a list of the commands you can use in the CODE program; you also get the list if you type "help"

h <command> [e.g., h codes] types a brief description of the specified command

kill terminates the coding session without recording information entered since the start of the session OR since the last "next" or "w" command; this may be a way to recover from an otherwise deadly decision to use the "dc" command. This command puts you back into LOGLISP: type ^C, then ^X to return to monitor level

main codes types a list of main codes and their file ids

stop stops the coding session temporarily, assuming you are coming back later in the day. Coding information entered during the session is permanently recorded. This command puts you back into LOGLISP; type ^C, then ^X to return to monitor level

sub-codes types a list of subcodes

text tags types all codes that affect given lines in the current data file. You must specify a line (e.g., text tags 24) or a set of lines (e.g., text tags 18 27) with this command
THE MEMORANDUM-WRITING FUNCTION

The memorandum-writing function records, organizes and stores the extensive notes generated during analysis. The program automatically begins a daily memo file on the first "memo" command, creating a two-line heading and displaying line numbers as you type (similar to SOS) for easy reference later in the day. There are three major components of this function:

1. "memo/mtype" creates or continues a daily general memo file and/or memo files for any specified coding categories; the former referenced as the chronological memo file and the latter referenced as individual coding category memo files;

2. "p/px/pt/pm" allows examination of data files and memos within the memo-writing function for easy reference;

3. "mcopy/mcopyx/mcopyt/mcopym" allows copying text from data files or memo files directly into the current memo.

As with the coding function, the memo-writing function allows great flexibility in organizing the memo information. At the command "endday", the program automatically creates and prints the daily chronological memo file which includes all information entered during any number of coding sessions on that day. This daily chronological memo file, called CHRON.CXX, is added to the previous days' memoranda in a comprehensive memo file (CHRON.CSM). The program also automatically excerpts those sections of the chronological memo file that pertain to specific coding categories and places those excerpts in their respective coding category memo files (CODE-ID.CAM). The result is a series of memo files which allows review of the memoranda from a "day-by-day" perspective, or from a "by-coding-category" perspective.

As you write memos, you can access any of your previous memos or any of your data files. You can print lines from any of these files, and you can copy text from any of these files into your memo. The roots of the commands are the same:

p - the print command
mcopy - the memo copy command

Two files are immediately accessible while you write memos - the current data file (if you specified one at the beginning of the coding session) and earlier memos that are part of the current day's memo
file. You can call up two other files with a command that has the root "file"—any other data file, designated by the letter "x" for auxiliary data file, and memos from previous days, designated by the letter "m" for memo file.

5.3 WRITING MEMOS

You can begin writing memos at any time after you enter the CODE program. To begin, you type the following commands at the prompt (>*):

**memo
**type

After you type these two commands, the program will type the line number "0003" on the left side of the terminal screen (or paper, if you're on a hard-copy terminal). This is the signal that the program has automatically started a chronological memo file for the day; the first lines are automatically created with a title and the date. You simply type in your notes/thoughts, just as you would with SOS. If you wish to start with a coding category memo, type the commands

**memo <main code>
**type

The program will produce the line numbers down the left side of the screen/paper; you type your memo.

Each time you give the "memo" or "memo <main code>" command, the program automatically inserts a "header line" in your memo. For example, if you type "memo tchq act", the header line in the memo will be

0001 >>> tchq act memorandum (25 Oct 82)

If, later in the session, you type "memo", the next header line in the memo will be

0436 >>> memorandum 25 Oct 82

These header lines identify those sections of the chronological memo which are specifically associated with coding categories.

If you catch spelling errors, typos, etc. on a line before you press "return", you can backspace to correct it. However, if you realize you've made an error several lines after the fact, you will
not be able to make changes until later. There is no edit mode in the memo-writing function. However, at the end of the day, before you send the memos off to be processed, the program will ask if you wish to edit your memos. If you respond "yes", the program will create an SOS file of your memos, and then you can edit them through SOS. When you finish editing, you send the memos off for "end of the day" processing.

Everything you type in this function becomes a part of the chronological memo file (CHRON.CSM). If you type the command "memo" in response to the prompt, the information you subsequently type appears only in the CHRON.CSM file. If you type the command "memo <main code>" in response to the prompt, the information you subsequently type appears in the CHRON.CSM file and in the coding category memo file (CCODE-ID-.CMM). You do not create any of these files; the program automatically creates and maintains all these files. Unlike other files in your disk area, these files are cumulative; as you type memos, they are automatically added to these files as a part of the "end of the day" memo processing.

As you type memos, you can switch back and forth between general memos and coding category memos. For example, if you have been typing a general memo and you now wish to type a memo pertaining to a particular coding category, you first hit the escape key after you have typed in your last line of general memo information. This gets you out of the general memo writing and returns you to a prompt (>). Now you type "memo <main code>" to the prompt, then "*type" to the next prompt. The program will begin producing line numbers again and you are ready to type the memo pertaining to the coding category you specified. Remember that coding category memo files are created and maintained by main codes only; the file id is the six-letter/digit abbreviation you specified at the time you first used the code, and the extension for all coding category memo files is ".CMM".

Whenever you wish to stop writing memos, press "return" after your last line, hit the "escape" key, and the program will return you to the prompt (>).

5.4 LOOKING AT MEMO AND DATA FILES

At any time you're in the CODE program, you can look at current memos, previous memos, the current data file (if you specified one), and any one other data file.
You type one command to access the current memo or data files:

```
p <first line> <end line> - prints specified lines from the current data file
pt <first line> <end line> - prints specified lines from the current memo file
```

You type two commands to access any other memos or any other data files. The first command calls up the file you want; the second allows you to print portions of that file:

```
xfile <file id> - tells the program to call up the specified auxiliary data file
px <first line> <end line> - prints specified lines from the auxiliary data file
mfile <file id> - tells the program to call up the specified previous memo file
pm (<date>) <first line> <end line> - prints specified lines from previous memo file
```

For example, if I want to look at previous memo files, I type the commands:

```
>**mfile CHRON.CSM
>**pm (9 Oct 83) 10 25
```

I am telling the program to print lines 10 through 25 of the October 9th memo entry in the chronological memo file. Perhaps you don't know exactly where specific information is in your previous memos, but you have a general idea. To give yourself a broader search range, you can use this form of the command:

```
>**mfile CHRON.CSM
>**pm (9 Oct 83) 10 (11 Oct 83) 50
```

You have asked the program to print your memos from the 10th line of your Oct 9th entry through the 50th line of your Oct 11th entry.
5.5 COPYING FROM MEMO AND DATA FILES

At any time you're writing memos, you can copy text from current memos, previous memos, the current data file (if you specified one), and any one other data file.

To copy text you use the "mcopy" command, which has the form:

\[ \text{mcopy} \text{ <first line> <last line>} \] - copies specified lines from the current data file to the current memo

\[ \text{mcopyt} \text{ <first line> <last line>} \] - copies specified lines from the earlier memos of the day to the memo on which you are currently working

\[ \text{mcopyx} \text{ <first line> <last line>} \] - copies specified lines from the auxiliary data file to the memo (remember you must call up this file with the "xfile" command first)

\[ \text{mcopym} \text{ (<date>) <first line> <last line>} \] - copies specified lines from previous memos to the current memo (remember you must call up this file with the "mfile" command first)

The format for the "mcopym" command is the same as the format for the "pm" command. You can copy memo text across dates using the same format described for the "pm" command.

When you are ready to copy data or memo text into your current memo, you hit the "escape" key to exit from writing your memo and the program will return you to the prompt. You can call up files ("file command) and/or look at files ("f" command). When you decide on the text to be copied, simply give the appropriate copy command. The specified text will be added to the end of your current memo, but you will not see this procedure on the terminal. However, if you return to writing memos, you will notice the difference in line numbers - the program will copy the text into your memo and will put the line numbers down the left side. When you get the processed memo file from the day (called CHRON.CXX), you will notice that any copied text can be identified by two different sets of line numbers: 1) the memo line numbers down the left side of the page, and 2) the file id information in the heading preceding the copied text, which includes the file id and the line numbers from the original file. For example,
if you copy lines 6 through 28 from COOK-I1, the header will be

0055 >> COOK-I1 lines 6 through 28:

followed by the text. This header line will appear for text copied
through commands "mcopy", "mcopym", and "mcopyx". The program does
not create a header line for text copied through "mcopyt". The reason
is simple: line numbers are likely to change when the day's memos are
processed.

If you want to continue typing memos after you have copied text,
just give the "mtype" command. You only need a "meme" command if you
are changing from daily memo to coding category memos or vice versa.

You may want to examine or copy from data or memo files, but you
can't remember the file names. In this case, you can look at your
directory without leaving the CCDE program. Use the ICGLIST command

>*(DIRF (*.CMN))

to get the directory listing of your coding category memo files; use
the command

>*(DIRF (<PARTICIPANT ID>.*))

to get the directory listing of any files pertaining to the specified
participant.

5.6 SUMMARY OF COMMANDS USED IN WRITING MEMOS

Any of these commands may be given at the prompt:

endday   stops the session, permanently records the coding/memo
information from that session; used to terminate the last
session of the day. On this command, the program will
allow you to edit the memos of the day, then will process
them (i.e., format and file them); the program also
automatically sends a print command for one copy of your
daily chronological memo file (CHRONCXX)

h        types a list of the commands you can use in the CCDE
program; you also get the list if you type "help"

h <command> [e.g., h codes] types a brief description
of the specified command
**kill**

Terminates the coding/memo session without recording information entered since the start of the session or since the last "w" or "next" command. This command puts you back into LOGLISE; type ^C, then ^X to return to monitor level.

**mcopy**

Copies specified lines from the current data file to the current memo.

**mccpym**

Copies specified lines from previous memo files to the current memo.

**mccpyt**

Copies specified lines from the earlier memos of the day to the memo on which you are currently working.

**mcopyx**

Copies specified lines from the auxiliary data file to the current memo.

**mdates**

Types a list of dates for which memos can be found in the file selected by the last "mfile" command.

**memo**

Tells the program to switch you to the memo-writing function of the CCDE program; followed by the "mtype" command.

**memo <code>**

Tells the program to record the text to follow as a memo specific to the designated coding category; this text appears in the chronological memo file and the specified coding category memo file.

**mfile**

Tells the program to call up a previous memo file.

**mtype**

Begins the line numbering input mode for memo writing; to stop this mode, press the "escape" key (ESC).

**newdate**

Tells the program to change the session date of the coding/memo-writing session to the date you specify; the command form: newdate (<day month year>)

**p**

Prints specified lines from the current data file.

**pm**

Prints specified lines from a previous memo file.

**pt**

Prints specified lines from earlier memos of today's (i.e., the current) memo.

**px**

Prints specified lines from the auxiliary data file.
stop stops the session temporarily, assuming you are coming back later in the day. Any coding information entered during the session is permanently recorded; any memos are permanently recorded, but not processed. This command puts you back into ICGLISP; type ~C, then ~X to return to monitor level.

w permanently records coding/memo information entered to that point in the session, then returns you to the prompt (>*). Useful if you're worried that the machine might "crash".

xfile tells the program to call up the specified auxiliary data file.

5.7 ENDING THE CODING SESSION

When you are ready to end a session in CODE, you can exit from the program one of three ways. Each command is given at the prompt (>*).

5.7.1 The "stop" Command

The command "stop" ends the session with the assumption that you'll be back later in the day. The program permanently records all coding and memo information, but it does not process any of the memos. When you come back for another session, the program will check with the computer for the date and, if it's still the same date as the last time you entered CODE, it will assume you wish to continue from the point at which you quit. The program will continue to record coding information, and it will continue daily memos and coding category memos (i.e., the line numbers will continue; they will not start renumbering from 0001). The main function of the "stop" command is to allow you some flexibility with memo writing. You can write memos periodically throughout the day and still have one, continuous, daily memo that reflects your activities and thoughts on any specific day.

You might end a session with "stop", perhaps thinking you'll be back later in the day, but you don't get back until a day or two later. The program will check the computer for the date. Finding that the current date and the date of the last session don't match, the program will remind you that you have unprocessed memos and will ask if you want to process them. If you say "yes", the program will allow you to edit them, then send them off for processing. You can re-enter CODE and start the current coding/memo-writing session under
the current date. If you want to continue under the previous date, answer "no" when the program asks if you want your previous memos processed. In this case, the program will pretend that the previous date still holds and will continue from the point at which you quit. When you want to stop working under the previous date, type "endday" (see explanation below). This command will take you out of CODE. If you want to do more work under the current date, you simply re-enter CODE, and you will be working under the current date.

5.7.2 The "endday" Command

The command "endday" ends the session with the assumption that you are through for the day. After you give the command, the program will ask if you want to edit your memos. If you answer "yes", the program will create an SOS file of your memos; you make changes as you would on any SOS file (see SOS reference manual). When you finish editing this SOS file, you type "e" to exit from the file, then type "/finish" to the monitor-level prompt (>). The program will process the daily memo file:

1. it creates a formatted file of the daily memo (CHRON.CIX) and prints one copy of it (this file will replace the previous one in your disk area);

2. it adds this file (CHRON.CIX) to CHRON.CSM, which is the cumulative, daily chronological memo file;

3. it excerpts those sections of CHRON.CIX that were written in reference to specific coding categories (from "memo <code>" command) and places those excerpts in their respective coding category memo files (CODE-ID.CFN).

All coding and memo information is permanently recorded with either "stop" or "endday". "Stop" leaves your memos unprocessed; "endday" tells the program to process them. You might use "endday", thinking you've quit for the day, then decide to go back later that same day. If you do, there's a problem. The program will check the date, will have nothing to check it against, since there are no unprocessed memos, will assume you're starting fresh, and will begin memos with new line numbers. The catch is you'll wind up with two sets of memo entries for the same date with the same line numbers, but different text. This creates program havoc when you try to reference these memos. To avoid this, you use the "newdate" command. SC, if you enter CODE for the purpose of writing memos (this isn't a problem if you're only entering coding information) on a day you've already used the "endday" command, be sure to give the "newdate" command to the first prompt (>*) after you enter the program. You will type
> *newdate (<day month year>)

For example, if I use "endday" on November 28th at 6 p.m., but decide to write another memo at 10 p.m., I'll type

> *newdate (29 Nov 83)

Whenever you type the name of a month in a "newdate" command, use only the following abbreviations:

<table>
<thead>
<tr>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>June</th>
</tr>
</thead>
<tbody>
<tr>
<td>July</td>
<td>Aug</td>
<td>Sept</td>
<td>Oct</td>
<td>Nov</td>
<td>Dec</td>
</tr>
</tbody>
</table>

5.7.3 The "kill" Command

The command "kill" ends the session with the assumption that you have made an error which has extensive implications. "kill" terminates the session without permanently recording any of the coding/memo information. There may be some trade-offs with this command, so it's important to be careful how you use it. For example, you may have entered coding information, written some memos, then used the "rc" command - with unforeseen and undesired results. You can use the "kill" command to recover from the "rc" command, but you may also lose your coding and memo information. With some care, this kind of thing can be avoided, making the "kill" command a useful one.

There are four commands that will save coding and/or memo information entered during a session: "endday", "stop", "next", and "w". The first two save your information, but dump you out of CODE program. You use them only if you want out of CODE with all information permanently recorded. "Next" and "w" save all coding/memo information entered prior to giving the command, and allow you to continue within the CODE program. If you decide to use the "kill" command, any information entered prior to a previous "next" or "w" command will be saved; anything entered between the "next" or "w" command and the "kill" command will be lost. "Next" is used for purposes other than saving information, so it's not the most useful command in this context. The "w" command is specifically designed to save coding/memo information entered to that point in time during the session. It's wise to use it occasionally as you code and/or write memos. Then if you make a deadly decision that you want "to kill" or the machine crashes, all information entered prior to the last "w" command is safe.

If, during a session, you want to use a somewhat risky command such as "rc" or "dc", use the "w" command first. That will save everything entered previously. If you get into trouble with the next
command or two, you can "kill" them with no loss of previously entered information.

Two other commands will save some previously entered information. When "pt" and "mcopyt" are used, the program internally creates new files to give you access to this memo information. Consequently, if you're considering using "kill" and have recently used either of these commands ("pt" or "mcopyt"), any memo information entered before these commands will be safe.

These three commands, "stop", "endday", and "kill" all end your session in CODE. If you type "endday", you will be asked about editing your memos, and given directions about how to proceed from that point. If you type "stop" or "kill", you will get the LOGLISP prompt (*). Holding down the "control" key (CTRL), type "C", then "X". This will get you back to the monitor-level prompt (*). To log out, type "k" to the monitor prompt.
References


