DOCUMENT RESUME

ED 037 833	24	EA 002 863

AUTHOR	Lewis, George Hallam
TITLE	The STGPROC System of Data Manipulation by Computer: Manipulation of Character Data in the Social Sciences.
INSTITUTION	Oregon Univ., Eugene. Center for Advanced Study of Educational Administration.
SPONS AGENCY	Office of Education (DHEW), Washington, D.C. Bureau of Research.
REPORT NO	TR-4
BUREAU NO	BR-5-0217
PUB DATE	Feb 70
CONTRACT	OEC-4-10-163
NOTE	158p.
AVAILABLE FROM	CASEA Editor, Center for the Advanced Study of Educational Administration, Univ. of Oregon, Eugene, Oregon 97403 ($2.00)
EDRS PRICE	EDRS Price MF-$0.75 HC-$8.00
DESCRIPTORS	Bibliographies, *Computational Linguistics, Data Analysis, Data Collection, *Data Processing, Information Processing, Information Storage, Input Output, Permuted Indexes, Programing, *Programing Languages, *Social Sciences, Sociometric Techniques, *Statistical Analysis, Taxonomy

ABSTRACT

STGPROC is a computer program designed to meet a need in the social sciences for a computing system that (1) can handle character strings of open-ended data, (2) does not require predefinition of the data, and (3) can handle variable numbers of responses per respondent. The purpose of this monograph is to allow the researcher to judge the utility of STGPROC and its value for his specific research needs. In the general presentation, no knowledge of computing procedures is assumed. The monograph provides the researcher with a general idea of how STGPROC operates and how it differs from existing programs in the social sciences. The STGPROC system of transcribing data is detailed and ways to utilize the program are explained. The researcher with no programing experience should be able to explain to a programmer any additional operations that he needs. For those with some programing experience, the main STGPROC program and its existing subroutines are reproduced, along with comment cards, in an appendix. (DE)

BR-5-0217
Project 2003
OE/BR
PA-24
TE/0

Technical
Report
No. 4

Manipulation of Character Data in the Social Sciences

A Presentation of the STGPROC Program

George Lewis

February, 1970
Center for the Advanced Study of Educational Administration

THE STGPROC SYSTEM OF DATA MANIPULATION

BY COMPUTER

George Hallam Lewis

Bureau No. 5-0217, Project No. 2003

Contract No. 4-10-163

Funding Authority: Cooperative Research Act

February, 1970

The research reported herein was conducted as part of the research and development program of the Center for the Advanced Study of Educational Administration, a national research and development center which is supported in part by funds from the United States Office of Education, Department of Health, Education, and Welfare. The opinions expressed in this publication do not necessarily reflect the position or policy of the Office of Education and no official endorsement by the Office of Education should be inferred.

Center for the Advanced Study of Educational Administration
University of Oregon Eugene, Oregon

TABLE OF CONTENTS

Page

LIST OF FIGURES . . . .

Chapter

I. OVERVIEW 1

The Program STGPROC 7
Perspective 9

II. CODING AND TRANSCRIPTION 10

Transcription as it Applies to STGPROC 14
Transcription of Interrogative Responses 21
Response and Respondent Identification 24
Keypunching 27
The System and Coding: Three Considerations 29
Coders and the Transcription System 35

III. THE MAIN PROGRAM 42

Procedures 43
The String Functions 44
Instruction Cards: The Main Program 48
STGPROC Operations 50
PRINT RUN 54
ERROR RUN 54
SUBSECTION 58
SELECT-IN and SELECT-OUT 59
The Instruction Cards 59
Summary 61

IV. THE MANIPULATIVE SUBROUTINES 64

The Storage of Data in QUES_ARRAY 65
Manipulative Subroutine: Type One 67
Manipulative Subroutine: Type Two 72
Manipulative Subroutine: Type Three 81
Summary 86

V. THE STATISTICAL SUBROUTINES 87

The Statistical Package for the Social Sciences 88

TABLE OF CONTENTS (Continued)

Chapter Page

The Statistical Subroutine SPSS 90
The Statistical Subroutine FACT_AN 95
Summary 99

VI. STGPROC IN ACTION 101

Conceptual Evolution 101
STGPROC: Phase One 105
STGPROC: Phase Two 109
STGPROC: Phase Three 114
STGPROC: The Future 116

BIBLIOGRAPHY 118

APPENDIX ONE
THE STGPROC PROGRAM 122

APPENDIX TWO
RESPONSE TYPE AND SUBRESPONSE: SUGGESTED TABULATION SHEET . . 145

APPENDIX THREE
THE INSTRUCTION CARDS 146

APPENDIX FOUR
JOB CONTROL CARDS 149

APPENDIX FIVE
BASIC STGPROC PROGRAM OUTPUT 151

LIST OF FIGURES

Figure Page

I. Coding and Translation 11

II. Example of Delimiter Symbols 16

III. Conceptualization of Interrogative Response (A) 18

IV. Conceptualization of Interrogative Response (B) 19

V. Arrangement of STGPROC Procedures (A) 45

VI. Arrangement of STGPROC Procedures (B) 46

VII. Example of Data Stored in QUES_ARRAY 66

VIII. Example of "THES" Array 75

CHAPTER ONE

OVERVIEW

"It can be said without fear of contradiction that the greatest value of computers to social science will be through innovations that have not yet been made." (Coleman, 1964: 1047) James Coleman penned the above statement in 1964. Indeed it has long been evident that many of the techniques of data manipulation at present employed in computer analysis are inadequate for the social scientist. The blame for this fact may be laid as heavily upon social scientists as upon the computer programmers--perhaps the fact that these are "two breeds" of man in many cases helps to account for the lack of communication between them.

Social scientists as well as educational researchers rely heavily upon reported data--interviews, open-ended questionnaires, written records. And yet the translation of these raw data into the numeric categories and codes necessary for computer analysis prohibits a vast amount of data manipulation. This technological imperative has resulted in the fact that many fruitful studies are not undertaken and many completed projects have been forced into a restricted mode of data manipulation which has led in turn to narrow and pre-defined types of analysis.

This problem, however, does not affect all types of data collection. Many studies, for example, those dealing with the demographic variables, are perfectly suited for conventional numeric data manipulation. The problem does become evident in the following types of studies: (1) panel studies; (2) nominational studies; (3) exploratory studies dealing with classification schemes; (4) survey analyses; (5) other types of studies dealing with responses of an open-ended nature.

Computers are machines that compare and rearrange information--not just machines that perform arithmetical operations. It sometimes comes as a surprise to those not familiar with computers that they can also "read" and manipulate words. Although the provision for handling alphabetic data has been standard since the earliest days of computer technology, it is only recently that methods for programming this type of data easily in a system primarily designed for numerical manipulation have been created.

In the past few years, new computer languages have been created. These languages allow for the extended manipulation of data in the form of character strings--potentially a real methodological breakthrough for the social scientist who is thinking of conducting large scale exploratory research. However, to date there has been little done to adapt this unique tool to the needs of the social scientist. In general, social scientists and educational researchers are either unaware that the tool exists or have not the training in computer science to utilize it.

The few adaptations of this tool that have been created fall generally into two main categories. One is the utilization of strings of character data to create simple frequency counts (cf. Nie and Bent, 1968). This type of computer program operates in a directly analagous fashion to the "standard" type which employs numeric data to perform "matches." In other words, instead of looking for a pre-defined "2" for a match, the program can instead look for a pre-defined "A" and match in the same manner. However, programs of this type cannot handle the manipulation of open-ended data, nor can they handle responses that are not pre-defined. A further problem is their inability to process varying numbers of subresponses to a question (these numbers varying per respondent).

A second type of program has been created to handle open-ended data. This type of program, however, is designed mainly for use in content analysis research--and that research not in sociology (cf. Stone, Dunphy, Smith, Ogilvie, 1966; Harway and Iker, 1964; Starkweather and Decker, 1964). Generally programs of this type are bulky, expensive to utilize, and not adapted to the needs of the social scientist.[1] However not all the blame can be attributed to the programmers. They have not heard from the social scientists on this matter. In speaking of the sociology of language (and indirectly to the problem of the manipulation and analysis of character string data through the social sciences) Bernstein has said: "What is a little odd is the negligible contribution of sociology to the study of language. The textbooks celebrate the fact of man's symbolic possibilities in chapters on culture and socialization and then the consequences are systematically ignored. One might go as far as saying that the only time one is made aware that humans speak in the writings of contemporary sociologists is incidentally through the statistical relations induced from social survey inquiries. And here all that is required is that the subjects can read; speech confounds the later arithmetic." (Stone, Dunphy, Smith, Ogilvie, 1966)

[1] A great deal of the work in this area has been in terms of information retrieval processes applied to bibliographical material--yielding data manipulation no more sophisticated than listings (cf. Janda, 1964; Janda, 1965; Vinsonhaler, 1967; Wilcox, Bobrow, Bwy, 1967). The exception is The General Inquirer (Stone, Dunphy, Smith, Ogilvie, 1966) system. Unfortunately this system also embodies the problems pointed out in the text above.

A major need in social science then is a type of computing system that: (1) can handle character strings of open-ended data; (2) does not require pre-definition of the data; (3) can handle variable numbers of responses per respondent.[1] With a system of this sort, no data need be coded prior to machine processing.

There are further problems with the conventional system of data manipulation, stemming from the necessity of coding the data in alphanumerics before computer analysis. First, the data have to be broken down into t.eir respective categories prior to analysis (resymbolized to a more abstract form). Second, when a coding error of the type in which the coder either misreads the datum or misplaces it in the category scheme occurs, there is no chance of ever finding the error. Third, the data have to be retranslated after computer analysis (resymbolized to a more empirical level). Fourth, differences among coders both within and between projects yield a lack of reliability and comparability of coded data.

The ability to manipulate data as character strings solves many of these problems. In the first place, data may now be entered into the computer just as they appear on the research instrument. This obviates the necessity of coding the data for entry purposes as well as obviating translation necessities after analysis. The import of this process is that the coding of the data can be done by the machine, eliminating all but standard machine error (physically faulty data cards, and so on) and coder spelling

[1] During March 1969 the author checked with the major academic computation centers in the United States and found no evidence of activity towards this goal.

errors.[1]

The feasibility of machine coding is in itself a great breakthrough in social science, for although processing error in research generally earns little more than a word of warning in methods literature (cf. Kish, 1965; Riley, 1963), recent findings suggest that processing error: (1) stems primarily from the human coding process; (2) may not be random; (3) even if it is random, can distort research findings; (4) is sufficiently high to warrant efforts at its elimination (Sussman and Haug, 1967). Machine coding of raw data obviously eliminates human bias in the coding process (although the problem of bias in category selection remains, it is the job of the researcher to insure reliability and validity in this area, no matter what type of coding is utilized).

By machine coding data, one solves a further problem that has plagued social science and educational researchers. The results of differing studies employing category systems may be compared. If the same computer coding system is used in two sets of data, the results are free of both random coding bias and the judgmental differences of coders and/or researchers.

By making use of the ability of the computer to handle character strings, a researcher opens the door to a great number of approaches to the data of the empirical world that were once deemed closed. Not only can he collect data without a stringent eye upon its simplified coding into pre-defined categories (methods of data collection do help determine results /cf. Runkel, 1965/) but he can also re-examine his data with as

[1] Spelling errors may be picked up upon data printout, as will be considered in the appropriate sections of this monograph.

many classification schemes as he wishes without resorting to the tedious and inefficient method of human recoding of the data (inflexible classification yields bias and a lack of comparability across studies).[1] In addition, he no longer has to worry about bias and error introduced by human coders. In a nutshell, the researcher may utilize the computer to manipulate his data with no loss of information or bias due to coding necessities.[2]

A word of warning: it is often tempting, given the capabilities of present day computers, to fall into the trap of measurement for measurement's sake. With the feasibility of character manipulation of string data by computer, this danger is increased. Theory building and hypothesis testing must not be sacrificed to the ease of data measurement and description. It cannot be stressed enough that the computer is a reliable aid to the researcher, handling the mechanics of formal analysis. However, the computer is not a substitute for thought. It is an aid to it. The program described in this monograph was designed with the above in mind. It is not intended as a "statistical panacea," but as a process of interaction and feedback between the machine processing and the researcher.

[1] STGPROC is one solution to this problem which has been plaguing survey researchers for years and to which no solution in terms of computer programming had been found (mainly because of the restrictions of most computer languages), although some solutions have been attempted (cf. Wilcox, Bobrow, Bwy, 1967).

[2] The traditional problem of respondents creating unanticipated data, such as the inclusion of extraneous comments and the modification of individual questions, is an exception to this statement. In these cases individual decisions as to how to deal with the data are still necessary.

The Program STGPROC

STGPROC is a series of interrelated procedures developed under the auspices of The Center for the Advanced Study of Educational Administration at the University of Oregon during the period from June, 1968 through June, 1969. The purpose of this monograph is to allow the researcher to judge the utility of STGPROC and its value for his specific research needs. In terms of the general presentation, no knowledge of computing procedures is assumed. Hopefully a researcher, after study of this monograph, will have a general idea of how STGPROC operates and how it differs from existing programs in the social sciences. He will also be aware of the form in which to shape his data in order to utilize the program and of the program's general capabilities.

Although many subprograms may be written or adapted for use with STGPROC only five are presented in this monograph. These five have been developed during the processes of data manipulation connected with actual research. Together they form an efficient, flexible, and highly general package of data manipulations.

The researcher with no programming experience, after studying this monograph, should be able to identify to a programmer any additional operations which he has need for and which are feasible in conjunction with the main STGPROC program. Those with some programming experience should be able to write or adapt subroutines to STGPROC with little difficulty. For those with programming experience, the main program STGPROC and its existing subroutines are reproduced, along with comment cards, in Appendix One of this monograph.

STGPROC is a set of computer procedures to: (1) identify systematically instances of words that belong to categories supplied by (a) the researcher, (b) the response content, (c) the content of another response; (2) count occurrences of these categories; (3) output listings, frequencies, and related percentages of these categories; (4) arrange the data in forms amenable for use by existing statistical and factor analysis programs; (4) sort and regroup categories according to supplied specifications.[1]

The program is written in IBM's Programming Language One (PL/1).[2] It may be used in any computer installation containing a PL/1 level F program compiler.[3] PL/1 was chosen in preference to three other computer languages with which the author is familiar: SNOBOL4, FORTRAN IV, and Assembler Language. Because SNOBOL4 was developed by Bell Telephone Laboratories, it is less likely that computation centers would have SNOBOL4 compilers than compilers for languages developed by more well-patronized computer firms. SNOBOL4 also is no more efficient for this particular computing problem than is PL/1 and is less agile, more bulky, and lacks certain necessary functions. Both Assembler Language and FORTRAN IV were rejected because of the fantastic mass of programming it would take to allow them to handle string processing in the clean manner PL/1 has been set up

[1] See Appendix Five for a general description of the basic STGPROC program output.

[2] Relevant IBM manuals concerning PL/1 programming language are cited in the bibliography.

[3] STGPROC was created by means of the PL/1 level F, version 4 compiler at the University of Oregon computing center.

to do. PL/1 is also IBM's most recent language, replacing FORTRAN IV in many academic industrial computation centers.

Perspective

Chapter Two of this monograph concerns itself generally with the differences between coding and transcription and discusses the problems associated with coders and how the new processes of the STGPROC system alter these problems. This chapter also explains in detail the STGPROC system of transcribing data. Chapter Three explains the main STGPROC program and how to utilize it. Chapter Four discusses three general types of manipulative subprograms that may be used in conjunction with the main program and gives general examples of each type. Chapter Five describes two additional subroutines which allow the researcher to tie STGPROC in with existing statistical and factor analysis programs. Finally, Chapter Six contains the reportage of an actual research project carried out under the auspices of the Center for the Advanced Study of Educational Administration at the University of Oregon utilizing STGPROC at its phases of data manipulation and analysis.

CHAPTER TWO

CODING AND TRANSCRIPTION

In order to understand more fully the purpose for which the program STGPROC was created, it is necessary to distinguish between the concepts of *coding* and *transcription*. Let me here offer definitions. *Coding* is the process of resymbolizing information. This resymbolizing can be in two modes, depending upon the use to which the process is to be put: (1) resymbolizing from a known symbol system to an unknown one; (2) resymbolizing from a known symbol system to one characterized by fewer symbols (see Figure I).

The first mode of resymbolizing has as a motive the transmission of information comprehensible to those in possession of a key with which to translate the unknown to a known symbol system.[1] The second mode of resymbolizing has as a motive the more efficient storing or transmission of information.

These two modes of resymbolizing are of course not mutually exclusive. For example, in social science research, the second motive for resymbolizing is prevalent. However the new symbol system is often incomprehensible without some sort of key.

A basic problem in any type of exploration is what to seek and to observe. In terms of social science exploration, this problem takes the

[1] Translation will be defined later in this section.

Figure I: Coding and Translation[1, 2]

Relative number of meaning-symbols	Symbol systems	
	Known	Unknown
Fewer symbols	A	B
More symbols	C	D

Coding: Pure Mode I

A to B
C to D

Coding: Pure Mode II

C to A
D to B

Coding: Mixed Mode

C to B

Ideal Translation

B to A
D to C

Imperfect Translation

D to A

Infeasible Translation

B to C
A to C
B to D

[1] The assumption underlying the use of the above concepts is that of a loss of information when passing from one symbol system to another characterized by relatively fewer symbols.

[2] The author recognizes the simplicity of the case as stated above. The grammatical context of meaning-symbols and adjoining meaning-symbols have also to be taken into consideration when one is attempting translation. The lexical and syntactical considerations are not covered here, as the primary purpose is not in describing translation but in pointing out the differences between coding-translation and the process of transcription.

form of what aspects of the empirical world one shall consider as bearing upon one's problem. Data collection then becomes a process of selectively excluding aspects of the empirical world while retaining those one feels are most relevant to one's problem.

In research there exists a step between data collection and data analysis--a step I shall call data manipulation. Data manipulation involves shaping the data to forms amenable to analysis. One such manipulation has to do with resymbolizing the data in the second of the two modes mentioned above. For example, in social stratification research, specific jobs (raw data) may be resymbolized into a system of occupational categories and a key, called a code book, may be drawn up.

In modern computer-oriented research, a further coding step has to be taken. This is resymbolizing of the first mode mentioned above. The system of occupational categories is resymbolized as decimal number combinations which are then further resymbolized by the computer to binary integer combinations. The computer may then perform operations upon the data, translate the results to decimal number combinations which are then returned to the researcher for further translation.

Here an important point is raised. In the first mode of resymbolizing mentioned above, such as the computer performs, translation may be performed. Translation may be defined as resymbolizing from an unknown symbol system, by use of a key, to a comprehensible one. Accurate translation may be performed only if there exists a unique meaning-symbol in one symbol system for each meaning-symbol in the other. Translation becomes less accurate the further one gets from this ideal (see Figure I).

In the second mode of resymbolizing mentioned above, translation, by definition, is at the very most highly inaccurate and in most cases virtually impossible. Said in other words, the loss of information in this type of coding nearly always precludes any chance of translation.

Thus in the above social stratification example, analysis results will be couched in terms of occupational categories. No further translation can be performed. (And in this case, none further would be desired.)

One important aspect of computers is their ability to perform a combination of many simple tasks rapidly and in an efficient and reliable manner. A major breakthrough for much exploratory research in the social sciences would be the performance of all the necessary coding of data by the computer. This would not only increase the reliability of the final results, but would cut down tremendously upon the paid hours of coding time. The STGPROC program is one designed to allow computerized coding of data. In the above social stratification example, the raw data consisting of job names would be processed by the computer directly. Results would be in terms of any occupational category scheme or schemes one had entered in STGPROC prior to data analysis. Said in other words, one of the major reasons for the creation of STGPROC is to transfer the majority of the step of data manipulation from human hands to the computer. In order to accomplish this, the raw data have to be transcribed.

Transcription then, is the transference of a set of symbols from one medium to another. In this case, from the collected research schedule to the computer input device.

In some instances human coding may be utilized with STGPROC. These instances will be examined in later portions of this chapter. However, the major data manipulation prior to computer analysis consists of the transcription of data. This will be taken up in detail in the following section.

Transcription as it Applies to STGPROC

The Concept of Delimiter Symbols

Every symbol system, whether written or verbal, is a vehicle with which to transmit information. In order to accomplish this objective, the information has to be *ordered* in such a way that it is comprehensible. Thus there exist two types of symbol in every symbol system: (1) those which represent the information to be imparted (content-informational symbols) and (2) those which order this information in a comprehensible way. This second type of symbol I shall refer to as a *delimiter symbol* and define thusly: A *delimiter symbol* divides or marks symbolic information at points significant to the ordering of this information such that it may be comprehended by one familiar with the utilized symbol system.

An example of delimiter symbols (or delimiters) in the semantic world are *punctuation marks*, without which symbolic information is difficult to understand. An excellent example of this is the problem of comprehending James Joyce's *Finnigan's Wake* and other examples of "stream of consciousness" writing which is characterized by, among other things, a dearth of punctuation. Delimiters are just as important in the transcription process associated with the program STGPROC as they are in any situation involving symbol transference.

Delimiters always have an *order of priority* in the division of symbolic information. Probably the easiest way to explain this is to use the example of punctuation in the English language.

In the English language, the delimiter with the highest priority is the *period*. This delimiter divides *sentences* (symbol strings which are understandable by themselves, containing both subject and predicate, and necessary qualifiers). Delimiter priorities in descending order in English include: the *colon*, the *semi-colon*, the *comma*, and the *blank*. Some delimiters stand on the same priority level and are utilized to distinguish between *modes of signification* of the symbol string. Examples are the *period*, *question mark*, and the *exclamation mark*. (I will have opportunity to employ delimiters in this fashion; however, this will be taken up later in the chapter. Here it is necessary to grasp only the concept of *delimiter priority*.)

Delimiters are used in a priority fashion in the program STGPROC. The program allows the individual researcher to determine not only what symbols he will use as delimiters, but also how many and upon what priority level they will appear. The only limitation is that *any symbol that appears or is likely to appear in the researcher's content-informational symbol strings cannot be used as a delimiter*. For example, in the English language, the period is a delimiter, denoting the termination of a sentence. It is not used by itself in any content-informational sense. However, in the decimal arithmetic symbol system, the same symbol, the period, is used to impart information. Here it is known as the *decimal point*. Obviously it would be confusing to use periods as delimiters in decimal arithmetic symbol strings. The analogy holds in the transcription system developed for STGPROC. In this case, any symbol correspondence between content-informational and delimiter symbols spells disaster, as the computer is programmed to identify specified symbols solely as delimiters and to process the symbol strings in certain specific manners upon identification of these delimiters.

The delimiter symbols are identified by the researcher for the specific program run in two instruction cards which he must insert as data in the STGPROC program.[1] The general form of these instructions are:

1) NUMBER OF DELIMITERS = n .

28) IDENTITY OF DELIMITERS = s_1 , s_2 , s_n .

where

n = the number of delimiter symbols utilized

s = a delimiter symbol

note

delimiter symbols are listed in order of descending priority

As an example taken from the actual use of the program STGPROC (see Chapter Six), the following delimiter symbols were utilized, priority order from high to low as read from left to right:

Figure II: Example of Delimiter Symbols

$ # % : ; |

In this case, the two delimiter instruction cards of STGPROC would

[1] A discussion of instruction cards and their placement will be found in Chapter Three. This aspect of STGPROC is not yet necessary to the ongoing explanation.

read in the following manner:[1]

1) NUMBER OF DELIMITERS = 6.

28) IDENTITY OF DELIMITERS = $,#,%,:,;,!.

All symbolic data in this example coded by STGPROC must employ the above six symbols as delimiters in their listed priority order. How these symbols will be employed is determined by further considerations to be spelled out in ensuing sections of this chapter.

The Concepts of Subresponse and Response Type

The informational response to any interrogative can be conceptualized in two ways:

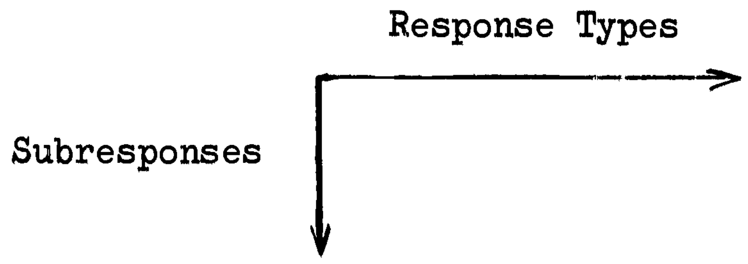
1. By the *types* of information given. For example, the response, "Saul Franks, technician at XYZ Corporation" includes three types of information: (a) name, (b) job title, (c) place of employment.
2. By the number of information bits of the same type (hereafter referred to as the number of *subresponses*). For example, the response, "Saul Franks, Mannie Roberts, Fred Benson" includes three subresponses.

[1] It should be apparent from the preceding discussion of delimiters that in the two delimiter instruction cards, the symbols *equals* (=), *comma* (,), and *period* (.) are employed as delimiters themselves. The same is true of all the instruction cards. This would normally prohibit the use of these three symbols as delimiters in the STGPROC program as their declarations in statement 28 would be impossible. Yet, provision has been made such that any of these three symbols may be utilized. If any of these symbols are to be declared as delimiters, they must be entered in instruction card 28 *enclosed in single quotes*. For example:

28) IDENTITY OF DELIMITERS = $, '=', ',' , '.' .

One may think of the two means of conceptualizing the response to an interrogative as related in the following way:

Figure III: Conceptualization of Interrogative Response (A)



As an example, suppose a respondent was asked to name those three with whom he interacts most frequently on the job and the official positions that each hold. Now suppose the response to this interrogative is as follows:

Saul Franks	technician
Mannie Roberts	technician
Fred Benson	foreman

The response can now be fitted into the "conceptual grid," or array, in the following manner:

Figure IV: Conceptualization of Interrogative Response (B)[1]

		Types of Response	
		1	2
Subresponses	1	Saul Franks	technician
	2	Mannie Roberts	technician
	3	Fred Benson	foreman

In this manner, every interrogative can be characterized by a two-dimensional array, the size of the maximum anticipated response upon each of the two dimensions. For example, the above interrogative response fits exactly into a three by two array. This is the way the program STGPROC would store this response, had it been transcribed as data.

There is one further thing that needs to be known about the response before it can be stored by use of STGPROC. That is the maximum number of symbols used in any single cell of the array. Here it is fourteen ("Mannie Roberts").[2]

[1] Figure IV is a three by two array, the "three" referring to the number of subresponses, the "two" referring to the number of response types. Each cell of the array can be identified by its subscripts. For example, the subscripts of the array cell containing "Saul Franks" are 1,1, and the subscripts of the array cell containing "foreman" are 3,2.

[2] Note that the blank between "Mannie" and "Roberts" counts as a symbol.

Hence in utilizing STGPROC, the researcher must first determine the following things.[1]

(1) The number of interrogatives per schedule

(2) For each interrogative:

(a) the number of response types

(b) the maximum number of subresponses (for open-ended interrogatives this is an educated guess)

(3) In terms of the total schedule, the maximum symbol string length

There are four instruction cards in which the researcher supplies the above information. The general forms of these statements are:

2) NUMBER OF QUESTIONS = n.

3) RESPONSE LENGTH = 1 .

29) SUBRESPONSES PER QUESTION = s_1 , s_2, s_n .

30) RESPONSE TYPES PER QUESTION = r_1 , r_2 , r_n .

where

n = the number of responses per schedule

1 = the maximum symbol string length

s = the maximum number of subresponses, by question

r = the maximum number of response types, by question

For example, a schedule with two interrogatives; the first with three response types, two subresponses, and a maximum response length of twenty characters; and the second with two response types, four subresponses, and a maximum response length of thirty characters would result in the follow-

[1] It is suggested that the researcher employ a tabulation sheet for this purpose. An example of such is supplied in Appendix Two.

ing instruction cards:

2) NUMBER OF QUESTIONS = 2.

3) RESPONSE LENGTH = 30.

29) SUBRESPONSES PER QUESTION = 2 , 4 .

30) RESPONSE TYPES PER QUESTION = 3 , 2 .

Transcription of Interrogative Responses

With the above discussion of response types and subresponses in mind, it should be clear why delimiter symbols are needed in transcribing data for use with STGPROC. Generally speaking, delimiters are needed for three purposes: (1) to separate responses; (2) to separate response types; (3) to separate subresponses. The separation of responses is of the highest priority, hence in our continuing example, responses will be separated by the delimiter symbol "$" (see Figure II for the delimiters utilized). The rule is RESPONSES SHALL BE FOLLOWED BY THE FIRST PRIORITY LEVEL DELIMITER. The use of delimiter symbols to separate response types and subresponses is a bit more complicated to explain. To begin, I shall use the response as diagrammed in Figure IV as the basis of explanation.

With the information thus far given, the transcription for Figure IV is as follows:

response$

Since the symbol "$" is used only to separate responses, the delimiter symbol "#" (Figure II) is of the highest priority <u>within</u> any response, just as in English the colon is the highest priority delimiter symbol <u>within</u> a sentence. I shall signify this relation from now on by referring to the delimiter symbol <u>within</u> a response as:

n level_w

with n signifying the priority level of the delimiter symbol within the response.

The second general rule of response transcription is ALL INFORMATION OF ONE RESPONSE TYPE IS TO BE COMPLETELY TRANSCRIBED BEFORE BEGINNING TRANSCRIPTION OF INFORMATION OF ANOTHER TYPE. In the case shown in Figure IV, the above rule means that type one should all be transcribed, then type two. Now it is obvious that the first level_w delimiter should be used to separate response types. The rule is RESPONSE TYPES ARE ALWAYS SEPARATED BY THE FIRST LEVEL_W DELIMITER. In this case, the two response types will be separated by the symbol "#" or the first level_w delimiter symbol (see Figure II) thusly:

type 1 (name)#type 2 (position)$

It is now necessary to separate the subresponses within each response type. The appropriate delimiter symbol is found by use of the following rule. SUBRESPONSES WITHIN RESPONSE TYPES ARE SEPARATED BY THE (NUMBER OF RESPONSE TYPE) LEVEL_W DELIMITER. In this case there were two response types, so it would be the (2) level_w delimiter. Since the first level is "#", the second level_w delimiter is "%" (see Figure II). The response seen in Figure IV, then, should be transcribed thusly:

SAUL FRANKS%MANNIE ROBERTS%FRED BENSON#TECHNICIAN%TECHNICIAN%FOREMAN$

Note that after "FRED BENSON," only a "#" is used. To be complete, both a "%" and a "#" should be used as it is both the end of a subresponse and the end of a response type. However, that would be redundant. In this respect, the general transcription rule is WHEN TWO OR MORE DELIMITERS SHOULD BE EMPLOYED, EMPLOY ONLY THAT OF THE HIGHEST PRIORITY LEVEL.

This same rule applies after "FOREMAN" where the symbols "%," "#," and "$" have been collapsed to a "$" to signify the end of that response.

To further illustrate the transcription system, two more responses will be shown and transcribed as examples.

Example I:

Response: "Franks, Roberts, Benson, Smith"

Transcription: FRANKS#ROBERTS#BENSON#SMITH$

Note that with one response type, only the delimiter of the first level$_W$ is employed.

Example II:

Response:

"Franks	carpenter	yes
Roberts		no
Benson	foreman	yes"

Transcription: FRANKS:ROBERTS:BENSON#CARPENTER:X:FOREMAN#YES:NO:YES$

Note that ROBERTS does not list a position. Here this non-response is transcribed as an "X," but it could have been transcribed as

...#CARPENTER::FOREMAN#...

in which case the appropriate array cell would contain nothing, or what is known as the null string.[1]

In summary, the five general rules governing transcription with de-

[1] Because of the way in which STGPROC operates, it is a good idea not to employ the null string in this manner, but instead to use some constant symbol, such as an "X" for non-responses embedded in a list of subresponses, as is the case here. If the non-response comes at the end of a list of subresponses (if in this case "Benson" had no position associated with him, the null string would be an appropriate alternative.

limiter symbols are:

1. Responses shall be followed by the first priority level delimiter.
2. All information of one type is to be completely transcribed before beginning transcription of information of another type.
3. Data types are always separated by the first level_w delimiter.
4. Subresponses within response types are separated by the (number of response type) level_w delimiter.
5. When two or more delimiters should be employed, employ only that of the highest priority level.

The following section will concern itself with how the delimiter symbols are employed in respondent and response identifications.

Response and Respondent Identification

Transcription obviously takes up more space than the familiar type of coding. Instead of reserving one or two columns of a data card for a response, one may perhaps utilize forty spaces for a single response. Hence there is a need to identify each card of a respondent record as to where it belongs within that record. In STGPROC the first two columns of each data card are reserved for the card number. This allows one to identify up to 100 cards per record, a number which should be sufficient.[1]

In most coding systems, each record is identified by a number, called the respondent identification number. This system may be used with STGPROC if desired, however it does not necessarily have to be. Since iden-

[1] If it is not, instructions as to how to increase this number have been included in the comment cards of the STGPROC program (see Appendix One).

tification numbers are merely coded symbols, it may be easier in many cases to forget the coding and use the respondent's name as his identification "number." If this is done it obviates the processes of coding each respondent identification and of translating that coded identification. It further eases the task of identification of respondents while manually manipulating respondent records. The only drawback to this is that there is no sequential order in which respondent identifications can be aligned. If this is desired, a combination of the two methods of identification may easily be used.

In STGPROC the respondent identification string may be as long as the researcher wishes, and it may vary in length from respondent to respondent. The only stipulation is that it be separated from the record data by the first level delimiter symbol. THE RESPONDENT IDENTIFICATION STRING BEGINS IN COLUMN THREE OF EACH DATA CARD, CAN BE OF VARYING LENGTH PER RESPONDENT, AND IS FOLLOWED BY THE FIRST LEVEL DELIMITER SYMBOL. Examples of respondent identification strings (with card numbers preceeding them) follow:

1) 01023$

2) 01SMITH,FRED$

3) 01SMITH1$

4) 01023SMITH$

Note that examples two and three reveal alternate ways of identifying "Smith." Each "Smith" may be numbered (coded) or the entire name may be used to insure the uniqueness of the string.

There is one instruction card associated with the identification string. It specifies the maximum size of an identification string for that run. Its general form is:

27) MAXIMUM LENGTH OF ID STRONG = n.

where

n = the maximum length of an identification string for that run

In terms of the program STGPROC, it is easiest to number question responses from one up by increments of one. Within each respondent record the responses do not have to be in any special order, and they may vary from record to record, however each succeeding decimal digit should be employed as a response number.

RESPONSE NUMBERS ARE SEPARATED FROM THE RESPONSE BY THE FIRST LEVEL$_W$ DELIMITER SYMBOL. Thus (again utilizing the delimiter symbols of Figure II) two examples of the first and second total responses plus the respondent identification symbol would be transcribed in the following manner:

Example I:[1]

```
01FRANKS$1#SMITH#JONES$2#TEACHER%TEACHER#GRADE TWO%GRADE FOUR$
```

Example II:[2]

```
01THOMAS2$1#FRANKS:ROBERTS:BENSON#CARPENTER:X:FOREMAN#YES:NO:
YES$2#MANUAL LABOR$
```

[1] The questions asked of "Franks" in example one might have been: (1) Name those persons you interact with the most during school hours, and (2) for each person you have listed in question one, list their position in the school and what grade level they are associated with.

[2] The questions asked of "Thomas" in example two might have been: (1) List those you interact with most frequently on the job, their positions with the company, and whether you interact with them off the job as well (yes or no), and (2) list the word that best characterizes what type of work you do.

Keypunching

There are two situations in which keypunching is utilized with STGPROC: (1) when the instruction cards are created; and (2) when the data are being transcribed. A word concerning keypunching in each of these instances is necessary.

Instruction Cards

Some of the instruction cards have been previously mentioned. In keypunching these cards, only a few points must be kept in mind. First, every instruction begins with a number. After the number, there is a right parenthesis ")."[1] Second, each instruction ends with a period ".". Third, each instruction may occupy up to four succeeding cards (may be four IBM cards in length). Fourth, blanks may appear anywhere in the instruction (there is no fixed format for punching the instruction).

This fourth point bears commenting upon. Most computer programs require instruction cards. Many times, however, the instructions on these cards are in a coded form and must be located in specific fields of the cards. This means that the user must concern himself not only with translating the code the instructions appear in, but he must also be *absolutely* correct in his placement of these coded instructions upon the instruction cards. Neither of the above problems need worry STGPROC users. The instructions are written in English and, because of the fact that blanks are automatically compensated for, the instruction may be written anywhere

[3] The only exception to this rule is the case of the thesaurus cards, covered in Chapter Four.

on the card or cards--it may be stretched to its limit of four cards or compressed upon a single card. It is advisable, however, in the instructions containing lists (such as number 28 /̲the listing of delimiters_7) to leave some blank space between the items on the list. This enables one to change the contents of the list for later computer runs by changing only those items necessary and saves the chore of repunching the entire instruction.

The Data Cards

In preceding portions of this chapter, the transcription of data was explained. It has been found through experience with the STGPROC program that it is best in reference to the original keypunching, to leave blank spaces between questions, as well as between subresponses and response types.[1]

If blank spaces are left, then any character omissions may be corrected without having to repunch all the respondent's data from the point of the error to the termination of his responses. If the correction takes the form of character deletions, then the ensuing blank spaces are automatically compensated for. Further, if each question is begun upon a separate card, then later insertion of a response (as for example, in a longitudinal study) may take place without repunching the remaining data.

The character of the data can and should be used as one guide in keypunching. As with the instruction cards, this "blank space compensation"

[1] The program is now set up to handle question responses up to a maximum of 10 IBM cards per question response. The presently defined size limit is purely arbitrary, but should be extensive enough to meet most programming needs.

feature of the STGPROC program allows the user a flexibility in planning and executing his data transcription that no other computer program that the author is familiar with can approximate.

The System and Coding: Three Considerations

As previously mentioned, there exist instances of coding data prior to the use of STGPROC. These instances occur when the composition of the questionnaire has not been planned with the use of the STGPROC system in mind. In other words, all instances of human coding may be avoided by carefully planning the questionnaire in advance.

Obviously, the above is an ideal situation--in many cases questions receive unanticipated types of answers. Further, a previously developed interrogative may be utilized for comparison with a previous study in which it was asked. There are many such instances which lead to some prior coding of the data being necessary in order to get it in shape for use by STGPROC or in order to save time and space by taking advantage of some of STGPROC's unique features.

In this section, three instances suggesting the utility of prior coding that have arisen in connection with actual studies employing STGPROC are presented, along with viable solutions to the problems raised. By studying these examples, the researcher should be able to anticipate analogous problems in his own research and solve them in the questionnaire construction stage. Further, he should be able to derive solutions to similar problems that might arise as a result of his own unique research.

One: Irregular Interrogatives

The first instance of coding occurs when one is faced with an irregular interrogative. An irregular interrogative is here defined as any interrogative with differing numbers of response types per response choice. For example, the interrogative: "For each day of the week, state whether you drive to work, take commercial transportation, or are driven in a car pool. If driven in a car pool, list those with whom you ride."

In order to fit this interrogative into the STGPROC system efficiently, some coding has to be performed. Here there are actually two interrogative levels. The first and most general is "how do you get to work each day." The second, based upon the answer to the first, is "list those in the car pool per day you utilize this form of transportation." Clearly the response to the second level interrogative depends upon the answer to the first level and there is a possibility of differing numbers of response type in this interrogative per response choice upon the first level.

The most efficient means of storing this response is to code it as two interrogatives. The first level interrogative (assuming a five-day work week) is a 5 x 1 array, with possible responses consisting of "driving," "commerical transportation," or "car pool" for each of the five array cells.[1]

The second level interrogative response array may be set up in a num-

[1] See Figure IV and accompanying footnote for an explanation of arrays and subscripts. The reader is also referred to any introductory text on computer programming or any standard reference on matrix algebra for a more comprehensive treatment of the concepts of array and subscript.

ber of manners, depending upon the needs of the researcher. Efficiency in terms of storage space may be gained by the choice of an array, say 25 x 2, in which the first response type is a car pool member's name and the second is the array subscript of the first level interrogative pertaining to the cell involved. For example, if only on Thursday (the fourth day) the respondent rides to work in a car pool with Smith, then the coded response for the second level interrogative (including the response number) would be:

2#SMITH#4$

(Thursday being the fourth entry in the 5 x 1 array of the first level interrogative.)

In this manner, the first level interrogative could be handled separately if one wishes information pertaining to modes of transportation. If one wishes later to know about those in the car pool, the second level interrogative may be pulled out and response type two matched with the subscripts of the first level interrogative to determine days of the week the respondent rode with these persons.

It is a good rule of thumb when using STGPROC to TRY TO AVOID THE UTILIZATION OF IRREGULAR INTERROGATIVES. They only cause complications by the generation of human coding.

Two: Identification of Modes of Signification

Delimiters may be used to determine modes of signification of data. This too, is a method of coding the data. Suppose one were asked to list those tasks one performs at one's job, then in another interrogative, which of those listed tasks interfere with the performance of others. It would be a waste of space to transcribe these two interrogatives separately. Instead, a symbol may be used in the larger list to determine the *mode of signification* of the response (in this case, whether the listed task does or does not interfere with other tasks). For example, if a task is noted as interfering with others, its transcription in the larger list could include a "*" to denote that fact.

If only the interfering tasks were to be examined, a subprogram could be addended to STGPROC (much in the manner of subprogram SOCIO [see Chapter Four]) that would search all responses for a "*" and deal only with them.

Another manner of handling this type of problem would be to create an additional response type, and to place a "*" in each subresponse of this type that corresponds to each subresponse denoting an interfering task. This coding takes essentially the same form as that suggested above under "Irregular Interrogatives."

Conversely, in the car pool example above (Irregular Interrogatives) the second level interrogative could consist of a 25 x 1 array, each cell of which includes a name plus the array subscript of the first level interrogative pertaining to the cell involved. The example used above would now be:

2#SMITH4$

In this case, the addended subprogram would search for the symbols "1" through "5" to identify the subresponse as to which day of the week it pertained to.

Delimiters utilized to identify modes of signification then have as their function *the avoidance of repetition of transcripted information*.

Their use increases the efficiency of the STGPROC system.

Three: Key-word Searches

One of the most important aspects of STGPROC is its capacity to handle raw data by means of computer assignment of informational bits into any pre-determined category system. Thus many differing categorization schemes may be applied to the same data without having to resort to the tedious and inefficient task of recoding the data. The essential operation of the program here is to search the data for pre-determined key words or word-phrases. When one of these is identified in a match with a datum, a counter is activated, incrementing the appropriate category count by one. This process will be presented in more detail in Chapter Four. It is mentioned here only because of special problems encountered with certain types of raw data.

If one is analyzing responses to an occupational category interrogative for example, the key word search is a fairly straightforward process. The list of key word-phrases, called a thesaurus, consists of an array of word-phrases of only one type; i.e., nouns (and their adjectives) describing the occupation ("bank manager," "principal," "first-grade teacher"). However, if the interrogative were phrased in the following manner, the responses would be word-phrases of three types (subject, verb, object): "What do you do for a living?" Answers might range from "bank manager" to "I manage a bank."[1]

[1] One method of solving this problem of differing methods of presenting the same information is to limit the mode of response on the questionnaire itself. In this case, the respondents could be forced to answer

Here some coding can be employed to ease the key word search. One may alter the responses such that they all are word-phrases of one type. In this case, "I manage a bank" would be coded as "bank manager."

Or one could alter the responses such that they all are word-phrases of three types (subject, verb, direct object). Examples would include "I teach school," and "I manage a bank." In this case, thesauri for key word searches could number three, one for each type of key word-phrase. Later (Chapter Four) it will be shown how these types of thesauri may be combined in analysis. Of course in some instances one might not wish to include all word-phrase types in the analysis. For example, in the above case, all responses will have "I" as their subject. Hence, one would most probably delete this word-phrase type, as it makes no sense to categorize over a constant response.

Generally speaking, one should minimize the amount of coding performed on the data. It is better to have data of three types of word-phrases and never categorize on one type than it is to code it into two types and later decide upon a three type analysis.

It seems safe to say that in most situations, no more than four types of word-phrase will suggest themselves (subject, verb, direct object, indirect object).[1] One may however use as many word-phrase types as one deems necessary. This is the researcher's option, based upon his analysis

with a complete sentence, employing subject, object, and verb. This solution to the problem is, unfortunately, not always feasible, due to either the nature of some questions or the nature of some respondents.

[1] An exception to this is the case of facet analysis, which does not utilize grammar as its basis for word-phrase formation (see Chapter Four).

of the situation.

A word of caution is necessary here. There is a point at which the danger in information loss due to coding is offset by expense and the unavailability of coders sufficiently competent in grammar to handle multitype word-phrase codes. This problem has plagued programmers in many attempts at creating content analysis programs (cf. Stone, Dunphy, Smith and Ogilvie, 1966; pp. 67-206). STGPROC is not intended as a content analysis program. In its use in categorical analysis it does sacrifice some information for ease of comprehension and efficiency. The amount of information loss is determined by the type of interrogative under examination as well as by the coding decisions of the researcher.[1]

Coders and the Transcription System

It appeared that the new system of preparing data would likely have effects upon those hired to perform this task. Therefore close attention was paid to the persons working under this new system during the first six months of data transcription for the CASEA Attributes Projects. This transcription was the first instance of actually utilizing the STGPROC system.[2] Observations of the workers were noted and later cross-checked by means of informal interviews and a formal questionnaire. Because of the small number of workers (ranging from three to eight during the course of the job) and the lack of control groups, findings are not intended as

[1] A coding system actually utilized with STGPROC will be described in detail later in this manuscript.

[2] See Chapter Six for a description of the on-going data analysis being performed upon this transcribed data.

scientifically valid conclusions. Interesting results did emerge, however, and will be presented in the following discussion.

Although work was begun with a total of three women preparing data during the day and double checking their results each morning before going on to the preparation of new data, it was soon learned that this system was not viable. The women made many errors at first and used the double checking process to help establish their own informal "pecking order." Since other workers were soon to be introduced to the situation, it was felt that work would progress more smoothly by appointing two of these women as "double checkers" and physically removing them from the room, while at the same time introducing two new women to the business of data preparation. The double checkers were given their own room and told that any error in the data as finally recorded (less key punch errors) were their responsibility.

In this manner a status system was deliberately set up. The original worker not given the job of double checker was automatically in charge of the data preparation and all training of novices that this entailed. Although she still felt some resentment that others were chosen as double checkers, she had her own universe to control, and the novices kept her busy reaffirming her own position.

The double checkers group seemed to have no internal status problems, possibly due to their individual personality characteristics. They, however, felt acutely the added responsibility their position placed upon them.

Since this is not a focused small group study of interaction patterns, I shall refrain from commenting further upon specific problems and adjustments the workers encountered during their employment. I do feel it necessary however to point out that interactional patterns probably had something to do with the workers' reported job satisfaction. It is also important to note that these workers were aware that the data preparation system was a unique one, although they were not aware they were to be studied in relation to the system, they were well aware of their experience. Therefore, there may well be some "Hawthorne effect" built into worker responses. The author is aware of this and has attempted to temper the workers' responses with his own and others' observations of worker behavior during the six month employment period.

It was recognized that in the research process, those persons employed on the lower or less skilled levels, or persons who have nothing to gain or lose by the total research effort, or who do not operate within the professional ethos, may not be as intellectually conscientious in the performance of their jobs as might be hoped for.[1] Although human error will always be a factor in any human endeavor, it seemed probable that performance might be strengthened by a "tying in" of those lower level nonprofessionals to the research process. This was attempted with the project workers in two manners: (1) creating an awareness of the research process by explaining <u>why</u> things were to be done as they were; and (2) creating anxiety by delegating judgmental authority and responsibility.

Actually, it is more legitimate to say that the transcription system necessarily led to more judgmental decisions on the part of the workers

[1] From a conversation with Howard S. Becker in October, 1968; this assumption seems to have been generally accepted, mainly in the area of survey research. (See also Hanson, Robert and Eli Marks, 1958.)

and that these decisions, being machine checked, were always subject to outside verification. Further, the fact that this was a new system led to questions and a continuing interest in the research project. As the above factors were noted, along with the increased efficiency of worker performance, actual planning was effected to help perpetuate this psychological state.

Although the majority of the workers felt that the transcription system was no more difficult than standard coding systems they had been previously subjected to, they felt a good deal more anxiety in handling the new system. This was explained as having to do with the need to often use their own judgment, rather than look up answers in a codebook. Generally, they were all quite aware that their errors, both judgmental and clerical, would be detected. The workers seemed to enjoy taking on responsibility and professed satisfaction with their jobs, both in terms of individual achievement and in terms of on-the-job relations.

With the exception of one, the workers all felt that they learned more about the research process while on this project than while on any other project upon which they had worked. They understood the reason behind string processing, claiming in one case that not only is it good "not to put words in people's mouths" (referring to open-ended questions on the instrument) but also that in coding it is more legitimate to just transcribe the answer as it appears than to have to place it in a predefined codebook category. Two were specific on this point. They admitted great amounts of error in this "fitting-into-categories" type of coding.[1] Comments of this sort suggest a relatively sophisticated level of understanding of the research process and a concomitant interest in it.

Although it is impossible to calculate an exact error rate for the project, due to the type of data being transcribed, two things may be noted: (1) the error rate decreased sharply after the first two weeks of transcribing, then decreased more slowly as it reached quite low levels; (2) in the last set of data punched, out of 42 respondents there were 12 detected errors, one of which was definitely a key punch error. This, with an average of five and one-half data cards per respondent, is a figure to marvel at.

In conclusion, let me state that those who worked with the transcription system seemed tied into the research process more. They were interested in what they were doing and anxious not to make mistakes. With the double checking method, few errors were actually made, anxiety was kept at a high but tolerable level (especially for the double checkers), and job satisfaction and morale seemed high. A last and certainly important point: the workers generally did not feel as though the transcription system was any more difficult to handle than normal coding systems.

In setting up a work situation, allow me the following recommendations: (1) a training session in the use of the system and why it is being used to be immediately administered; (2) the analogy of a code book be supplied each worker, with examples of how each question is to be transcribed; (3) a double checking system be set up, preferably with

[1] This supports Sussman and Haug's finding of up to 18 per cent error rate in this type of coding. They also report the results of another study in which up to 20 per cent error was detected. (Sussman, Marvin and Marie Haug, 1967)

double checkers physically isolated from transcribers; (4) a supervisor be nearby as much of the time as possible to answer any questions that might arise. These recommendations do not vary much from standard coding instructions; however, if they are not followed, complications are bound to arise. Although many errors in standard coding will not retard the physical research process (even as they cause unknown bias in results), with the transcription system much of the same type of error will halt the research process at the machine stage until it has been corrected. Thus it behooves the researcher to be as meticulous as possible in the early stages of data transcription. Care will pay off in ease of data manipulation later in the research process, as well as in assurances as to the validity of final reported results.

Summary

This chapter began by making the basic distinction between coding and transcription. Coding was defined as the process of resymbolizing information, while transcription was defined as the transference of a set of symbols from one medium to another. It was pointed out that coding could be made more efficient if performed by computer and that STGPROC was a program which would allow this.

In turning to the process of transcription, the concept of delimiter symbol was introduced and defined as any symbol that divides or marks symbolic information at points significant to the ordering of this information such that it may be comprehended by one familiar with the utilized symbol system. It was noted that no symbol likely to appear in content-informational symbol strings should be used as a delimiter symbol. Finally, the form in which delimiter symbols are to be introduced to STGPROC was covered and an example was given.

The concepts of response type (types of information given) and subresponse (information bits of the same type) were covered. The application of these concepts to STGPROC was discussed and examples were given.

At this point the transcription system itself was introduced along with five general rules governing the transcription process. Examples were presented as well as the actual transcription of these examples, utilizing the five rules of transcription.

After a discussion of the keypunching of instruction cards and transcribed data, three special considerations of the system were identified: (1) irregular interrogatives; (2) identification of modes of signification; (3) key word searches. The chapter concluded with a discussion of actual coder reactions to the transcription system and suggestions as to means of setting up this system effectively.

At this point the researcher should be able to plan his data collection with an eye to transcription, as well as being able to set up both the transcription process and the work situation. In the next chapter, the main program STGPROC will be presented. At its conclusion the researcher should be able to perform a data run through STGPROC, utilizing the data he has learned to transcribe by study of the present chapter.

CHAPTER THREE

THE MAIN PROGRAM

The STGPROC main program is reproduced in Appendix One along with comment cards and statement card numbers to aid comprehension. When reference is made to the main program, statements will be identified by statement card numbers.

The function of the main program is to: (1) assign values read from the instruction cards to the variables and storage arrays necessary for the program run; and (2) read the transcribed data and to arrange it in an order such that data manipulation and automatic coding may be performed. In order to accomplish this, all aspects of the data must be first checked in order to alleviate: (1) transcription errors; (2) the mis-ordering of data (if it appears on cards); (3) programming error in reserving array space for the data (see Chapter Two). Upon the researcher's signal, any or all of these aspects can be checked.

Once the above checks have been made, one can be certain that the only errors remaining in the data are spelling errors. These errors are automatically checked for in the manipulative subprograms. Hence one may deactivate the check procedures after a first "error run" through the data (in order to save programming time) and activate the desired manipulative procedures for analysis.

Before describing the main program in more detail, I shall explain: (1) the concept of <u>procedures</u>, and (2) the working of the PL/1 string functions. This is necessary in order for the researcher to understand in general how the program works and specifically how to adapt new procedures to it.

Procedures

In general, a procedure is a set of programming statements designed to accomplish a particular objective. The objective of a program as a w whole (the main objective) is successful program execution. However, there are intermediate results also to be performed. These results can be obtained directly by statements in the main program or they may be obtained by the execution of other programs subordinate to the main program. Such subprograms are known as procedures. (The main program is also a procedure--in this case the procedure STGPROC. However, other procedures exist in the total program, subprograms such as EPROR_CHECK which obtain specific intermediate objectives for the larger main program.) Thus a PL/1 program consists of two kinds of procedure; a single main procedure and any number of subprograms (or subroutines), each of which is also a procedure.

During the course of program execution, the main procedure may "call" subprograms, and the subprograms may "call" other subprograms. "Calling" a subprogram is the process of transferring control to it while suspending execution of the main program. When the subprogram has obtained its intermediate results, control is returned to the main program which then resumes execution at the point following the calling statement. A subprogram may be called as many times and from as many places in the main program as required.

In terms of the program STGPROC (see Appendix One) there exist one main procedure and 12 subprograms (this does not include the five manipulative and statistical subprograms to be discussed in Chapters Four and Five). These procedures are arranged in the order shown in Figure V. Figure VI shows how the procedures are called, rather than how they are physically assembled in Appendix One.

In examining Figure VI, it is evident, for example, that PRGM calls BREAK and that BREAK calls ERROR_CHECK, DATA_PRINT, AND PROC_CALL. Further, PRGM also calls PROC_CALL which in turn calls the manipulative and statistical procedures. A procedure may call any other procedure that is entirely included within it, or it may call any upon the same "level" as itself. However BREAK could not, for example, call STGPROC.

ERROR_CHECK, by activating the ERROR RUN option, may be called to check data errors. DATA_PRINT, by activating the PRINT RUN option, may be called to output all data for observation. If an error run has already been accomplished, then PROC_CALL may be called to activate the manipulative and statistical procedures. SUB, by activating the SUBSECTION option, may be called to output the totals of all data stored in the manipulative and statistical subroutines up to that point and to initialize all storage values to zero in anticipation of building a new set of totals with the following data. Finally, the activation of procedures in STGPROC is controlled by the insertion of instruction cards in the program. The instruction cards will be covered later in this chapter.

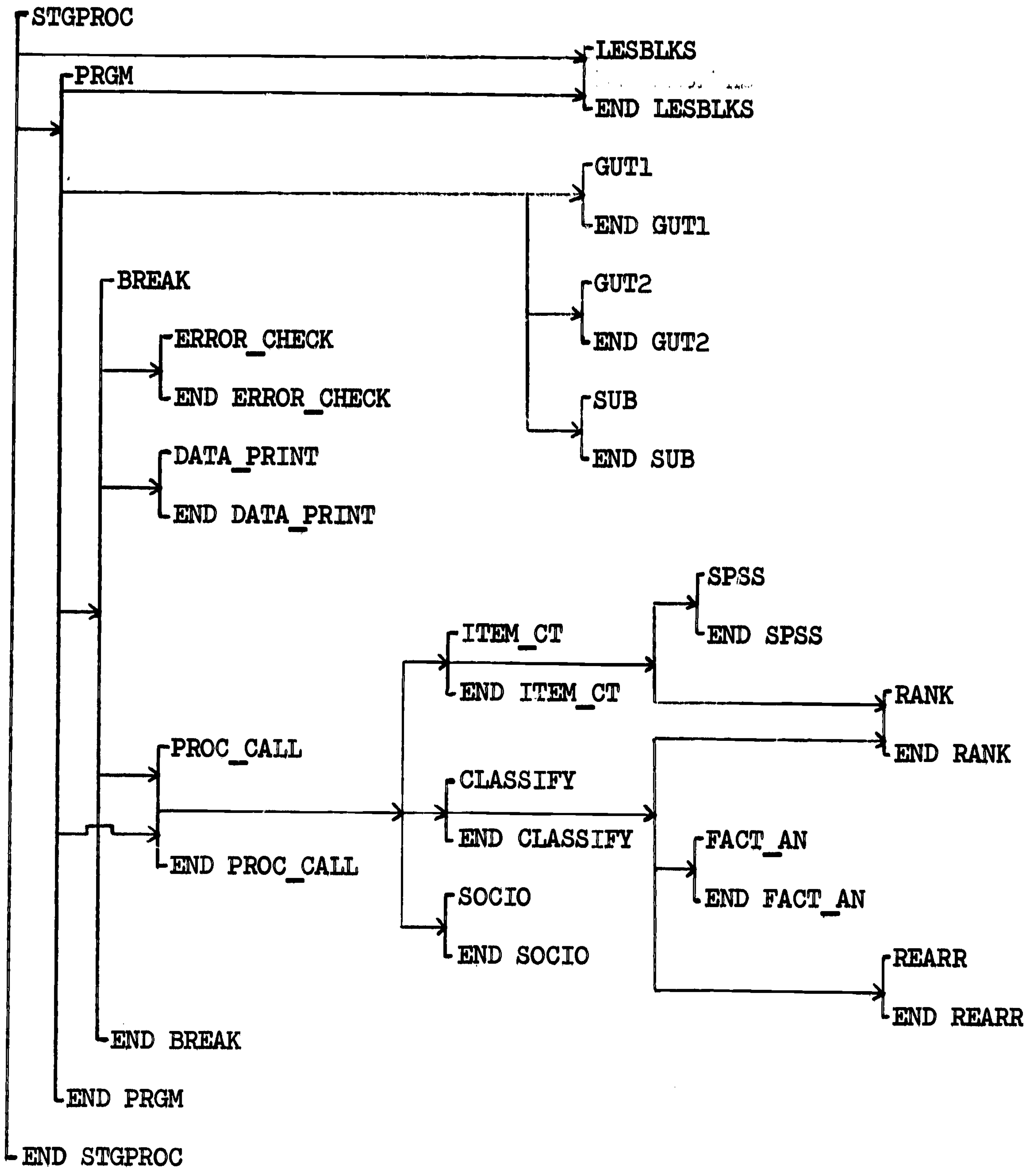
The String Functions

In general, to get a flavor of why string processing is an advance

Figure V: Arrangement of STGPROC Procedures (A)

```
STGPROC
      LESBLKS
      END LESBLKS
   PRGM
      REARR
      END REARR
      GUT1
      END GUT1
      GUT2
      END GUT2
      GET
      END GET
      PROC_CALL
      END PROC_CALL
      SUB
      END SUB
      RANK
      END RANK
      Manipulative and Statistical Subroutines
      BREAK
      END BREAK
      ERROR_CHECK
      END ERROR_CHECK
      DATA_PRINT
      END DATA_PRINT
   END PRGM
END STGPROC
```

Figure VI: Arrangement of STGPROC Procedures (B)



in computer technology, let me discuss briefly how the string functions of the PL/1 programming language work. There are three essential functions. One (LENGTH) determines how many characters a string contains. INDEX can match any string pattern with any other and return a value. If the value is zero, the match was unsuccessful. If the value is greater than zero, it is the position in the string in which a match is being sought in which the pattern begins. For example, if one wished to determine if 'ECONOMIC' were in the string 'SOCIAL AND ECONOMIC CONSEQUENCES', one would use the INDEX function thusly:

```
X = INDEX ('SOCIAL AND ECONOMIC CONSEQUENCES', 'ECONOMIC');
```

The match would succeed, and X would equal 12.

The last function to be discussed is SUBSTR. This function will break up a string in any way specified, creating a new string. For example, if one wished a string containing 'ECONOMIC' and had a string containing 'SOCIAL AND ECONOMIC CONSEQUENCES' one would utilize the SUBSTR function:

```
T = SUBSTR ('SOCIAL AND ECONOMIC CONSEQUENCES', 12, 8);
```

Here, the "12" indicates the beginning position of the new string and the "8" signifies the length of the new string in characters. In this case, one wished the new string to begin at position "12" in the base string and extend "8" characters in length.

All the functions can now be put together in a small program segment to illustrate their use:

```
Q = 'ECONOMIC';                    Q contains the string 'ECONOMIC'

R = 'SOCIAL AND ECONOMIC           R contains the string 'SOCIAL AND
     CONSEQUENCES';                  ECONOMIC CONSEQUENCES'
```

```
A = LENGTH (Q);                  A contains '8' or the length of
                                   the string 'ECONOMIC'

X = INDEX (R,Q);                 X contains '12'

T = SUBSTR (R,X,A);              T contains 'ECONOMIC'
```

This illustrates the power and generality of the string functions. It should now be more clear why with PL/1 string processing, it is a relatively easy task to match responses, creating sociometric matrixes for statistical analyses, as well as creating category systems based on researcher-compiled thesauri which can be manipulated to the researcher's content.

Instruction Cards: The Main Program

In applying the main program, the researcher need only concern himself with 14 instruction cards.[1] Seven of these instructions have been covered in Chapter Two. They determine the number and size of the responses to be found in the data as well as the types and levels of delimiters utilized. The seven remaining instruction cards determine whether or not five specific options are active during the program run. To activate an option, the appropriate instructions are included in the instruction card set. If the option is to remain passive, the instruction cards

[1] Every computation center has its own type of program-independent job control cards which need accompany any job submitted to the computer. In addition to the instruction cards, the researcher has to also deal with job control. Appendix Four offers the example of STGPROC job control as presently employed at the computation center of the University of Oregon. This job control will vary, however, and the researcher is urged to seek consultation upon this matter before attempting his first program run.

are not included. The general forms of the seven instruction cards are:

4) ACTIVATE PRINT RUN.

5) ACTIVATE ERROR RUN.

6) ACTIVATE SUBSECTION RUN.

7) ACTIVATE SELECT-IN RUN, NUMBER OF RESPONDENTS = n.

44) SELECT-IN RESPONDENT ID'S = s_1 , s_2,, s_n .

8) ACTIVATE SELECT-OUT RUN, NUMBER OF RESPONDENTS = n.

43) SELECT-OUT RESPONDENT ID'S = s_1 , s_2 , s_n .

where

n = the number of respondents

s = identification string

After having included those of the 14 instruction cards deemed necessary, and having transcribed and keypunched the data, the researcher is ready for a program run. Only one more thing is needed. The final data card must always look as follows:

01ENDfirst-level-delimiter

Or in the continuing example of delimiters utilized in this monograph:

01END$

The above characters begin in the first column of the data card. The "01" signifies that it is the first card of a new record. The "END" signifies that all the data have been processed. When the identification string is "END," control is transferred to the end of STGPROC. If manipulative subroutines are involved, totals are then acted upon and output. I am, however, not concerned with the manipulative subroutines at this stage in the explanation. These subroutines will be dealt with in the following chapters.

All operations necessary to activate the main program have now been covered. The researcher may choose whether he wishes to activate PRINT RUN and/or ERROR RUN in conjunction with his first main program run; however, at this stage, activation of both is the most logical course. This creates the first data run, checking on all possible errors in the data and printing the data itself in a recognizable form.

STGPROC Operations

Some persons are curious and others already know what goes on inside the computer. In either event, an attempt at explaining every step in the STGPROC program would only be tedious and time consuming. Those who know programming languages may follow the process upon the program and comment cards reproduced in Appendix One. For others, the general explanation below of how the program deals with a data card should prove illuminating. In any case, one does not need be a mechanic in order to drive an automobile (especially if a mechanic may be easily summoned).

Let us assume the following data card is being processed by STGPROC:[1,2]

```
01JONES1   $   1#FIELDS   #THOMPSON   $2   #   TEACHER   $
```

[1] Prior to this step the program has read the instruction cards and inserted the given values in the appropriate variable and array declarations; hence it is now ready to process a data card. It should be noted that the instruction cards are processed in essentially the same manner as the data cards. A general comprehension of this process will encompass both instances.

[2] This data card has the following information upon it: (1) This is the first card of the record; (2) the respondent is "JONES1"; (3) the responses to question one are two subresponses of the same response type, "FIELDS" and "THOMPSON"; (4) the response to question two is one subresponse of one response type, "TEACHER."

The following are steps taken to break up the above string.[1] The discussion is general and does not include the many nuances built into the program to account for all logical variations upon structure, many of which were discovered only through research experience. In this instance, simplicity serves the desired purpose.

A. The above string of characters is "read" and assigned to the variable CSTG. CSTG is the variable that holds in turn each respondent's data until it is completely processed by the program--the variable performs a memory function:

```
CSTG = 01JONES1    $    1#FIELDS    #THOMPSON    $2    #    TEACHER    $
```

B. The card continuation number is assigned to the variable CONT_NUM:

CONT_NUM = 01 (the first two characters of CSTG)

C. The identification string is assigned to the variable TD and the blanks are automatically removed:

TD = JONES1

D. If CONT_NUM equals "01" then ID equals TD. This two-step is performed because of the probability of more cards than one existing in the same record. It aids in determining if all the identification numbers in a record are the same or (if they differ) that the cards are out of order.

ID = JONES1

E. CSTG is now assigned to TSTG, in order to free CSTG for data from the next data card. The continuation number and the identification string are left behind on the transfer.

[1] These steps correspond roughly with statements 941-1014 of STGPROC (see Appendix One).

TSTG = 1#FIELDS #THOMPSON $2 # TEACHER $

F. QUES is formed. This is the question string and includes all characters up to the next first level delimiter. This first level delimiter ($) in QUES is changed to a second level delimiter (#) for purposes of later processing.

QUES = 1#FIELDS #THOMPSON #

G. TSTG now contains what is left after the removal of QUES.

TSTG = 2 # TEACHER $

H. The question number is assigned to the variable QNUM1, blanks are automatically removed, and QNUM1 is set equal to QNUM2 (necessary because of conversion procedures).

QNUM1 = 1

QNUM2 = 1

I. QUES now contains what is left after the removal of QNUM1.

QUES = FIELDS #THOMPSON #

J. The program is now ready to call the subroutine BREAK to break up the string QUES[1] and lodge its componant parts in the appropriate cells of QUES_ARRAY.[2] At the same time, if ERROR RUN is active, it will check QUES for any possible delimiter errors.

QUES_ARRAY (1,1) = FIELDS

QUES_ARRAY (2,1) = THOMPSON

K. When all operations are performed, control returns to step E,

[1] All leading or following blanks in the data are automatically removed at this point in the program.

[2] QUES_ARRAY is the array created by the program to store the questions broken down by subresponse and response type.

where TSTG is now:

TSTG = 2 # TEACHER $

L. Step F then results in:

QUES = 2 # TEACHER #

M. Step G then results in:

TSTG = '' (or the null string)

N. Step H then results in:

QNUM1 = 2

QNUM2 = 2

O. Step I then results in:

QUES = TEACHER #

P. BREAK is again called and the string QUES is broken up into QUES_ARRAY in the appropriate manner:

QUES_ARRAY (1,1) = TEACHER

Q. At this point there is no data left in TSTG. The next card is read into CSTG. If its continuation number (CONT_NUM) is numerically larger than the preceding continuation number (and the identification strings are the same) then work continues upon the same record. If the continuation number is numerically smaller than the preceding continuation number (and the identification strings are the same) then an error message results (as the cards are out of order). If the continuation number is "01" (and the identification strings differ) then the previous data are printed and/or stored as directed, and the process begins again for the next record.

PRINT RUN

If PRINT RUN is activated, it will automatically introduce its output by means of a heading page with the following printed upon it:

FOLLOWING PAGES ARE PRINTOUT OF QUESTION RESPONSES

If PRINT RUN is in operation, the following type of output will result, one page skipped between each respondent. For the example presented above, if that were all the data for JONES1, the output would be as follows:

```
QUESTION # 1:
   QUES_ARRAY (1,1) = 'FIELDS'
   QUES_ARRAY (2,1) = 'THOMPSON'
QUESTION # 2:
   QUES_ARRAY (1,1) = 'TEACHER'

   ID = 'JONES1'
```

This type of output gives one a check on just what the data look like in a raw form, as well as being helpful in detecting spelling errors that may have crept in. PRINT RUN is probably best utilized in conjunction with the ERROR RUN option.

ERROR RUN

If ERROR RUN is activated, it will automatically introduce its output by means of a heading page with the following printed upon it:

THE FOLLOWING PAGES ARE ERROR CHECK ON DATA

There are two major types of error that this subroutine checks upon. The first is whether the researcher has, by means of the instruction cards, set aside enough space in STGPROC's working arrays to hold the data being processed. The second is whether the transcription of the data conforms to the rules implicit in the researcher's setting up of delimiters in the appropriate instruction cards.

Errors of Type One

There are two basic error messages subsumed under this type:

A. Dimensionality. The error routine will print out the number of the question upon which this error arose and the present size of the "delinquent" dimension. It will then instruct the researcher to increase the size of this dimension as it appears in instruction card 29.[1]

Output Example:

```
ERROR: IN QUESTION # 2 THERE IS RESERVED SPACE FOR ONLY 5 SUBRE-
SPONSES. THIS NUMBER HAS HERE BEEN EXCEEDED. INCREASE THE APPRO-
PRIATE VALUE IN 'NUMBER OF SUBRESPONSES' INSTRUCTION CARD # 29.
```

B. Character String Size. The error routine will print out the number of the question upon which this error arose and give appropriate instructions as to how to correct it.

Output Example:

```
ERROR: QUESTION # 3 CONTAINS ONE OR MORE STRINGS LONGER THAN THE
```

[1] If ERROR RUN is not activated, no error will ensue and any further subresponses than there are spaces allowed for will be dropped from consideration. In this way, should the researcher wish to process only a portion of a response (for example, the first three subresponses of the fourth question), then he need only set up the instruction card "SUBRESPONSES PER QUESTION" to handle three subresponses for the fourth question, and only those three will be processed. This saves computer time as well as creating more flexibility in the manipulative process.

LENGTH SPECIFIED. INCREASE SIZE OF 'RESPONSE LENGTH' IN INSTRUCTION CARD # 3.

Errors of Type Two

A. Wrong Delimiters Used. The error routine will identify the question, the erroneous delimiters and their correct replacement.

Output Example:

ERROR: DELIMITER % WAS USED IN QUESTION # 3. REPLACE WITH DELIMITER #.

B. Missing Delimiters. The error routine will identify the question and the type of missing delimiter.[1]

Output Example:

ERROR: IN QUESTION # 3 ONE OR MORE DELIMITERS OF TYPE # ARE MISSING.

C. Merged Questions. This is a special case of missing delimiters. When the delimiter between questions is missing, it creates the grave problem of a machine attempt to fit two questions into QUES_ARRAY while the array specifications are set for only the first of the two merged questions. This error routine will identify the "left-hand" question in the pair and give the proper delimiter and its placement.

[1] This routine also lists (previously to the error message) the number of delimiters used per response type, so that the user can immediately identify the delinquent response type. For example, if a delimiter was missing from the third response type, the message might appear as follows:

```
NUMBER OF DELIMITERS PER RESPONSE TYPE:

COT(1) = 12    COT(2) = 12    COT(3) = 11
```

Output Example:

ERROR: TWO QUESTIONS MERGED: QUESTION # 3 AND THE FOLLOWING QUESTION. DELIMITER $ SHOULD PROCEED THE SECOND QUESTION THUSLY:

CORRECTED STRING: $4#EXAMPLE#

D. Response Types. A special case of the above. If too many response types (see Chapter Two) appear in a question string, it may in some cases be interpreted as a merged question error. Hence, for each instance of the above (C) error message, the following also appears.

Output Example:

IF CHARACTERS IMMEDIATELY TO THE RIGHT OF $ IN THE CORRECTED STRING DO NOT SIGNIFY A QUESTION NUMBER THEN ERROR IS THAT OF TOO MANY RESPONSE TYPES IN QUESTION # 3 FOR THE NUMBER OF TYPES ALLOWED FOR IN 'RESPONSE TYPES PER QUESTION' INSTRUCTION CARD # 30.

E. Too Many Delimiters. If there are too many delimiters of the correct type, then dimensionality will be violated and the appropriate error message (as covered above) will be generated.

The above are the types of errors covered by ERROR RUN. To the best of the author's knowledge, outside of spelling errors, they are the only error types not automatically machine corrected or flagged, that can occur.[1,2]

In conjunction with PRINT RUN, these error messages will appear upon

[1] Machine flagged errors would include problems in compiling a new set of faultily punched program cards or incorrect job control cards.

[2] There is one other type of error, briefly mentioned beforehand. STG-PROC automatically determines that the data cards are in correct order before commencing processing them. If they are not, appropriate error messages to that effect are generated.

the appropriate data printout page, along with the respondent's ID string.[1] Hence it is an easy task to locate the data card upon which the problem appears (if that is the case) and correct it. In any case, the appropriate corrective action is either specified or implied.

SUBSECTION

If SUBSECTION is activated, it will automatically introduce each section of respondent data defined as complete by the researcher with a heading page with the following printed on it:

NEXT SUBSAMPLE

The researcher defines a section of data as complete (as a subsample of respondents) by enclosing it within blank IBM cards. If SUBSECTION is active, the subroutine will trigger the totalling and output functions of the manipulative and statistical subroutines for the data STGPROC has processed up to the reading of a blank IBM card. It then will initialize all STGPROC storage variables and arrays to zero and commende building another data set with the material following the blank card. SUBSECTION is best utilized in conjunction with the manipulative subroutines, to create subsample totals (such as respondent data totalled by each school studied). Then, by deactivating SUBSECTION on a program run, one can arrive at the grand totals (totals by all schools studied).

[1] If PRINT RUN is active, the error messages will appear along with each respondent's printout of data in correct format. If PRINT RUN is not active, then the printout will consist solely of the ID strings of those respondents in whose data there were detected errors plus the appropriate error messages.

SELECT-IN and SELECT-OUT

Each of these options is the obverse of the other--each concerned with a specified subset of the data. If SELECT-IN is activated, only those respondents whose identification strings are listed in instruction card number 44 will be processed by STGPROC on that program run. If SELECT-OUT is activated, those respondents whose identification strings are listed in instruction card number 43 will be the only ones not processed by STGPROC on that run.

These two options make it possible to process many differing subsets of the data without having to physically manipulate the data cards (or tape) at all. They add great flexibility to the program--taken in conjunction with SUBSECTION they allow the processing of any specific respondent or combination of respondents whatsoever from a single entry up to the total sample size.

An example of the instruction cards necessary to activate these runs follows:[1,2]

8) ACTIVATE SELECT-OUT RUN, NUMBER OF RESPONDENTS = 3.

43) SELECT-OUT RESPONDENT ID'S = JONES1, SMITH4, SMITH', 'ANDREW.

The Instruction Cards

A great deal has been said already concerning the instruction cards,

[1] The instructions for SELECT-IN are set up in exactly the same fashion.

[2] Note that if the symbols equals (=), comma (,), or period (.) are a part of any identification string listed, they are to be enclosed in single quotes in the ID lists. In this case, the third entry in the ID list in instruction number 43 is actually "SMITH,ANDREW," but is entered as it appears above.

yet a further word of explanation is necessary at this point.

The options the researcher wishes to include in his STGPROC run, and the values he wishes to give to the variables attending that run are bits of information that have to be entered into the STGPROC program before it begins processing the respondent data. The way this information would be entered by a computer programmer would most probably be by manually modifying statements in the program itself. However, this method is tricky. Cards in the program might accidentally get shuffled. Or the entries (especially those of novice programmers) might not be in their correct coded form or might not be complete. Hence it would be greatly desirable if there were no need of altering the program at all.[1]

The instruction cards are a means to the above end. They are entered as data to the program and are processed prior to the processing of the respondent data. Their function is to specify the values necessary for the STGPROC run, and to do so in a fashion that is clear and unambiguous to the user.

It should be noted that each instruction begins with a number (excepting the thesaurus cards--which are only portions of a numbered instruction [see Chapter Four]). As the instructions are logically presented in this monograph, it is the case that these numbers are not in numeric order. This is because of the way in which the cards are to be processed by STGPROC. However, prior to submitting the instruction cards, they should be manually arranged such that they *are* in numeric order. This is a most important step. STGPROC will not operate successfully if this is

[1] Except to enter new subroutines, which should be done by an experienced programmer.

not done.

There are a total of 46 instructions (reprpduced in Appendix Three). Any or all of these may be included in a STGPROC program run (however, it will seldom, if ever, be the case that all the options will be utilized during a single run). The instruction cards are entered in a data file labelled "IN." The respondent data are entered in a data file labelled "INDATA." Instructions for entering the data in these fdles are given in Appendix Three.

Summary

This chapter has attempted to explain the STGPROC main program with its subroutine options PRINT RUN, ERROR RUN, SUBSECTION, SELECT-IN, and SELECT-OUT in such a manner that the researcher will be able to utilize the program. At this point, he should be ready to transcribe his data (Chapter Two) and set up a STGPROC program run for an error check and data printout (Chapter Three).

The researcher should also be aware, in a general sense, of how the STGPROC main program manipulates data, breaking it up to fill the appropriate cells of QUES_ARRAY. The first part of Chapter Four will illustrate how sample data would look stored in QUES_ARRAY, as this concept is integral in understanding how the manipulative subroutines function.

In brief, to operate STGPROC, the researcher must determine the form of fifteen instruction cards.[1] These cards take the following general

[1] The last instruction card simply signals the end of the instruction card set.

forms:[1]

1) NUMBER OF DELIMITERS = n.

2) NUMBER OF QUESTIONS = n.

3) RESPONSE LENGTH = n.

4) ACTIVATE PRINT RUN.

5) ACTIVATE ERROR RUN.

6) ACTIVATE SUBSECTION RUN.

7) ACTIVATE SELECT-IN RUN, NUMBER OF RESPONDENTS = n.

8) ACTIVATE SELECT-OUT RUN, NUMBER OF RESPONDENTS = n.

27) MAXIMUM LENGTH OF ID STRING = n.

28) IDENTITY OF DELIMITERS = d_1 , d_2 , d_n .

29) SUBRESPONSES PER QUESTION = s_1 , s_2 , s_n .

30) RESPONSE TYPES PER QUESTION = t_1 , t_2 , t_n .

43) SELECT-OUT RESPONDENT ID'S = s_1 , s_2 , s_n .

44) SELECT-IN RESPONDENT ID'S = s_1 , s_2 , s_n .

46) END OF INSTRUCTION CARDS.

The researcher must determine the correct job control (see Appendixes Three and Four), and finally, he must be sure to place a data card with the following characters upon it (beginning in column one) at the end of his data:

01ENDfirst-level-delimiter

[1] Instructions one through three and 27 through 30 are explained in Chapter Two. Statements four through eight and 43 and 44 are explained in Chapter Three.

At this point the researcher is ready to run the error check upon his data, getting it into shape for use with the manipulative subroutines. Chapter Four presents three major types of manipulative subroutines, gives working examples of each, and offers suggestions as to further possibilities. Chapter Five presents the means of converting STGPROC results into punched card data for use with statistical and factor analysis programs as an integral part of its discussion of statistical subroutines and their relation to STGPROC.

CHAPTER FOUR

THE MANIPULATIVE SUBROUTINES

As has been previously stated, many numbers of manipulative subroutines may be written for STGPROC. By a manipulative subroutine, the author is referring to a subroutine that deals with raw data, arranging it in some logical order. This means that data output from manipulative subroutines will be in the form of listings--be they categories, frequency counts, or rank orderings. Manipulative subroutines do not concern themselves with statistical analysis. Their function is to prepare the raw data--to manipulative it and shape it to forms amenable to analysis. Chapter Five will discuss modes of statistical analysis of STGPROC data. The present chapter is concerned with manipulative subroutines only.

There are three general forms of manipulative subroutine to be discussed: (1) those in which the response itself determines the data manipulation; (2) those in which the researcher supplies the information which determines data manipulation; (3) those in which the response to another specified question determines the data manipulation.

Each of these general types of data manipulation will be discussed, and an example of an operative STGPROC subroutine will be presented as illustration for each type. The subroutine examples, along with comment cards, are reproduced in Appendix One of this monograph. These subroutines may be utilized as they appear, or may be easily modified by one with a modicum of programming experience. Alternatively, new subroutines may be created to perform further data manipulation deemed significant by the researcher but not covered in this monograph.

The Storage of Data in QUES ARRAY

As should be evident from the explanation of how the main program STGPROC operates (Chapter Three), data from each question is stored in the appropriate cells of QUES_ARRAY.[1] This is the key to the operation of the manipulative subroutines--each datum is identified by row and column subscripts of QUES_ARRAY.

The size of QUES_ARRAY varies for each question and is defined by the researcher by means of instruction cards 29 and 30. QUES_ARRAY contains both the maximum number of rows and columns of the response to be processed (Chapter Two). Figure IV in Chapter Two could well be a representation of QUES_ARRAY for some question in a program run. This figure is here reproduced in a slightly different form.

Figure VIII shows a QUES_ARRAY with two columns and six rows. Note that there is room for up to six subresponses, but that in this case the respondent listed only three subresponses. With the data in this form, it is a simple task to write manipulative subroutines to process it. Whether the response cell is of response type one or two is determined by the column subscript of QUES_ARRAY. In the same manner, the number of the subresponses can be determined by the row subscript of QUES_ARRAY.

[1] QUES_ARRAY is an array, set up within the computer memory by the STGPROC program, to store question responses, one at a time, in a manner that allows them to be processed by the activated STGPROC program subroutines.

Figure VII: Example of Data Stored in QUES_ARRAY

	1	2
1	SAUL FRANKS	TECHNICIAN
2	MANNIE ROBERTS	TECHNICIAN
3	FRED BENSON	FOREMAN
4		
5		
6		

Any manipulative subroutine can thusly keep perfect track of each datum, having automatically indexed it for reference. Incidentally, when the subroutine searches QUES_ARRAY for data, it saves time by discontinuing its search when encountering null strings (empty cells). Thus in the above example, a data search for subresponses would halt when row four was encountered.[1]

In conclusion, the general procedure in terms of QUES_ARRAY is:

A. The array is created by the program of a size to handle the spedified

[1] This is the reason that it is not a good idea to employ null entries in the middle of a subresponse list. The manipulative routineswill take it as a sign of the completion of the subresponse list for that data type and will not process further subresponses. Null entries are advisable if placed at the end of subresponse lists, however, to save computer time (see Chapter Two).

response.

B. Each cell of QUES_ARRAY is set equal to the null string.

C. The data for one response is lodged in QUES_ARRAY.

D. If a manipulative subroutine is active in conjunction with this response, QUES_ARRAY is searched for the desired data, which is transferred to the special storage of the subroutine in question.

E. If PRINT-RUN is activated, the data are printed as output.

F. QUES_ARRAY is collapsed.

G. The array is created of a size to handle the next specified response.

Manipulative Subroutine: Type One

This first type of manipulative subroutine concerns itself with categorization supplied by the question response itself. Essentially, given a question with a multiple number of subresponses per respondent, this type of subroutine yields a simple frequency count of the occurence of each unique subresponse. This type of routine is especially useful in exploratory research when one has asked respondents to list as many items as the respondent wishes from a universe of items unknown to the researcher.

ITEM CT Option: Activation

ITEM_CT (see Appendix One, STGPROC statements 153-244) is the STGPROC example of the above type of manipulative subroutine. It can be set up to independently manipulate as many questions as the researcher wishes on the same data run. It also will perform upon whatever response type within each question that the researcher specifies. Thus it is conceivable that ITEM_CT could be activated to perform upon every response type of every question in the data during one data run.

There are five instruction cards that control the use of ITEM_CT. Their general form is as follows:

9) NUMBER OF ITEM_CT RUNS = n..

10) ITEM_CT LENGTH OF RESPONSE = l .

11) ESTIMATE OF MAXIMUM NUMBER OF DISCRETE ITEMS IN AN ITEM_CT RUN = e .

31) ITEM_CT QUESTION NUMBERS PER RUN = q_1 , q_2 , q_n .

32) ITEM_CT RESPONSE TYPES PER RUN = t_1 , t_2 , t_n .

where

n = number of instances of the application of the subroutine to the data in a program

l = the character length of the largest datum to be processed

e = the estimate of the size of the largest list of discrete items created by the subroutine

q = the question number of the data the subroutine is to be applied to

t = the response type of the data the subroutine is to be applied to

For example, suppose one wished to produce two listings of items in a STGPROC run by means of ITEM_CT:

A. The character length for the first listing is 20 and the second is 30.[1]

B. The first listing is of those data found in question two, response type four. It is estimated there will be 50 discrete items found.[2]

[1] These figures can easily be obtained by use of the table (shown in Appendix Two) that lists the size of the character strings per question.

[2] Estimates are best if they are generous.

C. The second listing being of those data found in question six, response type one. It is estimated there will be 80 discrete items found.

In order to activate ITEM_CT for these two runs, the following specific forms of the instructions must be inserted in the instruction card set:

9) NUMBER OF ITEM_CT RUNS = 2.

10) ITEM_CT LENGTH OF RESPONSE = 30.

11) ESTIMATE OF MAXIMUM NUMBER OF DISCRETE ITEMS IN AN ITEM_CT RUN = 80.

31) ITEM_CT QUESTION NUMBERS PER RUN = 2 , 6.

32) ITEM_CT RESPONSE TYPES PER RUN = 4 , 1.

Output

If, in instruction card 11, the researcher's estimate of the number of discrete items to be found (e) is not large enough, the output from ITEM_CT will consist of:

ERROR: FOR QUESTION # 6, 'DISCRETE ITEM ESTIMATE =' IS NOT LARGE ENOUGH. MAXIMUM REACHED ON ID # 0734.

If, however, a large enough estimate has been entered, the output for each independent activation of ITEM_CT will take the form of the following example:[1,2]

[1] The number of respondents processed in this example is 25.

[2] This output is of only one question on an interview schedule, that question being, "List all position you have ever held within this school system." In this hypothetical example there were 25 respondents and a total of 50 positions listed.

ITEM_CT SUBROUTINE: RUN # 1

ORIGINS OF NEW ITEMS

ITEM	ID STRING
TEACHER	0714
PRINCIPAL	0715
DIRECTOR	0715
CONSULTANT	0720
COUNSELOR	0724
COUNSILOR	0743

RANKED ITEMS

TOTAL CHOICES = 50

TOTAL CHOOSERS = 25

ITEM	DISTRIBUTION	% OF CHOICES	% OF CHOOSERS
TEACHER	17	34	68
COUNSELOR	14	28	56
CONSULTANT	10	20	40
DIRECTOR	4	8	16
PRINCIPAL	4	8	16
COUNSILOR	1	2	4

Considerations

In studying the output example for ITEM_CT, three things should be brought to attention.

A. The "Origins of New Items" list aids in detecting spelling errors. Each time a unique entry is encountered, it is listed under this heading along with the ID number of the respondent who listed the item. Here, "COUNSILOR" is obviously a spelling error. On the ranked distribution below the listing, it appears only once. Yet, a glance will reveal the ID number on the record upon which this error appeared. It is now a simple task to shuffle manually through the data to the specified record and correct the error. Similar output exists for each of the subroutines presented in this chapter.

B. The items are listed in rank order--ranked by the number of times they are mentioned by all respondents in the data subsection. This is accomplished by the automatic internal calling of the subroutine RANK, reproduced in Appendix One.[1] This subroutine may be called for any two or three dimensional array set; one array of which contains character data (ITEM as an example), the other of which contains numeric data (ITEM_DIS as an example). This subroutine is easily adaptable to more than three dimensions; however, the majority of research purposes should be served by the existing three dimensional limit. RANK is also utilized in the CLASSIFY subroutine appearing later in this chapter as an example of a type two manipulative subroutine.

C. The "% OF CHOOSERS" column indicates the percentage of the respon-

[1] STGPROC statements 127-152.

dents who choose a certain item. In the case of the above example, this list is different than the "% of Choices" list, as there were 25 respondents and 50 choices made (an average of two responses per respondent). If each respondent could make only one choice, these lists would be identical--the only instance in which this would be the case. The only assumption underlying these listings that is important here is that a respondent lists any certain item only once. If more than one entry of an item is contained in any respondent's subresponse list, then the "% of Choosers" column of figures will not be accurate (although the "% of Choices" column will still be correct).

The subroutine ITEM_CT is general, and seems highly amenable to instances of preliminary investigation of samplings from unknown universes. If, on the other hand, the researcher has some ideas concerning the universe and wishes to group data under some classificatory scheme of his choosing, he would be best off utilizing a manipulative subroutine of type two.

Manipulative Subroutine: Type Two

This type of subroutine concerns itself with categorization as supplied by the researcher. Essentially, given a question with an unlisted number of subresponses, this type of subroutine will classify these subresponses in any manner the researcher specifies.[1] This is an especially important type of subroutine, as it allows: (1) multiple classification of the same data with no coding; (2) the same classification of two or

[1] An example of data to be utilized with this subroutine would be a list of occupational hazards, which could be automatically classified in any of a number of manners.

more sets of data. Hence, two major types of manipulation may be performed: (1) on succeeding lata runs, a classification scheme may be built and modified to most closely represent and reflect the data; (2) many schemes, differing upon their bases of classification, may be simultaneously applied to the same data. The illustrative subroutine of this type is CLASSIFY, and is reproduced along with comment cards in Appendix One.[1]

CLASSIFY Option: Activation

There are ten instruction cards to be manipulated by the researcher to effect activations of the subroutine CLASSIFY. The general forms of these statements are:

12) NUMBER OF CLASSIFY RUNS = n.

13) CLASSIFY LENGTH OF RESPONSE = l.

14) ESTIMATE OF NUMBER OF UNCLASSIFIED RESPONSES = e.

15) NUMBER OF THESAURI UTILIZED = nt.

16) LARGEST NUMBER OF CATEGORIES IN A THESAURUS = nc.

17) LARGEST NUMBER OF ITEMS IN A THESAURUS CATEGORY = ni.

33) CLASSIFY QUESTION NUMBERS PER RUN = q_1 , q_2 , q_n.

34) CLASSIFY RESPONSE TYPES PER RUN = t_1 , t_2 , t_n.

35) CLASSIFY THESAURUS USED PER RUN = th_1 , th_2 , th_n.

45) THESAURI DECLARATIONS.

THESAURUS (1).

CATEGORY (1) = c_1, c_2, c_{ni}.

[1] STGPROC statements 245-367.

CATEGORY (2) = c_1, c_2, c_{ni}.

CATEGORY (nc) = c_1, c_2, c_{ni}.

THESAURUS (2).

THESAURUS (nt).

where

n = number of instances of the application of the subroutine to the data in a program run

l = the character length of the largest datum to be processed

e = estimate of the largest number of responses not covered by a thesaurus

nt = number of unique thesauri declared in statement 45

nc = largest number of categories in a thesaurus

ni = largest number of items in a category

q = the question number of the data the subroutine is to be applied to

t = the response type of the data the subroutine is to be applied to

th = the thesaurus to be utilized on this application of the subroutine

c = item in the thesaurus

The Thesauri

In order to classify responses, the subroutine CLASSIFY utilizes researchers' categories as proposed in thesauri. A thesaurus is simply a list of key word-phrases that the subroutine utilizes in attempted matches with the data.[1] This list of words is stored in a three dimensional array, defined in STGPROC as "THES."

Figure VIII: Example of "THES" Array[2]

	1	2
1	IMPROVE	BUILD UP
2	SAVE	PRESERVE
3	DESTROY	TEAR DOWN
4	IGNORE	

Figure VIII shows a four by two by one array, containing key word-phrases for possible matches with the data in seven of its eight cells. (If a cell consists of the null string, no match is attempted.) Note that, as in "QUES_ARRAY," "THES" is composed of subresponses and response types. Reading each row, one learns that it is composed of synonyms. Hence, row totals become the frequency counts for categories, each row comprising a category. "THES" is set up in this manner, as there may be many key word-phrases to be subsumed under the same category. Alter-

[1] This is essentially the same process employed by the BIRS system in the Descriptive Analysis Program; however, the added features of STGPROC make it many times more flexible than BIRS (see Vinsonhaler, 1967).

[2] This thesaurus might be employed to classify answers to the question: "What should local governmental attitudes be toward the central business district?"

natively, there may be only one.[1] In this case, a "THES" array with n by one by one dimensionality may be set up. The third dimension of "THES" is simply the number of unique two dimensional thesauri that have been set up. In the case of Figure VIII, the third dimension of "THES" takes the value of one.

The thesauri are entered in STGPROC by means of instruction card 45. In the example shown in Figure VIII, the instruction to enter the one thesaurus would take the following form:[2]

Example One

```
45) THESAURI DECLARATIONS.

    THESAURUS (1).

        CATEGORY (1) = IMPROVE,     BUILD UP.

        CATEGORY (2) = SAVE,        PRESERVE.

        CATEGORY (3) = DESTROY,     TEAR DOWN.

        CATEGORY (4) = IGNORE.
```

If other thesauri are to be entered for the same program run, their entries follow in numerical order by thesaurus (within thesaurus, by category). An example of the entry of two thesauri follows:[3]

[1] For example, in Figure VIII, the entries "IMPROVE" and "BUILD UP" are the synonyms comprising category one, however category four has only one entry, "IGNORE."

[2] Again, as in the case of declaring delimiters and the case of declaring identification strings, if any of the three symbols, equals (=), comma (,), or period (.) are to be a part of a thesaurus item, the symbol must be enclosed in single quotes.

[3] Note how simple it is to change the elements in the thesaurus. By leaving blank spaces between item declarations, it is an easy matter to change item entries within any single category without having to repunch the entire instruction card. Furthermore, whole categories can be changed by simply changing the number associated with the category, "CATEGORY (n),"

Example Two

45) THESAURI DECLARATIONS.

THESAURUS (1).

CATEGORY (1) = SUPERINTENDENT,	PRINCIPAL.
CATEGORY (2) = TEACHER,	TEACHING AIDE.

THESAURUS (2).

CATEGORY (1) = SECRETARY,	LIBRARIAN.
CATEGORY (2) = CUSTODIAN.	

As an example, suppose one wished to produce two listings of responses, each classified by one of the two thesauri presented directly above:

A. The character length for the first listing being 30, and for the second 20.

B. The estimate for the number of responses not covered by a thesaurus being 10.

C. There will be two thesauri; the largest number of categories in either being two, the largest number of items in any category being two.

D. The first listing is to concern those data found in question seven, response type one. The second thesaurus is to be employed.

E. The second listing is to concern those data found in question eight, response type six. The first thesaurus is to be employed.

In order to activate CLASSIFY for these two runs, the following specific forms of the instructions must be inserted in the instruction card set:[1]

or the category may even be entered in another thesaurus in the above manner.

[1] Instruction 45 (the declaration of the thesauri) is not included

12) NUMBER OF CLASSIFY RUNS = 2.

13) CLASSIFY LENGTH OF RESPONSE = 30.

14) ESTIMATE OF NUMBER OF UNCLASSIFIED RESPONSES = 10.

15) NUMBER OF THESAURI UTILIZED = 2.

16) LARGEST NUMBER OF CATEGORIES IN A THESAURUS = 2.

17) LARGEST NUMBER OF ITEMS IN A CATEGORY = 2.

33) CLASSIFY QUESTION NUMBERS PER RUN = 7, 8.

34) CLASSIFY RESPONSE TYPES PER RUN = 1, 6.

35) CLASSIFY THESAURUS USED PER RUN = 2, 1.

Output

The output for each independent activation of CLASSIFY will take the form of the following example.[1,2]

here, as the appropriate form of this instruction for this example has already been covered (the second example of instruction 45 appearing above).

[1] The thesaurus utilized in the output example is thesaurus number one of thesaurus example number two, reproduced on the preceeding page. The responses to which it is applied come from the hypothetical question: "List all positions you have held in this school system."

[2] The two columns of percentages may need some explanation. "PERCENT MATCHED" has as its base the total number of entries that are covered by the thesaurus (here, 40). Because (as in this example) the thesaurus may not cover all the respondent entries, there is also a "PERCENT OF TOTAL" column, which has as its base the total number of respondent entries (here, 44). Clearly, if all entries are covered by the thesaurus (an ideal case), then these two columns of percentages would be identical.

```
SUBROUTINE CLASSIFY RESULTS:  RUN # 2

RESULTS ORDERED BY THESAURUS

                                                          PERCENT
    ITEM              DISTRIBUTION    PERCENT MATCHED     OF TOTAL

CATEGORY 1

SUPERINTENDENT             4                10               8
PRINCIPAL                  6                15              12

CATEGORY 2

TEACHER                   10                25              20
TEACHING AIDE             20                50              40

RANKED DISTRIBUTION OF RESULTS:

BY CATEGORY

                                                          PERCENT
CATEGORY              DISTRIBUTION    PERCENT MATCHED     OF TOTAL

CATEGORY 2                30                75              60
CATEGORY 1                10                25              20

TOTAL MATCHES = 40

TOTAL CHOICES = 44

BY ITEM

                                                          PERCENT
    ITEM              DISTRIBUTION    PERCENT MATCHED     OF TOTAL

TEACHING AIDE             20                50              40
TEACHER                   10                25              20
PRINCIPAL                  6                15              12
SUPERINTENDENT             4                10               8


RESPONSES NOT COVERED BY THESAURUS:  VICE PRINCIPAL, LIBRARIAN, INSTRUC-

     TIONAL AIDE, INSTRUCTIONAL AIDE,

ID NUMBERS OF NOT COVERED RESPONSES:  SMITH1, FOCKETT, ADAMS3, LECREST,
```

Considerations

CLASSIFY is but one type of subroutine of this type. It employs basic concepts in such a manner that further types of subroutines should be relatively easy matters for one with a modicum of programming ability. There are three modifications the author will point out. The researcher is invited to use his own imagination in designing routines of this sort.

A. Thesauri of more than two dimensions (categories and items) can be easily incorporated in CLASSIFY.

B. Multiple thesauri may be employed upon the same data. Recalling the discussion in Chapter Two under "Key Word Searches," it is possible to code responses (for example) as subject, verb, and direct object. In this case, three thesauri might be created and run upon the data thusly broken up--thesauri covering subjects, verbs, and direct objects respectively. Any one of the three thesauri could be designated as the "base" thesaurus, such that a match on this thesaurus would activate a matching process utilizing another of the thesauri, and so on. In this manner, results would be <u>partialled</u> according to each set of categories. Sample results might take a form such as the following:

```
(Subject)                            THES1 (1) = 20

(Verb)                     THES2 (1) = 15          THES2 (2) = 5

(Object)    THES3 (1) = 10  THES3 (2) = 5   THES3 (1) = 3   THES3 (2) = 2


(Subject)                            THES1 (2) = 10

(Verb)                     THES2 (1) = 3           THES2 (2) = 7

(Object)    THES3 (1) = 3   THES3 (2) = 0   THES3 (1) = 4   THES3 (2) = 3
```

C. The above conceptualized subroutine could be adapted to perform a facet analysis upon the data.[1] In any of the above schemes a factor analysis of the data might well be of importance. This can be accomplished by use of the FACTOR ANALYSIS option explained in Chapter Five.

Manipulative Subroutine: Type Three

The third type of manipulative subroutine is characterized by those manipulative actions (including categorization) performed upon a multiple number of subresponses, listed as answers to one question in the respondent's record, being determined by some single response to another question in the respondent's record. This type of subroutine is especially useful in any type of nominational analysis of data, including the construction of sociometric maps, matrixes, and graphs. It also can be utilized as a further manipulative maneuver upon data processed under subroutines of type one, such as ITEM_CT, as it yields not only a count, but also specifies this count in relation to some conditional variable in the respondent record. The subroutine illustrative of this type is SOCIO, reproduced along with comment cards in Appendix One.[2]

SOCIO Option: Activation

SOCIO, as the STGPROC example of the above type of subroutine, can be set up to manipulate independently as many pairs of questions as the

[1] For an explanation of facet analysis, see Runkel, 1967, 2-26.

[2] STGPROC statements 368-489.

researcher wishes on the same data run. It also will perform upon whatever response type within each question that the researcher specifies. There are nine instruction cards controlling the utilization of SOCIO. The general forms of these instructions are:[1]

18) NUMBER OF SOCIO RUNS = n.

19) SOCIO LENGTH OF RESPONSE = l.

20) SOCIO LARGEST TOTAL NUMBER OF NOMINEES IN A RUN = tn.

21) MAXIMUM NUMBER OF NOMINATIONS PER NOMINATOR IN A SOCIO RUN = nn.

36) NOMINATOR QUESTION NUMBERS PER SOCIO RUN = q_1 , q_2, q_n.

37) NOMINATOR RESPONSE TYPES PER SOCIO RUN = t_1 , t_2, t_n.

38) NOMINEE QUESTION NUMBER PER SOCIO RUN = qn_1 , qn_2, qn_n.

39) NOMINEE RESPONSE TYPES PER SOCIO RUN = tn_1 , tn_2 , tn_n.

40) NUMBER OF NOMINEES PER SOCIO RUN = ns_1 , ns_2 , ns_n.

where

n = number of instances of the application of the subroutine to the data in a program run

l = the character length of the largest datum to be processed

tn = estimate of the largest total number of nominees in any single run

nn = the largest number of nominations possible in any single run, per nominator

[1] Note that in instruction 40, the number of nominees to be acted upon is specified. This allows one to process only a certain number of nominees per nominator; such as the first two nominees in the subresponse list, or the first four. This option is valuable, especially if the nominations appear in a ranked order in the question response (questions can be worded to insure this). If the number in instruction 40 is set at the maximum number of subresponses for that question, then the subroutine will process all nominees listed.

q = the question number of the nominator data the subroutine is to be applied to

t = the response type of the nominator data the subroutine is to be applied to

qn = the question number of the nominee data the subroutine is to be applied to

tn = the response type of the nominee data the subroutine is to be applied to

ns = the number of nominees to be considered per nominator per run

As an example, suppose one wished to produce one listing employing SOCIO:

A. The maximum character length for the data utilized by the listing being 40.

B. The total number of nominees being estimated at 50.

C. The maximum number of nominations per nominator being six.

D. The nominator data being question three, response type two.

E. The nominee data being question one, response type four.

The instruction cards necessary to generate this listing would look as follows:

18) NUMBER OF SOCIO RUNS = 1.

19) SOCIO LENGTH OF RESPONSE = 40.

20) SOCIO LARGEST TOTAL NUMBER OF NOMINEES IN A RUN = 50.

21) MAXIMUM NUMBER OF NOMINATIONS PER NOMINATOR IN A SOCIO RUN = 6.

36) NOMINATOR QUESTION NUMBERS PER SOCIO RUN = 3.

37) NOMINATOR RESPONSE TYPES PER SOCIO RUN = 2.

38) NOMINEE QUESTION NUMBERS PER SOCIO RUN = 1.

39) NOMINEE RESPONSE TYPES PER SOCIO RUN = 4.

40) NUMBER OF NOMINEES PER SOCIO RUN = 6.

Output

There is an instance, in setting up storage arrays for SOCIO, where array size has to be estimated. If the estimate is not large enough, the output for the relevant SOCIO run will identify this error in a statement such as the following:

```
ERROR:  IN SOCIO RUN # 4 THE INSTRUCTION CARD # 20 (SOCIO
NUMBER OF NOMINEES) IS NOT LARGE ENOUGH.  INCREASE THE
NUMBER.
```

If all arrays are of a correct size, the output from SOCIO takes the form of the following example:

A. The data are printed out to aid in detecting spelling errors; one page skipped between respondents.

```
SPELLING CHECK FOR SOCIO:  RUN # 4

ID NUMBER = 1076

NAME = JOHNS1

NOMINATION #1 = PETERS
NOMINATION #1 = DOESS
NOMINATION #3 = SAMBKI
NOMINATION #4 = LUSK
```

B. The final output:

```
RESULTS OF SUBROUTINE SOCIO:  RUN # 4
```

```
RECIPROCAL CHOICES

JOHNS1 CHOOSES PETERS AND VICE VERSA

SAMBKI CHOOSES PETERS AND VICE VERSA

NUMBER OF CHOICES = 4

ONE WAY CHOICES WHEN RECIPROCALITY WAS POSSIBLE

DOESS CHOOSES JOHNS1

SAMBKI CHOOSES JOHNS1

LUSK CHOOSES DOESS

NUMBER OF CHOICES = 3

ONE WAY CHOICES WHEN RECIPROCALITY WAS NOT POSSIBLE

JOHNS1 CHOOSES YERXA

SAMBKI CHOOSES TAMBE

LUSK CHOOSES WREN

NUMBER OF CHOICES = 3

POSSIBLE NUMBER OF CHOICES PER PERSON = 15

NUMBER OF RESPONDENTS = 10

NUMBER OF ACTUAL CHOICES = 10

NUMBER OF POSSIBLE CHOICES = 150
```

Considerations

A. If one wishes a frequency count of the nominations as well, one need only activate the ITEM_CT option for that purpose.

B. SOCIO need not be used solely with nominational data. The conditional variable may be of any type the researcher wishes it to be; however, as the subroutine now stands, its most effective use is in a nominational sense, whether the nomination be in terms of names, positions, tasks performed, and so on.

C. Subroutines such as SOCIO can be made to lend themselves readily to factor analysis--the data generated by these subroutines may be introduced into any relevant subroutine the researcher wishes with a minimal amount of programming.

Summary

Chapter Four concerned itself with the manipulative subroutines of STGPROC, identifying three major types of manipulation: (1) those in which the response itself determines the data manipulation; (2) those in which the researcher supplies the information which determines the data manipulations; and (3) those in which the response to another specified question determines the data manipulation. Each of these three general types of manipulation was discussed and specific programmed subroutines were presented (ITEM_CT, CLASSIFY, and SOCIO). Chapter Five will now proceed with a discussion of statistical subroutines and their use with STGPROC produced data.

CHAPTER FIVE

THE STATISTICAL SUBROUTINES

In many cases it would be advantageous for the researcher to be able to perform various statistical tests upon his data. Although STGPROC is primarily a data manipulation system, the need for statistical procedures has been recognized. Rather than designing these statistical procedures, however, it seemed more efficient to adapt the STGPROC system so that advantage could be taken of statistical programs already in existence.

The chapter deals with two subroutines, each of which performs automatic coding of data and repunches it in fixed-field format such that these new data cards may be utilized with existing statistical programs. In one subroutine (SPSS) the coding is done in terms of the manipulated data, in the other (FACT_AN) the coding criteria are thesaurus entries. Codesheets are automatically generated along with the punched data cards.

Although there have been many statistical procedures written, the two subroutines presented in this chapter have been designed with The Statistical Package for the Social Sciences (SPSS), created by Norman Nie and Dale Bent at Stanford University (Nie and Bent; 1968), and the Bio Medical Series factor analysis program in mind (Statistical Laboratory and Computing Center of the University of Oregon Library Program UOBMDX72). Both these systems are currently resident at the computation center of the University of Oregon and can be utilized with a minimum of effort and programming knowledge. It should be noted, however, that although the STG-

PROC statistical subroutines were designed for use with these specific systems, they are general enough so that their output may be utilized as data with many other statistical programs. The researcher is urged to check the availability of such programs at his resident computation center.

Before proceeding further with a presentation of the STGPROC statistical subroutines, I shall describe briefly the configuration of the SPSS system and the statistical applications covered by it. This exercise is designed primarily to show the researcher who is inexperienced with computer programs the flexibility and versatility of just one example of a statistical package available for use with STGPROC.

The Statistical Package for the Social Sciences

The SPSS system contains a number of subroutines, any or all of which can be called to process a given set of data. To activate the program the researcher, as he does with STGPROC, need only write a small number of procedure and job control cards calling for the desired subroutines and options within each.[1] The specific procedure cards are explained in the cited manual dealing with the SPSS system (Nie and Bent, 1968). These procedure cards specify the format of the data to be input and other information essential to the program.

[1] The form of these job control cards varies in terms of what procedures one wishes to activate as well as in terms of how the specific computation center has set up operations. Since the control cards for the SPSS system vary and are written in code best comprehended by those familiar with computing languages, it is suggested that the researcher consult his resident computation center for the forms necessary for the creation of his desired statistics.

The following features of the SPSS system are important in relation to the STGPROC system: (1) Up to 500 variables may be declared for a single processing run, although this number is reduced depending upon the subprogram being utilized; (2) there are a number of options available for processing missing data; (3) data can be recoded temporarily or permanently; (4) many variable transformations may be accomplished to normalize distributions, construct scales or indices, and so on; and (5) error codes and explanations are provided for debugging purposes.

The following subroutines are available in the SPSS system:[1]

A. Subroutine CONDESCRIPTIVE: Outputs any or all of the following: mean, standard error, standard deviation, variance, kurtosis, skewness, range, minimum, maximum.

B. Subroutine CODEBOOK: Outputs tables containing any or all of the following: simple raw frequencies, relative frequencies with missing values included, relative frequencies with missing values excluded, cumulative adjusted frequencies for grouped data, histograms, mean, standard error, median, mode, standard deviation, variance, kurtosis, skewness, range, miminum, maximum.

C. Subroutine CROSSTABS: Outputs any or all of the following: bi-variate joint frequency distributions with N-levels of control variables, tables and subtables percentaged by column, row, total table (or any combination of these--or all percentages may be suppressed), Chi square, Fisher's exact probability test, Phi, Cramer's V, Lambda, Uncertainty coefficient, Contingency coefficient, Kendall's Tau B, Kendall's Tau

[1] Subroutines I-K are not yet operational but are imminently expected to be.

C, Gamma, Sommer's D.

D. Subroutine PEARSON CORR: Outputs zero order product moment correlation coefficients, significance tests, cross-product deviations, and covariances.

E. Subroutine NONPAR CORR: Outputs Kendall and/or Spearman rank order correlation coefficients and levels of significance.

F. Subroutine GUTTMAN SCALE

G. Subroutine REGRESSION: Outputs multiple regression results as well as stepwise regression.

H. Subroutine MULTIVARIABLE PLOTTING

I. Subroutine SCATTER DIAGRAMMING

J. Subroutine PARTIAL R: Outputs partial correlations.

K. Subroutine FACTOR ANALYSIS

The Statistical Subroutine SPSS

This STGPROC subroutine handles as many variables as the researcher specifies, coding each variable value in a fixed field of length three and punching within succeeding fields of each record all values of that variable associated with each respondent. In other words, SPSS punches all respondent subresponses per record and yields as many records as there are total respondents. Because of this, variable values are linked with individual respondents and operations such as Guttman Scaling and multivariable plotting may be performed utilizing the punched data output from this subroutine.

Option SPSS: Activation

SPSS has been presented in conjunction with the manipulative subroutine ITEM_CT (see Appendix One).[1] In this case, each time a subresponse of a considered question is classified by ITEM_CT, the subroutine SPSS is called. This subroutine automatically codes the subresponses and retains its coded value for inclusion in the punched data output.

There are four instruction cards to be manipulated by the researcher to effect activation of the subroutine SPSS. The general forms of these instructions are:

22) SPSS RESPONDENT ESTIMATE = e.

23) SPSS NUMBER OF RUNS = n.

24) MAXIMUM NUMBER OF SUBRESPONSES IN A SPSS RUN = m.

42) ITEM_CT RUNS ACTED UPON BY SPSS = r_1, r_2, r_n.

where

e = estimate of total number of respondents in the data

n = number of questions to be acted upon by the subroutine

m = maximum number of responses in a question to be acted upon

r = the number of the ITEM_CT run acted upon[2]

As an example, suppose one wished to activate SPSS for the second ITEM_CT run covered in the ITEM_CT example given in Chapter Four:[3]

[1] STGPROC statements 550-638.

[2] These numbers have to be in ascending numeric order.

[3] Chapter Three does not show the hypothetical output from the second ITEM_CT run, only from the first. The hypothetical question covered by the second run however, is: "List all those from your school with whom you also interact socially."

A. There were 25 respondents in the example of output. The estimate here, to be generous, will be 35.

B. There is one ITEM_CT run to be acted upon.

C. The maximum number of subresponses to be acted upon being three.

D. The ITEM_CT run to be acted upon being the second.

The specific forms of the instructions necessary to generate the punched data called for above are:[1]

22) SPSS RESPONDENT ESTIMATE = 35.

23) SPSS NUMBER OF RUNS = 1.

24) MAXIMUM NUMBER OF SUBRESPONSES IN AN SPSS RUN = 3.

42) ITEM_CT RUNS ACTED UPON BY SPSS = 2.

Output

If the number of variables to be coded is erroneously listed in instruction 23 of STGPROC, the following type of error message will result.

ERROR: INCREASE SIZE OF INSTRUCTION CARD 23 (SPSS NUMBER OF RUNS).

If all is in order, the subroutine will print a code sheet such as the following for each variable it is considering.

[1] Note that statement 42 is a list of the ITEM_CT run numbers, not the number of ITEM_CT runs.

CODING FOR SUBROUTINE SPSS, RUN # 1

ON QUESTION # 6

ITEM	CODE
SMITH	1
JONES1	2
HEZOL	3
ADAMS	4
LASSITER	5
JONES2	6
HARRY	7
GOODWIN	8
JOHNSON	9
PARTMAN	10

ITEM_CT RUN # 2 IN QUESTION # 6 IS VARIABLE # 1 IN THE PUNCHED OUTPUT WITH 3 SUBRESPONSES OF THREE COLUMNS APIECE MAKING A TOTAL FIELD LENGTH OF 9.

The subroutine then reproduces the punched card data by means of printed output for visual inspection.[1]

SPSS PRINTOUT OF DATA CARDS:

```
 1 1  1  2  3
 2 1  4  5  6
 3 1  2  1  7
```

[1] Note there were actually only 12 respondents in this example.

```
 4 1  8  9 10
 5 1  1  7  2
 6 1  2  4 10
 7 1  3000000
 8 1  1  5  6
 9 1  4  2  6
10 1 10  3000
11 1  5  8  9
12 1  7000000
```

The SPSS Data Cards

The punched output takes the following form:

A. The first card contains "SPSS PUNCHED DATA FOLLOWS."

B. Columns one through three of the card are the card record number, from "1" up in increments of one.

C. Columns four and five are the card continuation number.

D. The remaining columns are the coded variable values. Each variable value is in a fixed field of three columns. There exist as many fields of three columns as there are declared subresponses (or variable values) for a respondent per question run (in this case, three). Again, the placement of the coded values is easily determined from the information upon the accompanying code sheets.

E. There will be as many records as there are respondents.

F. In the case of varying numbers of subresponses per respondent, the "blank" fields will be filled with "000" and the missing data option of the SPSS system will, upon command, ignore or include these field values, as the researcher wishes.

G. If there is not room for all the data on one card, it will be continued on further cards with the appropriate card continuation numbers.

Considerations

Although SPSS is set up to work in conjunction with ITEM_CT, it can be easily adapted to operate with any manipulative subroutine the researcher wishes. SPSS can handle as many variables as the researcher wishes; however, it creates only one data deck per subsample per STGPROC run. From the accompanying codesheets, it is a simple matter to identify the limits of the fixed fields of each variable for their inclusion in the instructions to any statistical program one is running. Furthermore, labels for the statistical tables may be obtained from the code sheets. A further advantage, in utilizing the SPSS system, is that codes may be merged and categories created during the statistical analysis.

The Statistical Subroutine FACT AN

The STGPROC subroutine handles as many responses as the researcher specifies, coding each response upon a separate data card. The data take the form of a string of "0's" and "1's" and is coded in such a way that the produced data cards may be added to the STGPROC data for future processing of other types, as well as being utilized as data for the Bio Medical Series factor analysis program of the library of the computation center of the University of Oregon (UOBMDX72).[1]

[1] This program is fairly standard and easily obtained. Hopefully, when the SPSS system factor analysis subroutine is developed, it also may be utilized in conjunction with this data.

Option FACTOR ANALYSIS: Activation

FACTOR ANALYSIS has been presented in conjunction with the manipulative subroutine CLASSIFY (see Appendix One).[1] In terms of any CLASSIFY run, each item in the utilized thesaurus is assigned a column on the respondent's data card. If the respondent listed that item, the column is assigned a "1." If he did not list that item, the column is assigned an "0." Therefore, the punched data takes the form of one string of "0's" and "1's" per CLASSIFY run per respondent.[2] Each string begins upon a new data card, labelled as to the respondent's identification string and the card continuation number.

There are three instruction cards to be manipulated by the researcher to effect activation of the subroutine FACT_AN. The general forms of these instructions are:

25) FACTOR ANALYSIS CONTINUATION NUMBER = c.

26) FACTOR ANALYSIS NUMBER OF RUNS = n.

41) CLASSIFY RUNS ACTED UPON BY FACTOR ANALYSIS = r_1, r_2, r_n.

where

c = the card continuation number the researcher wishes to factor analysis punched cards to begin with (important only if these

[1] STGPROC statements 490-549.

[2] The consideration of thesaurus items proceeds from the first item in category one of the thesaurus to the nth item in category n. For example, in the output shown for the CLASSIFY run in Chapter Four, the first column of the string of "0's" or "1's" would be defined by considering "SUPERINTENDENT," the second, "PRINCIPAL," the third, "TEACHER," and the last "TEACHER AIDE." This would generate, for each respondent, a string of "0's" and "1's" of a length of four characters. If a respondent listed "TEACHER" for example, his string would look as follows: 0010.

data are to be merged in the STGPROC data deck).

n = number of questions to be acted upon by the subroutine

r = the number of the CLASSIFY run acted upon[1]

As an example, suppose one wished to activate FACTOR ANALYSIS for the first and fourth CLASSIFY activations in some certain STGPROC run:

A. The card continuation number being 40.

B. The number of CLASSIFY runs to be acted upon being two.

The specific forms of the instructions necessary to generate the punched data called for above are:

25) FACTOR ANALYSIS CONTINUATION NUMBER = 40.

26) FACTOR ANALYSIS NUMBER OF RUNS = 2.

41) CLASSIFY RUNS ACTED UPON BY FACTOR ANALYSIS = 1, 4.

Output

The subroutine reproduces the punched card data by means of printed output for visual inspection.

PRINTOUT FOR FACTOR ANALYSIS DATA CARDS:

```
40SCHELLING    $15#0010$
41SCHELLING    $16#0101000$
40JOHNS        $15#0101$
41JOHNS        $16#0001011$
40THOMAS       $15#1100$
41THOMAS       $16#0100110$
```

[1] These numbers have to be in <u>ascending</u> numeric order.

```
40SAMKIT       $15#0001$

41SAMKIT       $16#1101110$
```

The FACTOR ANALYSIS Data Cards

The punched output takes the following form:

A. The first card contains "FACTOR ANALYSIS PUNCHED DATA FOLLOWS."

B. Columns one through two are the card continuation number ("40" and "41" above).

C. The columns up to the first level delimiter (here the "$") contain the identification strings. The number of columns is equal to the maximum size of the identification strings, as defined in instruction 27 (see Chapter Two).

D. The next two columns are the question number ("15" and "16" above). These question numbers begin at the upper limit of the existing STGPROC data. For example, in this case, the highest STGPROC data question number was "14," hence the factor analysis data begins with question number "15."

E. The second level delimiter (here "#") follows the question number.

F. The following string is as many columns long as there are item entries in the thesaurus being employed. This is the data to be utilized in the Bio Medical Series factor analysis program.

G. The data are terminated by the first level delimiter.

H. If there is not room for all the data on one card, it will be continued on further cards with the appropriate card continuation numbers.

Considerations

Although FACTOR ANALYSIS is set up to work in conjunction with CLASSIFY, it can be easily adapted to operate with any manipulative subroutine (except those of type one--in which the responses themselves define the categories) the researcher wishes. FACTOR ANALYSIS can handle as many responses as the researcher wishes; however, it creates only one data deck per subsample per STGPROC run. The data are not only in the form amenable to factor analysis programs, but they also can be merged with the original STGPROC data for any further manipulations which the researcher may decide upon. Also, in the punching of respondent factor analysis scores by the Bio Medical Series program, it is an easy task to specify the first _n_ columns of the data card as an identification number. If this is done, the Bio Medical program can be instructed to punch the scores, preceeded by all the necessary symbols to merge these scores with the original STGPROC data for further manipulation, with no hand punching of identification symbols necessary on the part of the researcher.

Summary

Chapter Five concerned itself with the statistical subroutines of STGPROC, presenting a brief overview of the Statistical Package for the Social Sciences (SPSS), then describing two general subroutines that adapt STGPROC data for use with: (1) the SPSS system, (2) the Bio Medical Series factor analysis program. Each subroutine produces codesheets and a deck of punched output as adapted data for the relevant statistical programs. The SPSS system and the Bio Medical Series program were recommended because of their availability, versatility, and ease of use. However the resulting punched output could as well be used as data in any other similar statistical programs. Chapter Six will now proceed with a description of the STGPROC system in action--the presentation of a "case study" of data manipulation and analysis by means of this system.

CHAPTER SIX

STGPROC IN ACTION

Perhaps the best method of describing the STGPROC system in action is not the presentation of one single example of research performed by means of this system. As was stressed earlier in this monograph, the STGPROC system is designed with interaction in mind--interaction between the researcher and the computer. Viewed in this way, STGPROC should not be thought of as a static system, but as one that is continually being modified by the dictates of the research to which it is applied. The concern then should be with the evolution of the system and its continuing adaptation to research designs. This is the manner in which all extensive computer systems come into being (cf. Stone, Dunphy, Smith, Ogilvie, 1966; Nie and Bent, 1968). Therefore I feel the most informative description of STGPROC in action is, in fact, a description of its evolution and adaptation to past and present research needs. Evolution will be discussed in terms of three general phases of development of the STG-PROC system. Because adaptation is an ongoing concern, at the close of the chapter new directions for possible extensions of the system will be touched upon.

Conceptual Evolution

During the academic year 1967-1968 the Center for the Advanced Study of Educational Administration (CASEA) collected extensive amounts of data from elementary schools across the nation.[1] These data, exploratory in nature and having to do with many organizational variables, focused upon (among other things) the fate of educational innovations and educational decision-making within and without the schools themselves. Research upon these data was termed the CASEA Attributes Projects.[2]

In the spring of 1968 the author was asked to participate in research sessions aimed at arriving at decisions in terms of coding these data. It was immediately apparent, upon inspection of the data, that some drastic coding decisions were going to be necessary if much information was not to be lost in the coding process.

There were many questions in the research instrument which dealt with open-ended listings of such items as the following: persons, their positions in the system, their duties, the nature of their relationships with other persons and other positions, job descriptions, perceptions of the decision-making structure of the organization, and so on. Because of the exploratory nature of the study, these listings were of items drawn from universes of items the bounds of which were unknown to the researcher. Further, there were no limits placed upon the number of items that a respondent could place in any list. Because of the above factors and the vast amount of data to be processed, normal coding procedures seemed at worst impossible and at best vastly inefficient in terms of time and loss of information in the coding process.

[1] These data were drawn from elementary schools in Wisconsin, Pennsylvania, and New Jersey, as well as from whole school districts in the state of Washington.

[2] For a comprehensive discussion of the CASEA Attributes Projects in all its aspects, see Pellegrin, 1968.

For example, one question asked the respondent to list the main tasks performed by him in pursuance of his job. In order to code this question numerically, one of two paths had to be taken:

A. Some categorization scheme, made up on the spot or borrowed from some other study, would have to be applied to the data by the coders, with the hope that most of the items could be "fitted" into the existing categories with a minimal amount of distortion. The problems arising from this (unfortunately common) solution to the problem are many. If the categories do not fit the data well, coded data are produced that are highly distorted and/or suffer from many entries in the catch-all category of "other." This, in turn, leads to information loss and possible distortion of research results, or to a reclassification of the data, involving time and effort. This reclassification may or may not solve the above problems. On top of this, studies have revealed 15 to 20 percent error on the part of experienced coders in manually processing this type of classificatory data (Sussman and Haug, 1967). Further, as this is an exploratory study, one may wish to reclassify the data in terms of some completely different criteria. When this is the case, the data are necessarily handed back to the coders for recoding and all the problems mentioned above are concerns once again.

B. The second path that can be taken is to assign a code number to each unique item. This requires that the coders compile a dictionary of items as they go, assigning a unique code number to each item, and making sure they do not duplicate items or omit any. Clearly this solution to the problem becomes infeasible when dealing with an unknown universe of items and when any appreciable amount of data is to be processed. The job is too demanding and time consuming and there is too much chance for error in coding. Yet, if the mechanical difficulties associated with this solution to the problem could be solved, there would be no information loss due to coding (this coding being a one-to-one translation [see Chapter Two]) and the data could be reclassified upon as many differing bases of classification as the researcher wished without the necessity of recoding.

Clearly, if the mechanical problems could be solved, the second solution to the problem as described above would be far superior to the first. The idea came to mind that these sorts of routinized mechanical problems can many times be solved by use of the computer. Yet, in order to solve this specific problem, the computer would have to "read" the actual string of characters that made up the respondent answer lists and distinguish among differing strings. This was a capability the computer possessed, yet little (if anything) had been done in the field to apply this capability in social science research. The decision was made to explore possibilities along this line.

A further problem presented itself in dealing with these data. The respondent lists were of unequal lengths. One respondent might list four tasks, another might list 14. How was this difference in amount of information to be handled? Methods in existing programs do handle this problem, though not adequately in all cases. Since standard coding procedure is to reserve columns on a IBM card for specific bits of information, one would need to go through all responses, counting the number of items in each listing, to determine the largest listing. Then one would set aside enough columns to handle this largest listing and, on smaller listings; leave the excess columns blank or assign some symbol to them that signified "no response." This process is understandably time-consuming and subject to coding error. After the data were coded, one would have to then instruct the processing program in the manner in which "no responses" were to be treated (to be included in frequency counts, to be ignored, and so on). The SPSS program (Nie and Bent, 1968) has this missing data option and can handle questions of this sort in a limited manner; however, many other systems are not as flexible--processing of the data in this form would depend on the capacities of the processing systems available. Further, if one wished a combination of certain data items in a single frequency count, the process would be difficult if not impossible to set up for most existing processing programs. And, as always, the chances of coding error remain high. It was evident that a method for more adequately handling these uneven response lists would be a desirable feature of any proposed processing system.

STGPROC: Phase One

The considerations discussed in the above section led to the initial development of STGPROC. An initial system of delimiters was worked out and the data were transcribed to IBM cards in terms of it. The main STGPROC program for processing these strings of character data was developed and subroutines were built and attached to this main program as demand dictated.

The first version of the STGPROC system was crude as compared to the version presented in this monograph. Data storage problems were not adequately solved--only small amounts of data could be processed at a time.

The program took up inordinately high amountsof computer time (from five to ten minutes per program run, not counting an average four minutes of compiling time). Instructions to the program were in the form of program statements, and had to be inserted within the working program itself. Subroutines were uncoordinated and were manually shuffled to be included or excluded on specific runs to save computer time. Yet with all its defects, phase one of STGPROC proved that the larger concepts lying behind its creation were sound. While internal problems were gradually being brought to light and solved, the program successfully processed a great deal of character data.

The data processed during phase one of STGPROC helped to determine the shape of future development to the program. Generally speaking, the data processing was of three forms: (1) frequency counts of items in data lists, (2) sociometric mappings, and (3) the development of classificatory schemes.

Frequency Counts

In studying the decision-making structure of schools, many questions were asked the respondents in the form of listings: of persons or groups of persons (if they existed) who helped the respondent evaluate pupils, choose materials, choose subject matter content, choose teaching methods, and schedule activities; to whom the respondent might go for support for his or her ideas; who the respondent felt proposed reasonable and useful solutions to school problems; who the respondent felt made decisions in certain policy areas; who the respondent depended upon to perform his or her job effectively; and so on. These lists of persons are nearly always of an open-ended form and were drawn from an unknown universe of names. Clearly, the coding involved in adapting these data to standard processing form was too extensive to be economically feasible and too complicated to guarantee anywhere near an error free result. Hence an early STGPROC subroutine was developed to buibd frequency counts of this type of data (the basis for the present subroutine ITEM_CT).[1] Some of these frequency counts were utilized in the paper "The Decision-Making Structure of Schools" (Pellegrin, Dudley, Smith, 1969), which was an initial attempt at describing the decision-making processes involved in elementary schools characterized by differing organizational structures. Further aspects of these data are utilized in the CASEA monograph dealing with Multiunit Schools now nearing completion. These data will also be utilized in an extensive CASEA monograph examining decision-making inethe elementary school.

Sociometric Mappings

In coming to grips with the relationships between and among actors in school systems, it seemed approproate to generate sociometric mappings of these systems. The methodological problems mentioned above under "frequency counts" were applicable in the case of sociometric listings also. An early sociometric subroutine (the basis for SOCIO[2]) was developed to handle this type of data manipulation.

[1] See Chapter Four for an explanation of this subroutine and Appendix One, statements 153-244 for a presentation of it.

[2] See Chapter Four for an explanation of this subroutine and Appendix One, statements 368-489 for a presentation of it.

From the program output, sociometric mappings of schools and subunits within schools were created. Sociometric matrixes were also drawn up to compare sociometric patterns based upon differing variables (authority, influence, esteem). The results of these exercises increased early comprehension of role relationship patterns found in the data. In point of fact, these diagrams revealed some patterns that were totally unanticipated. These unanticipated patterns have had a great deal to do with the path of subsequent analysis of the data and have helped lead to the type of analysis found in the forthcoming CASEA monographs dealing with Multiunit Schools and schools characterized by individually prescribed instruction.

Development of Classificatory Schemes

The third form of data processing deemed necessary was in terms of the development of a classificatory scheme or schemes for the listing of tasks associated with respondent jobs. The problems involved in the development of such schemes have already been mentioned. A STGPROC subroutine (the basis for CLASSIFY[1]) was developed to surmount these problems. Basically, the data were matched against a thesaurus, which could be variably constructed. Therefore it was only a matter of manipulating thesaurus entries to arrive at any type of classificatory scheme one wished. Reliance upon previously compiled categorization systems was unnecessary--the scheme could be constructed by means of ongoing interaction between the computer, the thesaurus, and the researcher. Further,

[1] See Chapter Four for an explanation of this subroutine and Appendix One, statements 245-367 for a presentation of it.

the data manipulation and frequency counts were performed by the computer, obviating the high perdentage of human error associated with this type of operation.

One such categorization scheme became the basis for the report "Task Differentiation in Elementary Schools: An Exploratory Analysis" (Stehr, Lewis, Pellegrin, 1969). This subroutine, and refinements of it, have been utilized in other ways upon CASEA data. Examples of further utilizations will be presented later in this chapter.

STGPROC: Phase Two

As a result of the above data manipulations, it soon became evident that system changes were necessary if STGPROC was to be applied to further data. The major problems were those of limited data storage space and the excessive amount of computer time necessary for processing runs. A re-evaluation of the entire system was undertaken, with a resultant major change in the data transcription process. This change in the structure of question transcription involved more standardized use of delimiters across questions. The results were fewer transcription errors on the part of the "coders," the feasibility of an extensive machine check on the form of the data (ERROR_CHECK), increased speed in data manipulation, and a more efficient use of storage areas in the machine. This new form of data transcription is currently the one employed with the STGPROC system.

Phase two of STGPROC was initiated because of a need for certain computer techniques to be employed in a study related to CASEA research and which utilized some of the same data. This study investigatedthe relationship between the division of labor in the school organization (as measured by the structuring of tasks performed by actors) and (1) variable allocations of power, (2) job satisfaction, and (3) the perceptions of rewards (Dudley, 1969).

In order to accomodate this study, two major conceptual modifications of STGPROC were made. First, the program was redesigned to handle the transcribed data in its new form. This helped in solving the problem of storage and excessive computer time, as experienced in phase one. Second, the form of analysis led to the necessity of utilizing existing statistical programs in relation to STGPROC. This consideration of program linking led to the development of ubroutines that produced coded output on punched cards for use with other programs.

The Division of Labor Study

The basis upon which this study rested was a successful classification of tasks listed by respondents. The tasks were to be subjected to factor analysis (as existing classificatory schemes were deemed inadequate) and the resulting respondent factor scores were to be correlated with: (1) the rankings respondents received upon authority, influence, and esteem dimensions of the power variable, (2) respondent scores on job satisfaction and reward perception.

There were therefore a number of manipulative steps to be performed:

A. A frequency count of the listed tasks was needed to determine which tasks, listed differently, were in fact similar (e.g. "give instruction" and "instruct pupils").

B. A thesaurus was needed, made up of all the tasks listed, with similar tasks given as differing items in the same categories.

C. Punched output, based upon the thesaurus categories, was needed to feed into the Bio Medical Series factor analysis program.

D. From this program, punched data were needed that included respondent identification strings and each respondent's factor scores.

E. The rankings each respondent received on questions dealing with authority, influence, and esteem needed to be punched on the respondent cards.

F. The scores each respondent received in terms of job satisfaction and reward perception needed to be punched on the respondent cards.

G..The computation of Kendall's Tau scores among these final data was necessary.

The STGPROC system was at the heart of this data analysis. The data to be utilized were first run through the computer and checked for transcription errors. When these had been corrected, a frequency count of the tasks was run. At the same time, the rankings of respondents in terms of authority, influence, and esteem were computed. All these manipulations were done by means of a subroutine similar to the one now known as ITEM_CT.[1]

A word concerning the task listings is in order here. The basic structure of the transcription of the tasks was approximately the same as that of an English sentence (verb, indirect object, direct object).[2] On the basis of the frequency counts, 43 categories were tentatively estab-

[1] See Chapter Four and Appendix One for explanation and presentation of ITEM_6T.

[2] Subjects were not utilized in this instance, as the referent of each action was always constant--the respondent.

lished. Unfortunately, it was learned upon examination of the first frequency distribution printout that respondents varied in the specificity of their task listings. Some would list only a verb ("plan"), some would indicate what they were planning, and still others would list those involved in the planning or for whom they were planning. Because the level of specificity of response varied, it was necessary to analyze responses at the level of the lowest common denominator of the data, the verb.[1] Hence another frequency count was necessary--one made in terms of verbs only. This frequency count, created by an alternate run of STGPROC with ITEM_CT activated only in relation to the response type "verb," yielded 14 very general variables identified as part of the task structures of the individuals in question.

At this point, another STGPROC subroutine (specific to this study) was utilized to automatically punch the thesaurus cards necessary for the next step in the analysis. Nine hundred thirty-one separate tasks had b been classified, on the basis of the verbs employed within them, into 14 separate categories.

By employing an earlier version of the subroutine CLASSIFY,[2] rank ordered frequency counts of the tasks grouped by category were obtained for each school and for experimental and control groupings.[3] Rank order

[1] This problem can be avoided by careful wording of the questionnaire, a problem taken up in Chapter Two of this monograph.

[2] See Chapter Four and Appendix One for presentation and explanation of CLASSIFY.

[3] It should be pointed out here that, with over 900 entries in the thesaurus, there was not enough memory space in the computer to obtain these results in one run. Hence, it was necessary to break the thesaurus

correlations of the task categories between schools were then easily obtained.

At this point, it was necessary to generate the punched card data for a factor analysis of the tasks. The subroutine FACT_AN[1] was developed for this purpose. The result was a set of data cards, each with the respondent identification number and a string of 14 "O's" and "l's" to be fed into the Bio Medical Series factor analysis program.

From this program, data cards were obtained, each with a respondent identification number and the respondent scores on the six factors found in the data. The respondent scores upon the power variables, job satisfaction, and reward perception were then added to these cards. This set of data was utilized in conjunction with IBM's Scientific Subroutine Package to produce the desired Kendall's Tau scores for the data.

The above example points out some of the reasons that the STGPROC system is of worth in the process of data analysis. In the type of exploratory work that has been described, it is imperative that the data remain in such a form that they may be manipulated in a number of ways without information loss or distortion. This is the case with STGPROC. If the data had been coded in a "normal" fashion, the above research would have been impossible without a complete manual coding of the data,

up into three separate thesauri of 400, 400, and 131 entries (the process of partitioning), and run each thesaurus segment separately. This in no way affected the results except in that the analysis took more computer time than it would have otherwise. The program has since been modified to accomodate much greater amounts of information at once, but if an overflow does occur, this "partitioning" of the thesaurus is an excellent solution to the problem.

[1] See Chapter Five for an explanation of this subroutine and Appendix One, statements 487-546 for a presentation of it.

as a prior classificatory scheme for tasks would have already been imposed (see Stehr, Lewis, Pellegrin, 1969). As it was, even within the above research, the basis for classification was easily shifted from the whole task to merely the verb employed, with no recoding necessary. There has been talk of further research employing the general categories of Durkheim's division of labor (Durkheim, 1933) upon the data. This too, may be accomplished with no recoding of the data. Furthermore, any thesaurus that is set up may be utilized with other data (as will be the case in the near future Attributes Projects research) without the variable of manual coding error creeping in, as the machine will be assigning items to categories.

Another presently available but as yet unused research alternative with STGPROC is to run a factor analysis on nominational data. This type of analysis would yield insights into the composition of whatever sociometric groupings might exist with the data.

The division of labor example also points out the ease with which STGPROC can be utilized in conjunction with existing programs. (It can now also be utilized with the SPSS program.)[1] Rather than attempting to duplicate functions, STGPROC has been designed with complementation in mind. This, I feel, is a very strong point in the program's favor.

STGPROC: Phase Three

As was the case with phase one of STGPROC, phase two pointed out problems with the system as it stood and acted as a guide to further develop-

[1] See the explanation of the SPSS subroutine in Chapter Five and its presentation in Appendix One, statements 547-635.

ment. The research process of the Attributes Projects was becoming more sophisticatdd as the analysis progressed, and to be of value, STGPROC had to become more flexible.

Attention was now being centered on small subsets of the data to search for possible patterns obscured by the analysis of the data as a whole. Hence, there was need of a process of selecting small data subsets for analysis on data runs through the entire data set. Subroutines were added to STGPROC (SUBSECTION, SELECT-IN, SELECT-OUT)[1] which yielded processing options ranging from any one respondent up to the entire data set.

As the number of options in STGPROC grew, the process of specifying certain options for a data run became more complex. If researchers unfamiliar with STGPROC or the programming language in which it was written were to successfully utilize the system, some new method of specifying options had to be devised. It was no longer feasible to alter values on specified statements in the STGPROC program itself. Therefore, a major alteration of the program was effected and the concept of instruction cards was introduced. These cards, processed as data by the system, allow all STGPROC options to be controlled with no manipulation of the program itself.

The addition of these new options understandably increased the size of the program. This reintroduced an old problem--that of the amount of time it took for the computer to process a data run. With the program in its present form, this time factor can be reduced considerably. The pro-

[1] See Chapter Three for explanations of these subroutines and Appendix One, statements 112-126, 951-955, and 956-959 respectively for their presentations.

gram can be translated to machine language by means of a simple program extant in most computation centers. This machine language version of the program can then be stored on tape at the computation center.. Thus, the program will not need to be compiled (translated to machine language by the computer) before the data are processed, saving nearly three minutes of computer time, on the average, per STGPROC run.

When the data have been checked by means of a STGPROC error run, and transcription errors have been corrected, the data also may be put on tape and stored at the computation center. Thus the researcher need submit only the necessary job control cards and instruction cards to produce any specific STGPROC run. Not only is this method more convenient, but it is cheaper, as the computer can process tape much faster than it can IBM cards. Furthermore, with the program and data in this form, it would be an easy step to set the system up for operation with a remote console --so that all the researcher need do is to punch the desired instruction card changes on the remote console to create a STGPROC run.

STGPROC: The Future

STGPROC was purposefully designed as an ongoing, interactive computer system. Three major phases in its development have been described. The system has grown in flexibility, in its number of processing options, and in its compatability with existing data processing systems. STGPROC will have to continue in its development if it is to remain viable in the research world. However, I feel that the point has been reached where further growth will take the direction of subroutines tailored for specific research needs, rather than the direction of increased efficiency of the system itself. In other words, I feel that the operational basis of the STGPROC system is now viable. Further growth will take the form of extensions, rather than alterations, of the existing system.

BIBLIOGRAPHY

Allen, John and Morris Salkoff
1964 "Machine translation: the state of the art, 1964." American Behavioral Scientist 7(June):9-11.

Bates, F. and M. Douglas
1967 Programming Language One. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Borko, Harold and Lauren Doyle
1964 "The changing horizon of information storage and retrieval." American Behavioral Scientist 7(June):3-8.

Carlson, Arthur
1967 "Concept frequency in political text: an application of a total indexing method of automated content analysis." Behavioral Scientist 12 (January):68-72.

Coleman, James
1964 "Mathematical models and computer simulation." Handbook of Modern Sociology, Robert Faris (ed.). Chicago: Rand McNally, 1027-1062.

Dudley, Charles
1969 Task Structure, Allocation of Power, and Satisfaction of Organizational Members in Six Schools. Eugene, Oregon: CASEA, University of Oregon Press.

Durkheim, Emile
1933 The Division of Labor in Society. New York: Macmillan Publishing Co.

Forte, Allen
1967 SNOBOL3 Primer. Cambridge, Mass.: M.I.T. Press.

Gardner, R. C., C. Kuehne, and A. G. Reynolds
1967 "A word count program for language." Behavioral Scientist 12 (July):344.

Grimshaw, Allen
1969 "Sociolinguistics and the sociologist." The American Sociologist 4(Nov.):312-320.

Griswold, R. E., J. P. Poage, and I. P. Polonsky
1968 The SNOBOL4 Programming Language and Its Relation to SNOBOL3. Gerald King (ed.). Statistical Laboratory and Computing Center, University of Oregon, Eugene, Oregon, (mimeo).

Harway, N. and H. Iker
1964 "Computer analysis of content in psychotherapy." Psychological Reports 14:720-722.

International Business Machines, Inc.
1967 A Guide to PL/1 for Fortran Users, Form C20-1637. New York: International Business Machines Corp.

1967 IBM System 360, PL/1 F-Level Compiler Program Logic Manual, Form C28-6589. New York: International Business Machines Corp.

1967 IBM System 360, Operation System, PL/1 (F) Programmer's Guide, Form C28-6594. New York: International Business Machines Corp.

1967 IBM System 360, PL/1 Reference Manual, Form C28-8201. New York: International Business Machines Corp.

1968 A PL/1 Primer, Form C28-6808. New York: International Business Machines Corp.

1968 IBM System 360, PL/1 Subroutine Library, Computational Subroutines, Form C28-6590. New York: International Business Machines Corp.

1968 Fortran IV Language, Form C28-6515. New York: International Business Machines Corp.

Janda, Kenneth
1964 "Keyword indexes for the behavioral sciences." American Behavioral Scientist 7(June):55-58.

1965 Data Processing. Evanston, Ill.: Northwestern University Press.

Janden, B. Douglas
1966 "A system for content analysis by computer of international communications for selected categories of action." American Behavioral Scientist 10(March):28-32.

Kish, Leslie
1965 Survey Sampling. New York: Wiley and Sons.

Morris, Charles
1946 Signs, Language and Behavior. Englewood Cliffs, N.J.: Prentice-Hall, Inc.

Nie, Norman and Dale Bent

1968 Statistical Package for the Social Sciences, Provisional Users Manual. Palo Alto, Calif.: Stanford University (mimeo).

Ogden, C. K. and I. A. Richards
1923 The Meaning of Meaning. New York: Harcourt, Brace & World, Inc.

Pellegrin, Roland J.
1969 An Orientation to the Attributes Projects. Eugene, Oregon: CASEA, University of Oregon (mimeo).

Pellegrin, Roland J., Charles Dudley, and Keith Smith
1969 "The decision-making structure of schools." Paper presented at the Annual Meeting of the American Educational Research Association, Los Angeles, February, 1969.

Pylyshyn, Zenon
1969 "FINDSIT: A computer program for language research." Behavioral Scientist 14(May):248-251.

Riley, Matilda White
1963 Sociological Research: A Case Approach. New York: Harcourt, Brace & World, Inc.

Rummel, R. J.
1967 "Understanding factor analysis." Journal of Conflict Resolution 11:444-480.

Runkel, Philip
1965 Some Recent Ideas in Research Methodology. CASEA, University of Oregon, Eugene, Oregon, (mimeo).

Schench, Erwin and Philip Stone
1964 "The general inquirer approach to an international retrieval system for survey archives." American Behavioral Scientist 7(June): 23-28.

Shoemaker, David
1968 "A FORTRAN IV program for analysis of sociometric data." Behavioral Scientist 13(July):346.

Spence, Donald
1969 "PL/1 programs for content analysis." Behavioral Scientist 14 (Sept.):432-434.

Starkweather, J. and J. Decker
1964 "Computer analysis of interview content." Psychological Reports 15:875-882.

Stehr, Nico, George Lewis, and Roland J. Pellegrin
1969 "Task differentiation in elementary schools: an exploratory analysis." Paper presented at the Annual Meeting of the American Educational Research Association, Los Angeles, February, 1969.

Stoloff, Peter
1969 "PEER: A peer rating and sociometric data analyzer." Behavioral Scientist 14(May):253.

Stone, P. J., D. C. Dunphy, M. S. Smith, and D. M. Ogilvie
1966 The General Inquirer: A Computer Approach to Content Analysis. Cambridge, Mass.: M.I.T. Press.

Struble, George
1968 Assembler Language Programming: The IBM System 360. 2nd edition, prepublication draft.

Sussman, Marvin and Marie Haug
1967 "Human and mechanical error--an unknown quality in research." American Behavioral Scientist 11(Nov.-Dec.):55-56.

Vinsonhaler, John
1967 "BIRS: A system of general purpose computer programs for information retrieval in the behavioral sciences." American Behavioral Scientist 10(Feb.):12-24.

Wilcox, Allen, Davis Bobrow and Douglas Bwy
1967 "SESAR--automation in an intermediate stage of survey research." American Behavioral Scientist 10(Jan.):8-11.

APPENDIX ONE

THE STGPROC PROGRAM

The following pages contain a computer printout of the STGPROC program with general comment cards included.

```
/*PROGRAM 'STGPROC' DESIGNED FOR MANIPULATION OF OPEN-ENDED STRING DATA.
  DEVELOPED 1968 - 1969 AT THE UNIVERSITY OF OREGON BY GEORGE H. LEWIS
  UNDER THE AUSPICES OF THE CENTER FOR THE ADVANCED STUDY OF EDUCATIONAL
  ADMINISTRATION IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS FOR THE
  DEGREE OF DOCTOR OF PHILOSOPHY IN SOCIOLOGY. */

STGPROC:  PROC OPTIONS (MAIN);

/* 'LESBLKS' IS A FUNCTION THE PURPOSE OF WHICH IS TO DELETE ALL LEADING
  AND FOLLOWING BLANKS FROM A CHARACTER STRING USED AS THE ARGUMENT
  (CHK). */

LESBLKS:  PROC  (CHK) VAR CHAR (320);
    DCL
        CHK VAR CHAR (*);
B1: TO = LENGTH (CHK);
    T = INDEX (CHK, ' ');
    IF T = 1 THEN DO;
        CHK = SUBSTR (CHK,2);
        GO TO B1;
        END;
    T = INDEX (SUBSTR (CHK, TO) , ' ');
    IF T = 1 THEN DO;
        CHK = SUBSTR (CHK,1, TO - 1);
        GO TO B1;
        END;
B2: RETURN (CHK);
    END LESBLKS;



PRGM:  PROC;



/* 'REARR' IS A SUBROUTINE USED TO RESTORE AN ARRAY OF CHARACTER
  STRINGS TO ITS ORIGINAL SEQUENCE AFTER ITS HAVING BEEN RANKED
  BY THE SUBROUTINE 'RANK' IN TERMS OF A CORRESPONDING ARRAY
  OF ARITHMETIC VALUES. */

REARR:  PROC;
    DCL TEMP VAR CHAR (NUMARR (13));
    LS = THES_QUES_COL (IP2,3);
    DO X = 1 TO HBOUND (THES,1);
        DO K = 1 TO HBOUND (THES,2);
            IF THES (X,K,LS) = '' THEN GO TO RE2;
RE1:        NU4 = SUBSTR (THES (X,K,LS),1,2);
            I = NU4;
            NU4 = SUBSTR (THES (X,K,LS),3,2);
            J = NU4;
            IF I = X & J = K THEN GO TO RE2;
            TEMP = THES (I,J,LS);
            THES (I,J,LS) = THES (X,K,LS);
            THES (X,K,LS) = TEMP;
            GO TO RE1;
RE2:        END;
    END REARR;
```

```
/* 'GUT1' IS A SUBROUTINF THAT OPERATES UPON CERTAIN INSTRUCTION CARDS,
   FILLING AN ARRAY CREATED FOR THIS PURPOSE WITH THE CHARACTER STRINGS
   IT FINDS UPON THE INSTRUCTION CARD. THESE STRINGS ARE SEPERATED BY
   COMMAS (,); THE LIST BEGINS WITH AN EQUAL SIGN (=) AND IS TERMINATED
   BY A PERIOD (.). IF ANY OF THESE ABOVE THREE SIGNS IS DESIRED AS ONE
   OF THE CHARACTER STRINGS, IT MAY BE SO DESIGNATED BY ENCLOSING IT IN
   SINGLE QUOTES. */

GUT1:  PROC (DELIM);
    DCL
        DELIM (*) VAR CHAR (*);
        DO HM = 1 TO HBOUND (DELIM,1);
        TSTG = SUBSTR (TSTG, IN2 + 1);
        T = ',';
        CALL GET (IQ7);
        IF IQ7 = 0 THEN DO;
            T = ',';
            CALL GET (IQ7);
        IF IQ7 = 0 THEN GO TO ENDGT;
            END;
        DELIM (HM) = LESBLKS (SUBSTR (TSTG, 1, IQ7 - 1));
        T = ',';
G1: LOT = LT || T || LT;
    U = INDEX (DELIM (HM), LOT);
        IF U > 0 THEN DO;
        TO = LENGTH (DELIM(HM));
        IF TO = 3 THEN DO;
            DELIM (HM) = T;
            GO TO G1;
            END;
            DELIM (HM) = SUBSTR (DELIM (HM), 1, U - 1)  || T ||
            SUBSTR (DELIM (HM), U + 3);
            GO TO G1;
            END;
        IF T = ',' & U = 0 THEN DO;
            T = '.';
            GO TO G1;
            END;
        IF T = '.'  & U = 0 THEN DO;
            T = '=';
            GO TO G1;
            END;
    IN2 = IQ7;
    END;
ENDGT:  END GUT1;


/* 'GUT2' IS A SUBROUTINE THAT OPERATES UPON CERTAIN INSTRUCTION CARDS,
   FILLING AN ARRAY CREATED FOR THIS PURPOSE WITH THE ARITHMATIC STRINGS
   IT FINDS UPON THE INSTRUCTION CARD. THESE STRINGS ARE SEPERATED BY
   COMMAS (,); THE LIST BEGINS WITH AN EQUAL SIGN (=) AND IS TERMINATED
   BY A PERIOD (.). */

GUT2:  PROC (ARY,JOT);
    DCL
        ARY (*,*) FIXED DEC (3,0),
        JOT FIXED BIN (15,0),
        JQ1 VAR CHAR (3);
    DO HM = 1 TO HBOUND (ARY,1);
    IQ7 = INDEX (SUBSTR (TSTG, IN2 + 1), ',');
        IF IQ7 = 0 THEN IQ7 = IN3 - IN2;
        JQ1 = LESBLKS (SUBSTR (TSTG, IN2 + 1,
          IQ7 - 1));
        NUM1 = JQ1;
        ARY (HM, JOT) = NUM1;
        IN2 = IN2 + IQ7;
    END GUT2;


/* 'GET' IS A SUBROUTINE THAT DETERMINES THE POSITION OF THE CHARACTER
  STRING USED AS THE ARGUMENT (I) IN A STRING IN WHICH IT IS EMBEDDED.
  IT IGNORES OCCURANCES OF THE STRING THAT ARE ENCLOSED IN
  SINGLE QUOTES. */

GET:  PROC (I);
    DCL
        I FIXED BIN (15,0);
    X = 1;
    LOT = LT  || T  || LT;
G1:  I = INDEX (SUBSTR (TSTG, X), T);
    J = INDEX (SUBSTR (TSTG, X) , LOT);
    IF J > 0 & I = J + 1 THEN DO;
        IF X = 1 THEN X = 0;
        X = X + J + 3;
        GO TO G1;
        END;
    IF I > 0 THEN I = I + X - 1;
    END GET;


/* 'PROC_CALL' IS A SUBROUTINE THAT, IF CALLED, WILL IN TURN CALL THE
  SUBROUTINES 'ITEM_CT,' 'CLASSIFY,' AND/OR 'SOCIO' TO ACT UPON THOSE
  QUESTIONS SO DESIGNATED BY THE PROPER INSTRUCTION CARDS. */

PROC_CALL:  PROC;
    DO IP2 = 1 TO NUMARR (9);
        IF QNUM2 = ITEM_QUES_COL (IP2,1) THEN CALL ITEM_CT;
        END;
    DO IP2 = 1 TO NUMARR (12);
        IF QNUM2 = THES_QUES_COL (IP2,1) THEN CALL CLASSIFY;
        END;
    DO IP2 = 1 TO NUMARR (18);
        IF QNUM2 = NAME_QUES_COL (IP2,1)  |  (QNUM2 =
        NOM_QUES_COL (IP2,1) & OP = 0) THEN CALL SOCIO;
    END PROC_CALL;


/* 'SUB' IS A SUBROUTINE THAT: 1) INITIALIZES ALL COUNTER VALUES TO ZERO
  AND ALL CHARACTER ARRAY LISTS TO NULL STRINGS; 2) OUTPUTS VARIOUS PAGE
  HEADINGS. IT IS CALLED WHEN BEGINNING  COMPUTER CONSIDERATION OF
  EACH SECTION OF DATA DESIGNATED AS INDEPENDENT OF THE OTHERS. */

SUB:  PROC (READ);
    DCL  READ LABEL;
    PUT EDIT ('NEXT SUBSAMPLE') (PAGE, LINE (30), X (50), A);
  PUT PAGE;
  IF FACT_ARR (1,1) ¬= 0 THEN DO;
    PUT EDIT ('PRINTOUT OF FACTOR ANALYSIS DATA CARDS:') (A);
    PUT FILE (OUT) EDIT ('FACTOR ANALYSIS PUNCHED DATA FOLLOWS')
    (A(80));
    END;
  ID = '';
  OP = 0;   Q = 0;   P = 0;   N5 = 0;
  SM = 0;   RK = 0;   I241 = 0;   NUBRES = 0;
  F = 0;   RES_NUM = 0;   TSTID = '';  IQ = 0;
  GO TO READ;
  END SUB;


/* 'RANK' IS A SUBROUTINE THAT RANKS A THREE-DIMENSIONAL ARITHMETIC ARRAY
  ARGUMENT (B) FROM HIGHEST TO LOWEST VALUES.  AT THE SAME TIME, IT
  PERFORMS CORRESPONDING ADJUSTMENTS ON THE THREE-DIMENSIONAL
  CHARACTER ARRAY ARGUMENT (A). */

RANK:  PROC (A,B);
  DCL
    A (*,*,*) VAR CHAR (*),
    B (*,*,*) FIXED BIN (15,0),
    TEMP_A VAR CHAR (60),
    TEMP_B FIXED BIN (15,0);
  DO X = 1 TO HBOUND (A,1);
    DO U = 1 TO HBOUND (A,2);
RK1:      DO I = 1 TO HBOUND (A,1);
RK2:        DO J = 1 TO HBOUND (A, 2);
            IF X = I & U > J THEN GO TO RK3;
            IF X > I THEN GO TO RK4;
            IF B (X,U,IP1) < B (I,J,IP1) THEN DO;
             TEMP_A = A (X,U,IP2);
             TEMP_B = B (X,U,IP1);
             A (X,U,IP2) = A (I,J,IP2);
             B (X,U,IP1) = B (I,J,IP1);
             A (I,J,IP2) = TEMP_A;
             B (I,J,IP1) = TEMP_B;
             I = 1;
             J = 1;
             GO TO RK1;
             END;
RK3:        END RK2;
RK4:      END RK1;
  END RANK;


/* 'ITEM_CT' IS A SUBROUTINE THAT, FOR EACH SEPERATE ACTIVATION, COUNTS
  ALL UNIQUE ENTRIES UNDER A QUESTION RESPONSE TYPE, LISTS THEM, THEIR
  RANKED FREQUENCY OF OCCURANCE, AND VARIOUS ASSOCIATED FREQUENCIES AND
  PERCENTAGES.  IT ALSO LISTS THE ORIGINS OF UNIQUE ITEMS AS A SPELLING
  CHECK AND MONITORS FOR ERRORS IN THE INSTRUCTION CARDS PERTAINING TO
  THIS SUBROUTINE.  IF ACTIVATED, THE SUBROUTINE 'SPSS' IS CALLED FROM
  WITHIN THIS SUBROUTINE.  */

ITEM_CT:  PROC;
  IF P(IP2) = 0 THEN DO;
    DO K = 1 TO HBOUND (ITEM,1);
```

```
            ITEM (K,1,IP2) = '';
            ITEM_DIS (K,1,IP2) = 0;
            END;
        END;
    IF P(IP2) = -1 THEN GO TO ENDIT;
    IF OP = 1 THEN GO TO IT2;
    NUBRES(IP2) = NUBRES(IP2) + 1;
    L = ITEM_QUES_COL (IP2,2);
    J = 1;
    DO I = 1 TO QUES_SIZE (QNUM2,1);
        IP1 = 0;
        IF QUES_ARRAY (I,L) = '' THEN GO TO IT1;
        DO K = 1 TO P (IP2);
            IF ITEM (K,J,IP2) = '' THEN GO TO IT1;
            IPO = INDEX (QUES_ARRAY (I,L), ITEM (K,J,IP2));
            IF IPO = 1 & LENGTH (QUES_ARRAY(I,L)) =
            LENGTH (ITEM (K,J,IP2)) THEN DO;
                IF RUN_ACT (1,1) ¬= 0 THEN DO;
                  DO LS = 1 TO HBOUND (RUN_ACT,1);
                    IF IP2 = RUN_ACT (LS,1) THEN DO;
                    RK = K;  CALL SPSS;  END;  END;
                  END;
                ITEM_DIS (K,J,IP2) = ITEM_DIS (K,J,IP2) + 1;
                IP1 = 1;
                END;
            END;
IT1:    IF IP1 = 0 THEN DO;
            IF P(IP2) > HBOUND (ITEM,1) THEN DO;
                P(IP2) = P(IP2) - 1;
                PUT EDIT ('ERROR: FOR QUESTION #' ||  QNUM1  ||
                ', ''DISCRETE ITEM ESTIMATE'' ON INSTRUCTION CARD     IS
 NOT LARGE ENOUGH. MAXIMUM REACHED ON ID # '  ||  ID) (PAGE,
                LINE(10), A);
                P(IP2) = -1;
                GO TO ENDIT;
                END;
            P(IP2) = P(IP2) + 1;
            ITEM (P(IP2), J, IP2) = QUES_ARRAY (I,L);
            ITEM_DIS (P(IP2), J, IP2) = 1;
            LST (P(IP2), 1, IP2) = QUES_ARRAY (I,L);
            LST (P(IP2), 2, IP2) = ID;
            IF RUN_ACT (1,1) ¬= 0 THEN DO;
                DO LS = 1 TO HBOUND (RUN_ACT,1);
                  IF IP2 = RUN_ACT (LS,1) THEN DO;
                    RK = P(IP2);
                    CALL SPSS;
                    END;
                  END;
                END;
            END;
        END;
    GO TO ENDIT;

/* TOTALLING AND OUTPUT ACTIONS AT THE END OF EACH DATA SET SPECIFIED AS
  COMPLETE BEGIN HERE.  */

IT2:  PO = 0;
    DO K = 1 TO HBOUND (ITEM_DIS, 1);
        PO = PO + ITEM_DIS (K,1,IP2);
        END;
    IP1 = IP2;
    CALL RANK (ITEM, ITEM_DIS);
    PUT EDIT ('ITEM_CT SUBROUTINE: RUN #', IP2,
    ' ORIGINS OF NEW ITEMS', 'ITEM', 'ID STRING') (PAGE,A,F(3),
    SKIP(10),A,SKIP(4),A(44),A);
    DO I = 1 TO P(IP2);
        PUT EDIT (LST(I,1,IP2), LST(I,2,IP2)) (SKIP(2),A(44),A);
        END;
    PUT EDIT ('RANKED ITEMS', 'TOTAL CHOICES = ', PO,
    'TOTAL CHOOSERS = ', NUBRES(IP2), 'ITEM', 'DISTRIBUTION',
    '% OF CHOICES', '% OF CHOOSERS')
    (PAGE, SKIP(2), A, 2 (SKIP(2), A, F(7)), SKIP(4), 4 (A(35)));
    DO K = 1 TO P(IP2);
        ON ZERODIVIDE PERCENT = 0;
        PERCENT1 = ((ITEM_DIS(K,1,IP2) / NUBRES(IP2)) * 100);
        ON ZERODIVIDE PERCENT2 = 0;
        PERCENT2 = ((ITEM_DIS (K,1,IP2) / PO     ) * 100);
        REVERT ZERODIVIDE;
    PUT EDIT (ITEM (K,1,IP2), ITEM_DIS(K,1,IP2), PERCENT2,PERCENT1)
    (SKIP(2), A(40), F(5), 2 (X(30), F(3)));
    END;
    IF RUN_ACT (1,1) ¬= 0 THEN DO;
        DO LS = 1 TO HBOUND (RUN_ACT,1);
            IF IP2 = RUN_ACT (LS,1) THEN DO;
                RK = -2;
                CALL SPSS;
                END;
            END;
        END;
ENDIT:  END ITEM_CT;


/* 'CLASSIFY' IS A SUBROUTINE THAT, FOR EACH SEPERATE ACTIVATION,
  CLASSIFIES ALL ENTRIES UNDER A QUESTION RESPONSE TYPE IN TERMS OF A
  CATEGORY SYSTEM SET UP AS THE THESAURUS UTILIZED FOR THE ACTIVATION.
  THIS THESAURUS IS ENTERED BY MEANS OF INSTRUCTION CARDS.  THE
  SUBROUTINE LISTS THE RESULTS AND THEIR DISTRIBUTIONS BY CATEGORY
  AND ITEM, RANKED BY CATEGORY, RANKED BY ITEM, AND LISTS ALSO VARIOUS
  ASSOCIATED FREQUENCIES AND PERCENTAGES.  IT ALSO LISTS ANY ITEMS
  NOT COVERED BY THE THESAURUS AND THEIR LOCATION (AS SPELLING AND
  INCLUSION CHECKS) AND MONITORS FOR ERRORS IN THE INSTRUCTION CARDS
  PERTAINING TO THIS SUBROUTINE.  IF ACTIVATED, THE SUBROUTINE 'FACT_AN'
  IS CALLED FROM WITHIN THIS SUBROUTINE.  */

CLASSIFY:  PROC;
    IF Q (IP2) = 0 THEN DO;
        DO I = 1 TO HBOUND (THES,1);
            DO J = 1 TO HBOUND (THES,2);
                THES_CT (I,J,IP2) = 0;
                END;
            END;
        NOTFIND (1,IP2) = '';
        NOTFIND (2,IP2) = '';
        Q (IP2) = 1;
        END;
    LS = THES_QUES_COL (IP2,3);
    IF OP = 1 THEN GO TO CL5;
    IF FACT_ARR(1,1) ¬= 0 THEN DO;
        LK = 0;
    DO I = 1 TO HBOUND (FACT_ARR,1);
      IF IP2 = FACT_ARR (I,1) THEN CALL FACT_AN;
    END;
    END;
  DO A = 1 TO QUES_SIZE (QNUM2, 1);
    IP1 = 0;
    DO I = 1 TO HBOUND (THES,1);
    IF THES (I,1,LS) = '' THEN GO TO CL2;
      DO J = 1 TO HBOUND (THES,2);
        IF QUES_ARRAY (A,THES_QUES_COL(IP2,2)) = '' THEN GO TO
        CL4;
      IF THES (I,J,LS) = '' THEN GO TO CL1;
        IP3 = 0;
        IF LENGTH (QUES_ARRAY(A,THES_QJES_COL(IP2,2))) =
      LENGTH (THES(I,J,LS)) - 4
        THEN IP3 = 1;
        IPO = INDEX (QUES_ARRAY(A,THES_QUES_COL(IP2,2)),
      SUBSTR (THES (I,J,LS),5));
        IF IPO = 1 & IP3 = 1 THEN DO;
         THES_CT (I,J,IP2) = THES_CT (I,J,IP2) + 1;
         IP1 = IP1 + 1;
      END;
      END;
CL1:  END;
CL2:   IF IP1 = 0 THEN DO;
      NOTFIND (1,IP2) = NOTFIND (1,IP2)  ||
      QUES_ARRAY (A, THES_QUES_COL (IP2,2))  ||  ',';
      NOTFIND (2,IP2) = NOTFIND (2,IP2)  ||  ID  ||  ',';
      SM (IP2) = SM(IP2) + 1;
    END;
    END;
CL4:  GO TO END_CL;

/* TOTALLING AND OUTPUT ACTIONS AT THE END OF EACH DATA SET SPECIFIED AS
 COMPLETE BEGIN HERE.  */

CL5:  SMCT = 0;
  THES_CAT = 0;
    TH_CAT = '';
  HM = 1;
CL6:  DO K = 1 TO HBOUND (THES,1);
    DO L = 1 TO HBOUND (THES,2);
      SMCT = SMCT + THES_CT (K,L,IP2);
    END CL6;
  PERCENT1 = 0;
  PERCENT2 = 0;
  SM(IP2) = SM(IP2) + SMCT;
  IP3 = IP2;
  PUT EDIT ('SUBROUTINE CLASSIFY RESULTS:  RUN # ', IP2,
  'RESULTS ORDERED BY THESAURUS') (PAGE,A,F(2),SKIP(10),A);
CL7:  PUT SKIP (4) EDIT ('ITEM', 'DISTRIBUTION', 'PERCENT MATCHED',
  'PERCENT OF TOTAL') (4(A(30)));
CL8:  DO I = 1 TO HBOUND (THES,1);
    IF THES (I,1,LS) = '' THEN GO TO CL10;
    IF HM ¬= 0 THEN PUT EDIT ('CATEGORY ', I)
    (SKIP(4),A,F(3));
    DO J = 1 TO HBOUND (THES,2);
      IF THES (I,J,LS) = '' THEN GO TO CL9;
      ON ZERODIVIDE PERCENT1 = 0;
      PERCENT1 = ((THES_CT(I,J,IP2) * 100) / SMCT);
```

```
               ON ZERODIVIDE PERCENT2 = 0;
               PERCENT2 = ((THES_CT(I,J,IP2) * 100) / SM(IP2));
       PUT EDIT (SUBSTR (THES (I,J,LS),5), THES_CT(I,J,IP2),PERCENT1,
           PERCENT2) (SKIP(2),A(36),F(5),2(X(28),F(4)));
CL9:     END CL8;
CL10:   IF HM = 0 THEN GO TO CL14;
       PUT EDIT ('RANKED DISTRIBUTION OF RESULTS:','BY CATEGORY')
       (PAGE,A,SKIP(4),A);
       DO I = 1 TO HBOUND (THES,1);
           IF THES (I,1,LS) = '' THEN GO TO CL12;
           HM = 0;
           DO J = 1 TO HBOUND (THES,2);
               IF THES (I,J,LS) = '' THEN GO TO CL11;
               HM = THES_CT (I,J,IP2) + HM;
               END;
CL11:          THES_CAT (I,1,1) = HM;
           VK = I;
           TH_CAT (I,1,1) = SUBSTR (VK,4);
               END;
CL12: IP1 = 1;
       IP2 = 1;
         CALL RANK (TH_CAT, THES_CAT);
       PUT SKIP (4) EDIT ('CATEGORY', 'DISTRIBUTION', 'PERCENT MATCHED',
       'PERCENT OF TOTAL') (4(A(30)));
       DO I = 1 TO HBOUND (THES,1);
               IF TH_CAT (I,1,1) = '' THEN GO TO CL13;
               ON ZERODIVIDE PERCENT1 = 0;
               PERCENT1 = ((THES_CAT(I,1,1) * 100) / SMCT);
               ON ZERODIVIDE PERCENT2 = 0;
               PERCENT2 = ((THES_CAT(I,1,1) * 100) / SM (IP3));
           PUT EDIT ('CATEGORY ', TH_CAT(I,1,1),THES_CAT(I,1,1),
               PERCENT1,PERCENT2) (SKIP(2),A,F(2),3(X(22),F(4),X(4)));
           END;
CL13:  PUT EDIT ('TOTAL MATCHES =', SMCT, 'TOTAL CHOICES =', SM(IP3))
       (2(SKIP(4),A,F(3)));
       IF NOTFIND (1,IP3) = '' THEN DO;
           NOTFIND (1,IP3) = 'ALL ARE COVERED';
           NOTFIND (2,IP3) = 'NONE';
           END;
       IP2 = THES_QUES_COL (IP3,3);
       IP1 = IP3;
       CALL RANK (THES, THES_CT);
       PUT EDIT ('BY ITEM') (PAGE,A);
       HM = 0;
           IP2 = IP1;
       GO TO CL7;
CL14: PUT EDIT ('RESPONSES NOT COVERED BY THESAURUS:', NOTFIND(1,IP2),
       'ID NUMBERS OF NOT COVERED RESPONSES:', NOTFIND (2,IP2))
       (PAGE,LINE(10),2(SKIP(4),A,X(6),A));
              REVERT ZERODIVIDE;
       IF NUMARR (6) = 0 THEN CALL REARR;
END_CL:  END CLASSIFY;


/* 'SOCIO' IS A SUBROUTINE THAT, FOR EACH SEPERATE ACTIVATION,
  RELATES THE ENTRIES IN TWO SPECIFIED QUESTION RESPONSE TYPES TO EACH
  OTHER IN SOCIOMETRIC FASHION.  ONE RESPONSE TYPE CAN BE THOUGHT OF AS
  THE NOMINEE, THE OTHER AS A LIST OF NOMINATIONS.  THE SUBROUTINE LISTS
  RECIPROCAL NOMINATIONS AND ONE-WAY NOMINATIONS WHEN RECIPROCALITY IS
  AND IS NOT POSSIBLE, AS WELL AS ASSOCIATED FREQUENCIES.  IT ALSO LISTS
  THE NOMINEES AND THEIR NOMINATIONS AS A SPELLING CHECK AND MONITORS
  FOR ERRORS IN THE INSTRUCTION CARDS PERTAINING TO THIS SUBROUTINE. */

SOCIO:  PROC;
   DCL
       CHECK2 VAR CHAR (NUMARR(27)),
       CHOICETOT (2) FIXED BIN (15,0),
       CHOICE1 (3) FIXED BIN (15,0);
   IF F (IP2) = -20 THEN GO TO END_SOCIO;
   IF OP = 1 THEN GO TO SOC8;
   IF QNUM2 = NAME_QUES_COL (IP2,1) THEN DO;
       TSTID (IP2,1) = ID;
       NAME (IP2) = QUES_ARRAY (NAME_QUES_COL (IP2,2),1);
       IPO = INDEX (TSTID (IP2,1), TSTID (IP2,2));
       IF IPO = 1 THEN GO TO SOC2;
       GO TO END_SOCIO;
       END;
   IF QNUM2 = NOM_QUES_COL (IP2, 1) THEN DO;
       TSTID (IP2,2) = ID;
       DO I = 1 TO NOM_QUES_COL (IP2,3);
           NOMIN (I,IP2) = '';
           IF QUES_ARRAY (I,NOM_QUES_COL(IP2,2)) = '' THEN GO TO SOC1;
           NOMIN (I,IP2) = QUES_ARRAY (I,NOM_QUES_COL(IP2,2));
           END;
SOC1:   IPO = INDEX (TSTID(IP2,1), TSTID (IP2,2));
       IF IPO = 0 THEN GO TO END_SOCIO;
       END;
SOC2:  RES_NUM (IP2) = RES_NUM (IP2) + 1;
   PUT EDIT ('SPELLING CHECK FOR SUBROUTINE SOCIO: RUN #',
   IP2, 'ID NUMBER= ' || ID, 'NAME = ' || NAME (IP2))
   (PAGE,A,F(3),2(SKIP(4),A));
   DO I = 1 TO HBOUND (NOMIN,1);
       IF NOMIN (I,IP2) = '' THEN GO TO SOC3;
       PUT SKIP (4) EDIT ('NOMINATION #', I, ' = ' ||
       NOMIN (I,IP2)) (A,F(3),A);
SOC3:   END;
   IF F (IP2) = 0 THEN DO;
       DO I = 1 TO HBOUND (LIST,1);
           LIST (I,IP2) = '';
           END;
       GO TO SOC5;
       END;
SOC4:  DO I = 1 TO NOM_QUES_COL (IP2,3);
       IF NOMIN (I,IP2) = '' THEN GO TO SOC5;
       QUES = NOMIN (I,IP2) || '&' || NAME (IP2);
       DO J = 1 TO F (IP2);
           IP1 = INDEX (LIST (J,IP2), QUES);
           IF IP1 > 0 THEN DO;
               LIST (J,IP2) = LIST (J,IP2) || '*';
               NOMIN (I,IP2) = '**';
       END SOC4;
SOC5:  DO I = 1 TO NOM_QUES_COL (IP2,3);
       IF NOMIN (I,IP2) = '**' THEN DO;
           LIST ((F(IP2) + I), IP2) = NAME (IP2) || '&';
           GO TO SOC6;
           END;
       IF NOMIN (I,IP2) = '' THEN DO;
           F (IP2) = F (IP2) - 1;
           GO TO SOC7;
```

```
          END;
      LIST ((F(IP2) + I), IP2) = NAME (IP2) || '&' || NOMIN (I,IP2)
      || '$';
SOC6:   END;
    IF F (IP2) > HBOUND (LIST,1) THEN GO TO SOCERR;
SOC7:  F (IP2) = F(IP2) + I;
    GO TO END_SOCIO;

/* TOTALLING AND OUTPUT ACTIONS AT THE END OF EACH DATA SET SPECIFIED AS
  COMPLETE BEGIN HERE.  */

SOC8:  DO I = 1 TO F(IP2);
      IPO = INDEX (LIST (I,IP2), '&');
      QUES = SUBSTR (LIST (I,IP2), 1, IPO - 1);
      DO J = 1 TO F (IP2);
          IPO = INDEX (LIST (J,IP2), '&');
          CHECK2 = SUBSTR (LIST (J,IP2), IPO + 1);
          IPO = INDEX (CHECK2, QUES);
          IP1 = INDEX (CHECK2, '*');
          IF IPO > 0 & IP1 = 0 THEN LIST (J,IP2) =
          LIST (J,IP2) || '#';
      END SOC8;
    PUT EDIT ('RESULTS OF SUBROUTINE SOCIO: RUN #',
    IP2, 'RECIPROCAL CHOICES') (PAGE,A,F(3),SKIP(10),A);

/* THE SYMBOLS THAT MAKE UP 'QI' HAVE BEEN PLACED AT THE END OF EACH STORED
  STRING IN 'LIST' BY ABOVE STATEMENTS IN THE SUBROUTINE.  THEY SIGNIFY:
  $* = RECIPROCAL CHOICE; $# = ONE-WAY CHOICE WITH POSSIBLE
  RECIPROCALITY; $ = ONE-WAY CHOICE WITH NO POSSIBLE RECIPROCALITY. */

    X = 1;
    QI = '$*';
    CHOICE1 = 0;
SOC9:  DO I = 1 TO F (IP2);
      IPO = INDEX (LIST (I,IP2), QI);
      IF IPO > 0 THEN DO;
          LIST (I,IP2) = SUBSTR (LIST (I,IP2),1,IPO - 1);
          IF X = 1 THEN LIST (I,IP2) = LIST (I,IP2) || '*';
          IP1 = INDEX (LIST (I,IP2), '&');
          QUES = SUBSTR (LIST (I,IP2), 1, IP1 - 1);
          CHECK2 = SUBSTR (LIST (I,IP2),IP1+1,(IPO-IP1)-1);
          CHOICE1 (X) = CHOICE1 (X) + 1;
          IF X ¬= 1 THEN PUT SKIP (4) EDIT (QUES ||
          ' CHOOSES ' ||CHECK2) (A);
          ELSE IF X = 1 THEN PUT SKIP (4) EDIT (QUES ||
          ' CHOOSES ' || CHECK2 || ' AND VICE-VERSA') (A);
      END SOC9;
    IF X = 1 THEN CHOICE1 (X) = CHOICE1 (X) * 2;
    PUT SKIP (6) EDIT ('NUMBER OF CHOICES =', CHOICE1 (X)) (A,F(4));
    X = X + 1;
    IF X = 2 THEN DO;
      PUT SKIP (10) LIST ('ONE-WAY CHOICES WHEN RECIPROCALITY WAS POS
SIBLE');
      QI = '$#';
      GO TO SOC9;
      END;
    IF X = 3 THEN DO;
      PUT SKIP (10) LIST ('ONE-WAY CHOICES WHEN RECIPROCALITY WAS NOT
 POSSIBLE');
      QI = '$';
    GO TO SOC9;
    END;
  CHOICETOT = 0;
  CHOICETOT (1) = SUM (CHOICE1);
  CHOICETOT (2) = NOM_QUES_COL (IP2,3) * RES_NUM (IP2);
  PUT EDIT ('POSSIBLE NUMBER OF CHOICES PER PERSON =',
  NOM_QUES_COL (IP2,3), 'NUMBER OF RESPONDENTS =',
  RES_NUM (IP2), 'NUMBER OF ACTUAL CHOICES = ',
  CHOICETOT (1), 'NUMBER OF POSSIBLE CHOICES =',
  CHOICETOT (2)) (PAGE, 4(SKIP(4),A,F(4)));
  GO TO END_SOCIO;
SOCERR:  PUT EDIT ('ERROR: IN SOCIO RUN #', IP2,
  ' THE INSTRUCTION CARD #20 (SOCIO NUMBER OF NOMINEES) IS NOT LARGE
ENOUGH. INCREASE THE NUMBER.') (SKIP(4),A,F(4),A);
  F (IP2) = -20;
END_SOCIO:  END SOCIO;



/* 'FACT_AN' IS A SUBROUTINE THAT, FOR EACH SEPERATE ACTIVATION, PUNCHES
 A SET OF DATA CARDS AMENABLE TO USE WITH THE BIO-MEDICAL SERIES
 FACTOR ANALYSIS PROGRAM OR OTHER PROGRAMS REQUIRING DATA IN SIMILAR
 FORM.  THIS DATA IS ALSO IN A FORM THAT CAN BE UTILIZED AS FURTHER DATA
 FOR THE STGPROC PROGRAM WITH NO REVISIONS OR REPUNCHING.  IT IS UTILIZED
 IN CONJUNCTION WITH THE 'CLASSIFY' SUBROUTINE.  */

FACT_AN:  PROC;
  J = I;
  L = I + LK;
    NUM3 = L + LK;
  VK = NUM3;
  NU3 = SUBSTR (VK,5);
  QUES = NU3 || ID;
  NUM3 = LENGTH (QUES);
  IF NUM3 < NUMARR (27) + 2 THEN DO;
    DO X = 1 TO (NUMARR (27) + 2) - NUM3;
      QUES = QUES || ' ';
      END;
    END;
  QUES = QUES || DELIM (1);
  NUM3 = HBOUND (QUES_SIZE,1) + (I - 1);
  VK = NUM3;
  NU3 = SUBSTR (VK,5);
  QUES = QUES || NU3 || DELIM (2);
  DO I = 1 TO HBOUND (THES,1);
    IF THES (I,1,LS) = '' THEN GO TO FA4;
    DO X = 1 TO HBOUND (THES,2);
      IF THES (I,X,LS) = '' THEN GO TO FA3;
      DO J = 1 TO QUES_SIZE (THES_QUES_COL (IP2,1), 1);
        IF QUES_ARRAY(J,THES_QUES_COL(IP2,2)) = '' THEN
        GO TO FA1;
        K = INDEX (SUBSTR (THES(I,X,LS),5),
        QUES_ARRAY(J,THES_QUES_COL(IP2,2)));
        IF K = 1 THEN DO;
          QI = '1';
          GO TO FA2;
          END;
        END;
FA1:      QI = '0';
FA2:      QUES = QUES || QI;
```

```
            END;
FA3:    END;
FA4:  QUES = QUES || DELIM (1);
FA5: K = LENGTH (QUES);
    IF K < 80 THEN DO;
        DO I = 1 TO 80 - K;
            QUES = QUES || ' ';
            END;
        K = 80;
        END;
    IF K = 80 THEN CSTG = QUES;
    IF K > 80 THEN DO;
        LK = L< + 1;
        CSTG = SUBSTR (QUES,1,80);
        PUT EDIT (CSTG) (SKIP (3),A);
        PUT FILE (OUT) EDIT (CSTG) (A);
    NUM3 = NUMARR (24) + (L - 1);
        VK = NUM3;
        NU3 = SUBSTR (VK,5);
        QUES = NU3 || SUBSTR (QUES,3,NUMARR(27) + 1) ||
        SUBSTR (QUES,81);
        GO TO FA5;
        END;
    PUT EDIT (CSTG) (SKIP(3),A);
    PUT FILE (OUT) EDIT (CSTG) (A);
    END FACT_AN;


/* 'SPSS' IS A SUBROUTINE THAT, FOR EACH SEPERATE ACTIVATION, CREATES A
  CODE FOR THE 'ITEM_CT' ACTIVATION UPON WHICH IT IS WORKING, OUTPUTS A
  CODE SHEET, AND A SET OF NUMERICALLY PUNCHED DATA CARDS FOR UTILIZATION
  WITH THE MANY OPTIONS OFFERED IN THE STANFORD SPSS PROGRAM OR OTHER
  PROGRAMS REQUIRING DATA IN SIMILAR FORM.  THIS SUBROUTINE IS UTILIZED
  IN CONJUNCTION WITH THE 'ITEM_CT' SUBROUTINE.  */

SPSS:  PROC;
    IF N5 = -3 THEN GO TO END_SPSS;
    IF RK = -2 THEN GO TO SP1;
    IF IQ (1) = 0 THEN DO;
        FACT_AR = '';
        FTD = '';
        MAX_COL = 0;
        MAX = 0;
        END;
    MAX (LS) = MAX (LS) + 1;
    IF FTD (LS) ¬= ID THEN DO;
        IQ (LS) = IQ (LS) + 1;
        MAX (LS) = 1;
        FTD (LS) = ID;
        END;
    IF IQ (LS) > HBOUND (FACT_AR, 1) THEN GO TO SPERR;
    IF MAX (LS) > MAX_COL (LS) THEN MAX_COL (LS) = MAX (LS);
    NUM3 = RK;
    VK = NUM3;
    NU5 = SUBSTR (VK,4);
    FACT_AR (IQ(LS), LS) = FACT_AR (IQ(LS), LS) || NU5;
    GO TO END_SPSS;
SP1:  PUT EDIT ('CODING FOR SUBROUTINE SPSS, RUN #', LS,
    'ON QUESTION #', QNUM2, 'ITEM', 'CODE')
  (PAGE, 2 (SKIP (4), A, F (3)), SKIP (6), A (40), A);
  DO LK = 1 TO P (IP2);
    PUT EDIT (ITEM(LK,1,IP2), LK) (SKIP(2), A(40), F(3));
    END;
  LK = MAX_COL (LS) * 3;
  PUT EDIT ('ITEM_CT RUN #', RUN_ACT (LS,1), ' IN QUESTION #', QNUM2,
  ' IS VARIABLE #', LS, ' IN THE PUNCHED OUTPUT WITH',
  MAX_COL (LS), ' SUBRESPONSES OF THREE COLUMNS APIECE MAKING A TOTAL
 FIELD LENGTH OF', LK) (SKIP(10),5 (A, F(3), X(2)));
  IF LS ¬= HBOUND (RUN_ACT,1) THEN GO TO END_SPSS;

/* TOTALLING AND OUTPUT ACTIONS AT THE END OF EACH DATA SET SPECIFIED AS
 COMPLETE BEGIN HERE.  */

  PUT FILE (OUT) EDIT ('SPSS PUNCHED DATA FOLLOWS') (A(80));
  PUT EDIT ('SPSS PRINTOUT OF DATA CARDS:') (PAGE,A);
SP2:  DO I = 1 TO IQ (LS);
    N5 = 1;
    QUES = '';
    NUM3 = I;
    VK = NUM3;
    NU3 = SUBSTR (VK,5);
    NUM3 = N5;
    VK = NUM3;
    NU4 = SUBSTR (VK,5);
    QUES = QUES || NU3  || NU4;
SP3:    DO J = 1 TO LS;
      LK = LENGTH (FACT_AR (I,J));
      IF LK ¬= (MAX_COL (J) * 3) THEN DO;
SP4:        DO K = 1 TO (MAX_COL (J) - LK);
         FACT_AR (I,J) = FACT_AR (I,J) || '000';
         END;
        END;
SP5:      LK = LENGTH (QUES);
      L = LENGTH (FACT_AR (I,J));
      IF L = 0 THEN GO TO SP7;
      IF L + LK > 80 THEN DO;
        QUES = QUES || SUBSTR (FACT_AR(I,J),1,(80-LK));
        FACT_AR(I,J) = SUBSTR (FACT_AR(I,J),(80-LK)+1);
        LK = 80;
        GO TO SP6;
        END;
      QUES = QUES || FACT_AR (I,J);
SP6:      IF LK = 80 THEN DO;
        CSTG = QUES;
        PUT FILE (OUT) EDIT (CSTG) (A);
    PUT EDIT (CSTG) (SKIP(3), A);
        N5 = N5 + 1;
        NUM3 = N5;
        VK = NUM3;
        NU4 = SUBSTR (VK,5);
        QUES = NU3 || NU4;
        GO TO SP5;
SP7:      END SP3;
    LK = LENGTH (QUES);
    IF LK < 80 THEN DO;
      DO J = 1 TO (80 - LK);
        QUES = QUES || ' ';
        END;
      END;
```

```
        CSTG = QUES;
        PUT FILE (OUT) EDIT (CSTG) (A);
        PUT EDIT (CSTG) (SKIP(3), A);
        END SP2;
    GO TO SP8;
SPERR:  PUT EDIT ('ERROR: INCREASE SIZE OF INSTRUCTION CARD 23 (SPSS NU
MBER OF RUNS).') (SKIP(4),A);
SP8:   N5 = -3;
END_SPSS:  END SPSS;



/* 'BREAK' IS A SUBROUTINE THAT BREAKS EACH QUESTION STRING UP AND
  PLACES ITS PARTS IN THE RELEVENT CELLS OF THE ARRAY 'QUES_ARRAY.'
  IT ALSO DETECTS ERRORS AND OUTPUTS ERROR MESSAGES ASSOCIATED WITH THIS
  PROCESS.  THE SUBROUTINES 'ERROR_CHECK,' 'DATA_PRINT,' AND 'PROC_CALL'
  ARE CALLED FROM THIS SUBROUTINE.  */

BREAK:  PROC;
    DCL
        (IP1,IP2,CT1) FIXED BIN (15,0) INIT (0);
    I = 1;
    J = 1;
    IF NUMARR (5) = 0 THEN DO;
        CALL ERROR_CHECK (IP2);
        IF IP2 > 0 THEN GO TO ENDBK;
        END;
    ALLOCATE QUES_ARRAY (QUES_SIZE (QNUM2,1), QUES_SIZE (QNUM2,2));
    QUES_ARRAY = '';
BR1:  IP1 = INDEX (QUES, DELIM (2));
    IP2 = INDEX (QUES, DELIM (QUES_SIZE (QNUM2, 2) + 1));
    IF IP2 > IP1  |  (IP1 > 1 & IP2 = 0) THEN DO;
        IP2 = IP1;
        CT1 = CT1 + 1;
        END;
    IF I > QUES_SIZE (QNUM2, 1) THEN DO;
        IF NUMARR (5) = 1 THEN GO TO BR2;
        IPO = QUES_SIZE (QNUM2,1);
        IP2 = QNUM2 + (QNUM2 - 1);
        PUT SKIP (4) EDIT ('ERROR:  IN QUESTION #'  || QNUM1
        || ' THERE IS RESERVED SPACE FOR ONLY ', IPO,
        ' SUBRESPONSES.  THIS NUMBER HAS HERE BEEN EXCEEDED.  INCREASE
THE APPROPRIATE VALUE IN ''NUMBER OF SUBRESPONSES'' INSTRUCTION CARD #
29')  (A,F(2),A);
        GO TO ENDBK;
        END;
    QUES_ARRAY (I,J) = LESBLKS (SUBSTR (QUES, 1, IP2 - 1));
    IPO = LENGTH (QUES_ARRAY (I,J));
    IF IPO > NUMARR (3) & NUMARR (5) = 0 THEN DO;
        PUT SKIP (4) EDIT ('ERROR: QUESTION #'   || QNUM1
        || ' CONTAINS ONE OR MORE STRINGS LONGER THAN THE LENGTH SPECIF
IED.  INCREASE SIZE OF ''RESPONSE LENGTH'' IN INSTRUCTION CARD # 3')
        (A);
        GO TO ENDBK;
        END;
BR2:  QUES = SUBSTR (QUES, IP2 + 1);
    IF QUES = '' THEN GO TO BR3;
    I = I + 1;
    IF CT1 = 1 THEN DO;
        I = 1;
        J = J + 1;
        CT1 = 0;
        END;
     GO TO BR1;
BR3:  IF NUMAR2 (4) = 0 THEN CALL DATA_PRINT;
     CALL PROC_CALL;
ENDBK:  END BREAK;



/* 'ERROR_CHECK' IS A SUBROUTINE THAT CHECKS EACH QUESTION STRING TO
  DETERMINE THAT IT IS CORRECT AS FAR AS DELIMITER USE IS CONCERNED, AND
  OUTPUTS THE APPROPRIATE ERROR AND CORRECTION MESSAGES.  */

ERROR_CHECK:  PROC (IP3);
    DCL
        (IF, IP3) FIXED BIN (15,0),
        CK FIXED BIN (15,0) INIT (0),
        COT (QUES_SIZE (QNUM2, 2)) FIXED DEC (3,1);
    HM = 0;
    COT = 0;
    IP3 = 0;
        IF = 2;
ERR1:  DO IL = (IF + 1) TO HBOUND (DELIM, 1);
        CK = INDEX (QUES, DELIM (IL));
        IF CK > 0 & IL ¬= QUES_SIZE (QNUM2, 2) + 1 THEN DO;
        IP3 = 1;
        IF = IL;
    HM = IP3;
    GO TO ERR4;
        END ERR1;
    IL = 0;
    IPO = 0;
        IF QUES_SIZE (QNUM2,2) ¬= 1 THEN DO;
          DO IL = 1 TO QUES_SIZE (QNUM2,2);
          IP1 = IPO;
          IPO = INDEX (SUBSTR (QUES,IPO + 1), DELIM (2))+ IPO;
ERR3:     IP4 = IP1;
          IP1 = INDEX (SUBSTR (QUES,IP1 + 1, (IPO-1) - IP1), DELIM
            (QUES_SIZE (QNUM2,2) + 1))+ IP1;
          IF IP1 ¬= IP4 THEN DO;
               COT (IL) = COT (IL) + 1;
               GO TO ERR3;
               END;
          END;
          IP4 = INDEX (SUBSTR (QUES, IPO + 1), DELIM (2));
          IF IP4 > 0 THEN IP3 = IP3 + 3;
          IF (SUM (COT)) / QUES_SIZE (QNUM2, 2) ¬= COT (1) THEN DO;
            IP3 = IP3 + 5;
            PUT EDIT ('NUMBER OF DELIMITERS PER DATA TYPE:')
            (SKIP(4) , A);
            PUT SKIP (2);
            PUT DATA (COT);
            END;
        END;
ERR4:  IF IP3 = 1  | IP3 = 6 THEN DO;
        PUT SKIP (4) EDIT ('ERROR: DELIMITER ' || DELIM (IF) ||
        ' WAS USED IN QUESTION #' || QNUM1 || '.  REPLACE WITH DELIMITE
R '  || DELIM (QUES_SIZE (QNUM2, 2) + 1)) (A);
           IP3 = 0;
          GO TO ERR1;                                                        731
        END;                                                                 732
    IF IP3 = 5 THEN PUT SKIP (4) EDIT ('ERROR: IN QUESTION #'  ||            733
    QNUM1  || ' ONE OR MORE DELIMITERS OF TYPE '  ||                         734
    DELIM (QUES_SIZE (QNUM2, 2) + 1)  ||  ' ARE MISSING') (A);               735
    IF IP3 = 9  | IP3 = 8 | IP3 = 4 | IP3 = 3 THEN                           736
    PUT SKIP (4) EDIT ('ERROR:  TWO QUESTIONS MERGED: QUESTION #'  ||        737
    QNUM1  || ' AND THE FOLLOWING QUESTION.  DELIMITER '  ||                 738
    'CORRECTED STRING:   '  ||  DELIM (1)  || SUBSTR (QUES,IPO+1),           739
    'CORRECTED STRING:   '  ||  DELIM (1)  || QSTG1,                         740
    'IF CHARACTERS IMMEDIATELY TO RIGHT OF ' || DELIM (1) ||                 741
    ' IN THE CORRECTED STRING DO NOT SIGNIFY A QUESTION NUMBER THEN  ER      742
ROR IS ONE OF TOO MANY RESPONSE TYPES IN QUESTION #'  ||                     743
    QNUM1  || ' FOR NUMBER OF TYPES ALLOWED FOR IN ''RESPONSE TYPES PER      744
QUESTION '' INSTRUCTION CARD # 30') (3 (A, SKIP (2)));                       745
    IF IP3 = 0 & HM > 0 THEN IP3 = 1;                                        746
    END ERROR_CHECK;                                                         747


/* 'DATA_PRINT' IS A SUBROUTINE THAT OUTPUTS EACH RESPONDENT'S DATA,
  CORRECTLY BROKEN DOWN AND LABELED.  */

DATA_PRINT:  PROC;                                                           748
    PUT SKIP (4) EDIT ('QUESTION #'  ||  QNUM1  || ':') (A);                 749
    DO I = 1 TO QUES_SIZE (QNUM2, 1);                                        750
        PUT SKIP (2);                                                        751
        PUT DATA ((QUES_ARRAY (I,J) DO J = 1 TO QUES_SIZE (QNUM2,2)));       752
    END DATA_PRINT;                                                          753


/* THE FOLLOWING CARDS (754 - 797) DEFINE THE VALUES VARIOUS VARIABLES
  WILL HAVE DURING THE PROGRAM RUN, AS WELL AS DEFINING AMOUNTS OF STORAGE
  SPACE NECESSARY FOR THESE VARIABLES.  */

    DCL                                                                      754
        QUES VAR CHAR (758) INIT (''),                                       755
        (ID,TD) VAR CHAR (NUMARR (27)) INIT (''),                            756

/* NOTE: TO CHANGE SIZE OF THE CONTINUATION NUMBER STRING (NOW SET AT 2),
  CHANGE THE DECLARATION IN CARD 957 AND THE COMMAND IN CARD 942.  IF THE
  SUBROUTINE 'SUBSECTION' IS TO BE ACTIVE THEN THE VALUE OF
  'CONT_NUM' MUST BE CHANGED IN THE FOLLOWING CARDS:
  944, 945, 960, 961, 968.  */

        CONT_NUM CHAR (2) INIT ('  '),                                       757
        QNUM1 VAR CHAR (3) INIT (''),                                        758
        (N5,IPO,QNUM2) FIXED BIN (15,0) INIT (0),                            759
        (I241, OP) FIXED BIN (1) INIT (0),                                   760
        RK FIXED DEC (3,0),                                                  761
        T CHAR (1),                                                          762
        LT CHAR (1) INIT (''''),                                             763
        LOT CHAR (3),                                                        764
        (IN2, IN3) FIXED BIN (15,0) INIT (0),                                765
        DELIM (NUMARR(1)) VAR CHAR (3),                                      766
        QUES_SIZE (NUMARR(2), 2) FIXED DEC (3,0),                            767
        QUES_ARRAY (1,1) VAR CHAR (NUMARR(3)) CONTROLLED,                    768
        SEL_IN (NUMARR(7)) VAR CHAR (NUMARR(27)),                            769
```

```
          SEL_OUT (NUMARR(8)) VAR CHAR (NUMARR(27)),                        770
          SM (NUMARR (12)) FIXED BIN (15,0),                                771
          Q (NUMARR (12)) FIXED BIN (15,0),                                 772
          THES (NUMARR(16),NUMARR(17),NUMARR(15)) VAR CHAR (NUMARR(13)),    773
          THES_CT (NUMARR(16),NUMARR(17),NUMARR(12)) FIXED BIN (15,0),      774
          THES_FILL (NUMARR(17)) VAR CHAR (NUMARR(13)),                     775
          TH_CAT (NUMARR(16),1,1) VAR CHAR (2),                             776
          THES_CAT (NUMARR(16),1,1) FIXED BIN (15,0),                       777
          NOTFIND (2,NUMARR(12)) VAR CHAR (HM),                             778
          THES_QUES_COL (NUMARR(12), 3) FIXED DEC (3,0),                    779
          ITEM_QUES_COL (NUMARR(9), 2) FIXED DEC (3,0),                     780
          ITEM (NUMARR(11), 1, NUMARR(9)) VAR CHAR (NUMARR(10)),            781
          ITEM_DIS (NUMARR(11), 1, NUMARR(9)) FIXED BIN (15,0),             782
          LST (NUMARR(11), 2, NUMARR(9)) VAR CHAR (NUMARR(10)),             783
          (F,RES_NUM) (NUMARR (18)) FIXED BIN (15,0),                       784
          LIST (NUMARR(20), NUMARR(18)) VAR CHAR (LK),                      785
          NAME (NUMARR(18)) VAR CHAR (NUMARR(19)),                          786
          NOMIN (NUMARR(21), NUMARR(18)) VAR CHAR (NUMARR(19)),             787
          TSTID (NUMARR(18),2) VAR CHAR (NUMARR(27)),                       788
          NAME_QUES_COL (NUMARR(18),2) FIXED DEC (3,0),                     789
          NOM_QUES_COL (NUMARR(18),3) FIXED DEC (3,0),                      790
          FACT_ARR (NUMARR(26),1) FIXED DEC (3,0),                          791
          (IQ,MAX,MAX_COL) (NUMARR (23)) FIXED BIN (15,0),                  792
          FTD (NUMARR (23)) VAR CHAR (NUMARR (27)),                         793
          RUN_ACT (NUMARR (23),1) FIXED DEC (3,0),                          794
          FACT_AR (NUMARR (22), NUMARR (23)) VAR CHAR (LS),                 795
          (P,NUBRES) (NUMARR(9)) FIXED BIN (15,0);                          796
      ITEM_QUES_COL = 0;                                                    797
      THES_QUES_COL = 0;                                                    798
      NAME_QUES_COL = 0;                                                    799
      NOM_QUES_COL = 0;                                                     800
      FACT_ARR = 0;                                                         801
          RUN_ACT = 0;                                                      802
      GO TO PGM3;                                                           803
  PGM1: TSTG = '';                                                          804
  PGM2:  READ FILE (IN) INTO (CSTG);                                        805
  PGM3:  TSTG = TSTG || CSTG;                                               806
      T = '.';                                                              807
      CALL GET (IN3);                                                       808
      IF IN3 = 0 THEN GO TO PGM2;                                           809
      PUT EDIT (TSTG) (SKIP(2),A);                                          810
      IF NUM1 ¬= 28 THEN DO;                                                811
      NUM = LESBLKS (SUBSTR (TSTG, 1, (INDEX (TSTG, ')') - 1)));            812

  /* CARDS 813 - 836 ARE CONCERNED WITH DEFINING THE ELEMENTS OF THE
    THESAURI UTILIZED.  */

      IF NUM = 'TH' THEN DO;                                                813
          NUM = LESBLKS (SUBSTR (TSTG, INDEX (TSTG, '(') + 1,               814
          (INDEX (TSTG, ')') - INDEX (TSTG, '(')) - 1));                    815
          HMT = NUM;                                                        816
          GO TO PGM1;                                                       817
          END;                                                              818
      IF NUM = 'CA' THEN DO;                                                819
          NUM = LESBLKS (SUBSTR (TSTG, INDEX (TSTG, '(') + 1,               820
          (INDEX (TSTG, ')') - INDEX (TSTG, '(')) - 1));                    821
          NUM1 = NUM;                                                       822
          NU3 = NUM;                                                        823
          T = '=';                                                          824
          CALL GET (IN2);                                                   825
          THES_FILL = '';                                                  826
          CALL GUT1 (THES_FILL);                                           827
          DO I = 1 TO HBOUND (THES_FILL,1);                                828
              NUM3 = I;                                                    829
              VK = NUM3;                                                   830
              NU4 = SUBSTR (VK,5);                                         831
              THES (NUM1,I,HMT) = NU3 || NU4 || THES_FILL (I);             832
              IF THES_FILL (I) = '' THEN THES (NUM1,I,HMT) = '';           833
              END;                                                         834
          GO TO PGM1;                                                      835
          END;                                                             836
      NUM1 = NUM;                                                          837
      IF NUM1 = 46 THEN GO TO PGM4;                                        838
      END;                                                                 839

  /* CARDS 840 - 917 ARE CONCERNED WITH DEFINING THE VALUES OF ELEMENTS OF
    VARIOUS ARRAYS NECESSARY FOR UTILIZATION OF THE PROGRAM.  */

      T = '=';                                                             840
      CALL GET (IN2);                                                      841
      JOT = 1;                                                             842
      IF NUM1 = 28 THEN DO;                                                843
          CALL GUT1 (DELIM);                                               844
          SEL_IN = DELIM (1);                                              845
          SEL_OUT = DELIM (1);                                             846
          NUM1 = 0;                                                        847
          GO TO PGM1;                                                      848
          END;                                                             849
      IF NUM1 = 29 THEN DO;                                                850
          CALL GUT2 (QUES_SIZE,JOT);                                       851
          GO TO PGM1;                                                      852
          END;                                                             853
      IF NUM1 = 31 THEN DO;                                                854
          CALL GUT2 (ITEM_QUES_COL,JOT);                                   855
          GO TO PGM1;                                                      856
          END;                                                             857
      IF NUM1 = 33 THEN DO;                                                858
          CALL GUT2 (THES_QUES_COL,JOT);                                   859
          GO TO PGM1;                                                      860
          END;                                                             861
      IF NUM1 = 36 THEN DO;                                                862
          CALL GJT2 (NAME_QUES_COL,JOT);                                   863
          GO TO PGM1;                                                      864
          END;                                                             865
      IF NUM1 = 38 THEN DO;                                                866
          CALL GUT2 (NOM_QUES_COL,JOT);                                    867
          GO TO PGM1;                                                      868
          END;                                                             869
      IF NUM1 = 41 THEN DO;                                                870
          CALL GUT2 (FACT_ARR,JOT);                                        871
          GO TO PGM1;                                                      872
          END;                                                             873
      IF NUM1 = 42 THEN DO;                                                874
          CALL GJT2 (RUN_ACT, JOT);                                        875
          GO TO PGM1;                                                      876
          END;                                                             877
      JOT = 2;                                                             878
      IF NUM1 = 30 THEN DO;                                                879
          CALL GUT2 (QUES_SIZE,JOT);                                       880
          GO TO PGM1;                                                      881
```

```
        END;                                                                882
    IF NUM1 = 32 THEN DO;                                                   883
        CALL GUT2 (ITEM_QUES_COL,JOT);                                      884
        GO TO PGM1;                                                         885
        END;                                                                886
    IF NUM1 = 34 THEN DO;                                                   887
        CALL GUT2 (THES_QUES_COL, JOT);                                     888
        GO TO PGM1;                                                         889
        END;                                                                890
    IF NUM1 = 37 THEN DO;                                                   891
        CALL GUT2 (NAME_QUES_COL,JOT);                                      892
        GO TO PGM1;                                                         893
        END;                                                                894
    IF NUM1 = 39 THEN DO;                                                   895
        CALL GUT2 (NOM_QUES_COL,JOT);                                       896
        GO TO PGM1;                                                         897
        END;                                                                898
    JOT = 3;                                                                899
    IF NUM1 = 35 THEN DO;                                                   900
        CALL GUT2 (THES_QUES_COL, JOT);                                     901
        GO TO PGM1;                                                         902
        END;                                                                903
    IF NUM1 = 40 THEN DO;                                                   904
        CALL GUT2 (NOM_QUES_COL,JOT);                                       905
        GO TO PGM1;                                                         906
        END;                                                                907
    IF NUM1 = 43 THEN DO;                                                   908
        CALL GUT1 (SEL_OUT);                                                909
        GO TO PGM1;                                                         910
        END;                                                                911
    IF NUM1 = 44 THEN DO;                                                   912
        CALL GUT1 (SEL_IN);                                                 913
        GO TO PGM1;                                                         914
        END;                                                                915
    IF NUM1 = 45 THEN GO TO PGM1;                                           916

/* CARDS 917 - 940 OUTPUT INFORMATION AS TO WHICH SUBROUTINE OPTIONS ARE
  ACTIVE ON THE PROGRAM RUN. */

PGM4:  PUT PAGE;                                                            917
    PUT LIST ('OPTIONS UTILIZED THIS RUN:');                                918
    PUT SKIP (2);                                                           919
    IF NUMARR (6) = 0 THEN PUT SKIP (4) LIST                                920
    ('SUBSECTION OPTION IS ACTIVE');                                        921
    IF SEL_IN (1) ¬= DELIM (1) THEN PUT SKIP (4) LIST                       922
    ('SELECT-IN OPTION IS ACTIVE');                                         923
    IF SEL_OUT (1) ¬= DELIM (1) THEN PUT SKIP (4) LIST                      924
    ('SELECT-OUT OPTION IS ACTIVE');                                        925
    IF ITEM_QUES_COL (1,1) ¬= 0 THEN PUT SKIP (4) LIST                      926
    ('ITEM_CT OPTION IS ACTIVE');                                           927
    IF THES_QUES_COL (1,1) ¬= 0  THEN PUT SKIP (4) LIST                     928
    ('CLASSIFY OPTION IS ACTIVE');                                          929
    IF NAME_QUES_COL (1,1) ¬= 0 THEN PUT SKIP (4) LIST                      930
    ('SOCIO OPTION IS ACTIVE');                                             931
    IF FACT_ARR (1,1) ¬= 0 THEN PUT SKIP (4) LIST                           932
    ('FACTOR ANALYSIS OPTION IS ACTIVE');                                   933
    IF RUN_ACT (1,1) ¬= 0 THEN PUT SKIP (4) LIST                            934
    ('SPSS OPTION IS ACTIVE');                                              935
    IF NUMARR (4) = 0 THEN PUT SKIP (4) LIST ('FOLLOWING PAGES ARE PRIN     936
TOUT OF QUESTION RESPONSES');                                               937
```

```
    IF NUMARR (5) = 0 THEN PUT SKIP (4) LIST ('FOLLOWING PAGES ARE ERRO  938
R CHECK ON THE DATA');                                                   939
    CALL SUB (READ);                                                     940

/* CARD 941 INSTRUCTS THE READING OF A DATA CARD.  */

READ:  READ FILE (INDATA) INTO (CSTG);                                   941
    CONT_NUM = SUBSTR (CSTG, 1, 2);                                      942
    IF HM = -1 & NUM8 = 1 THEN GO TO ENDPROG;                            943
    IF HM = -1 & CONT_NUM ¬= '01' THEN GO TO READ;                       944
    IF CONT_NUM = '  ' THEN GO TO PGM6;                                  945
    HM = CONT_NUM;                                                       946
    IF HM = 1 THEN HMT = 0;                                              947
    IPO = INDEX (CSTG, DELIM(1));                                        948
    TD = LESBLKS (SUBSTR (CSTG, 3, IPO - 3));                            949
    IF SEL_IN (1) = DELIM (1) THEN GO TO PGM5;                           950

/* CARDS 951 - 959 ARE CONCERNED WITH THE 'SELECT-IN' AND 'SELECT-OUT'
  OPTIONS.  */

    DO I = 1 TO HBOUND (SEL_IN,1);                                       951
        IF TD = 'END' THEN GO TO PGM7;                                   952
        IF TD = SEL_IN (I) THEN GO TO PGM5;                              953
        END;                                                             954
        GO TO READ;                                                      955
PGM5: IF SEL_OUT (1) = DELIM (1) THEN GO TO PGM7;                        956
    DO I = 1 TO HBOUND (SEL_OUT,1);                                      957
        IF TD = SEL_OUT (I) THEN GO TO READ;                             958
        END;                                                             959

/* CARDS 960 - 982 ARE CONCERNED WITH THE 'SUBSECTION' OPTION.  */

PGM6: IF CONT_NUM = '  ' THEN TD = '';                                   960
PGM7: IF CONT_NUM = '01'  | CONT_NUM = '  ' THEN DO;                     961
        IF QNUM1 ¬= '' & NUMARR (4) + NUMARR (5) ¬= 2 THEN DO;           962
            IF ID ¬= '' THEN DO;                                         963
                PUT SKIP (4) DATA (ID);                                  964
                PUT PAGE;                                                965
                END;                                                     966
                END;                                                     967
            IF CONT_NUM = '  ' THEN DO;                                  968
                IF NUMARR (6) = 0 THEN DO;                               969
                  I241 = 1;                                              970
                  GO TO ANALYZE;                                         971
                  END;                                                   972
                IF NUMARR (6) = 1 THEN DO;                               973
                  ID = '';                                               974
                  GO TO READ;                                            975
                  END;                                                   976
                TD = '';                                                 977
            END;                                                         978
        TSTG = '';                                                       979
        IF TD = 'END' THEN GO TO ANALYZE;                                980
        ID = TD;                                                         981
        END;                                                             982

/* CARDS 983 - 995  DEAL WITH DATA CARDS OUT OF CORRECT SEQUENCE. */

    ELSE IF ID ¬= TD THEN DO;                                            983
        PUT LIST ('ERROR: CARDS OUT OF ORDER BY ID NUMBER WHERE: ');     984
        PUT DATA (ID, TD);                                                985
        HM = -1;                                                          986
        GO TO READ;                                                       987
        END;                                                              988
    IF HMT > HM THEN DO;                                                  989
        PUT EDIT ('ERROR: CARDS OUT OF ORDER BY CONTINUATION NUMBER WHE   990
RE ID = ' || ID || ' AND THE NUMBERS ARE: ', HMT, ',' , HM)               991
        (SKIP(4),2(A,F(4)));                                              992
        HM = -1;                                                          993
        GO TO READ;                                                       994
        END;                                                              995
    HMT = HM;                                                             996

/* CARDS 997 - 1012  ISOLATE A QUESTION NUMBER AND ITS QUESTION STRING
  FOR PROCESSING IN 'BREAK' AND MANIPULATION IN THE OTHER
  SUBROUTINES.  */

    TSTG = TSTG || SUBSTR (CSTG, IPO + 1);                                997
PGM8: QUES = '';                                                          998
    IPO = INDEX (TSTG, DELIM (1));                                        999
    IF IPO = 0 THEN GO TO READ;                                          1000
    QUES = SUBSTR (TSTG, 1, IPO - 1) || DELIM (2);                       1001
    TSTG = SUBSTR (TSTG, IPO + 1);                                       1002
    IPO = INDEX (QUES, DELIM (2));                                       1003
    QNUM1 = LESBLKS (SUBSTR (QUES, 1, IPO - 1));                         1004
    ON CONVERSION GO TO PGM9;                                            1005
    QNUM2 = QNUM1;                                                       1006
    REVERT CONVERSION;                                                   1007
    QUES = SUBSTR (QUES, IPO + 1);                                       1008
    GO TO PGM10;                                                         1009
PGM9: PUT EDIT ('CONVERSION ERROR ON QUESTION NUMBER.') (SKIP(4),A);     1010
    GO TO PGM8;                                                          1011
PGM10: IF QNUM2 < (HBOUND (QUES_SIZE,1) + 1) THEN CALL BREAK;            1012
    FREE QUES_ARRAY;                                                     1013
    GO TO PGM8;                                                          1014

/* CARDS 1015 - 1024 CONTROL TOTALLING AND OUTPUT ACTIONS AT THE END OF
  THE PROCESSING OF EACH DATA SET SPECIFIED AS COMPLETE. */

ANALYZE:  PUT PAGE;                                                      1015
    OP = 1;                                                              1016
    J = HBOUND (QUES_SIZE,1);                                            1017
    QNUM2 = 0;                                                           1018
PGM11: QNUM2 = QNUM2 + 1;                                                1019
    CALL PROC_CALL;                                                      1020
    IF QNUM2 > J THEN GO TO PGM12;                                       1021
    GO TO PGM11;                                                         1022
PGM12:  RK = -1;                                                         1023
    IF I241 = 1 THEN CALL SUB (READ);                                    1024
ENDPROG:  END PRGM;                                                      1025



    DCL                                                                  1026
        LESBLKS RETURNS (VAR CHAR (320)),                                1027
        (NU3,NU4) CHAR (2),                                              1028
        NU5 CHAR (3),                                                    1029
        QI VAR CHAR (3),                                                 1030
        VK CHAR (6),                                                     1031
        (IN,INDATA) RECORD INPUT,                                        1032
        OUT OUTPUT,                                                       1033
        (LS,LK) FIXED BIN (15,0) INIT (1),                                1034
        (I,J) FIXED DEC (2,0),                                            1035
        (PERCENT1,PERCENT2) FIXED DEC (4,1),                              1036
        CSTG CHAR (80),                                                   1037
        (NUM,NJM2) VAR CHAR (2),                                          1038
        (NUM1,NUM3) FIXED DEC (3,0),                                      1039
        TSTG VAR CHAR (800),                                              1040

/* CARD 1041 DECLARES THE ARRAY 'NUMARR.' THIS ARRAY IS FILLED WITH
  VALUES READ FROM THE INSTRUCTION CARDS NUMBERED 1 - 27.  THEN THESE
  VALUES ARE ENTERED IN CARDS 754 - 796 AS SIZE DECLARATIONS OF THOSE
  VARIABLES AND ARRAYS.  */

        NUMARR (27) FIXED DEC (3,0);                                      1041
    NUMARR = 1;                                                           1042
    NUM8 = 0;                                                             1043
    TSTG = '';                                                            1044
    PUT LIST ('INSTRUCTION CARDS UTILIZED THIS RUN:');                    1045
    PUT SKIP (2);                                                         1046

/* CARDS 1047 - 1073 DETERMINE THE VALUES OF THE CELLS OF 'NUMARR'
  AS READ FROM THE INSTRUCTION CARDS.  */

INST1:  READ FILE (IN) INTO (CSTG);                                       1047
    NUM = LESBLKS (SUBSTR (CSTG, 1, (INDEX (CSTG, ')') - 1)));            1048
    NUM1 = NUM;                                                           1049
    IF NUM1 > 8  & NUM1 < 28 THEN NUM8 = 1;                               1050
    IF NUM = 28 THEN DO;                                                  1051
        TSTG = '';                                                        1052
        HM = NUMARR (13) * NUMARR (14);                                   1053
    IF NUMARR(27) > NUMARR(13) THEN HM = NUMARR(27) * NUMARR (14);        1054
        CALL PRGM;                                                        1055
        GO TO END_IT;                                                     1056
        END;                                                              1057
    PUT EDIT (CSTG) (SKIP(2),A);                                          1058
    IF NUM1 = 4 | NUM1 = 5 | NUM1 = 6 THEN DO;                            1059
        NUMARR (NUM1) = 0;                                                1060
        GO TO INST1;                                                      1061
        END;                                                              1062
    NUM2 = LESBLKS (SUBSTR (CSTG, (INDEX (CSTG, '=') + 1),                1063
      (INDEX (CSTG, '.') - 1)));                                          1064
    ON CONVERSION PUT EDIT ('MOST LIKELY ERROR IS PERIOD MISSING AT END   1065
 OF INSTRUCTION CARD #', NUM) (SKIP(2),A,A);                              1066
    NUM3 = NUM2;                                                          1067
    REVERT CONVERSION;                                                    1068
    IF NUM1 = 13 THEN NUM3 = NUM3 + 4;                                    1069
    IF NUM1 = 19 THEN LK = (NUM3 * 2) + 4;                                1070
    IF NUM1 = 24 THEN LS = NUM3 * 3;                                      1071
    NUMARR (NUM1) = NUM3;                                                 1072
    GO TO INST1;                                                          1073
END_IT:  PUT PAGE;                                                        1074
    END STGPROC;                                                          1075
```

APPENDIX TWO

RESPONSE TYPE AND SUBRESPONSE: SUGGESTED TABULATION SHEET

Response Number (From 1 up by increments of 1)	Number of Response Types	Number of Subresponses	Number of Character Symbols

Maximum Number of Symbols

APPENDIX THREE

THE INSTRUCTION CARDS

Following are the general forms of all the STGPROC instruction cards.

1) NUMBER OF DELIMITERS = n.

2) NUMBER OF QUESTIONS = n.

3) RESPONSE LENGTH = 20.

4) ACTIVATE PRINT RUN.

5) ACTIVATE ERROR RUN.

6) ACTIVATE SUBSECTION RUN.

7) ACTIVATE SELECT-IN RUN, NUMBER OF RESPONDENTS = n.

8) ACTIVATE SELECT-OUT RUN, NUMBER OF RESPONDENTS = n.

9) NUMBER OF ITEM_CT RUNS = n.

10) ITEM_CT LENGTH OF RESPONSE = 1.

11) ESTIMATE OF MAXIMUM NUMBER OF UNIQUE ITEMS IN AN ITEM_CT RUN = e.

12) NUMBER OF CLASSIFY RUNS = n.

13) CLASSIFY LENGTH OF RESPONSE = 1.

14) ESTIMATE OF NUMBER OF UNCLASSIFIED RESPONSES = e.

15) NUMBER OF THESAURI UTILIZED = n.

16) LARGEST NUMBER OF CATEGORIES IN A THESAURUS = n.

17) LARGEST NUMBER OF ITEMS IN A CATEGORY = n.

18) NUMBER OF SOCIO RUNS = n.

19) SOCIO LENGTH OF RESPONSE = 1.

20) SOCIO LARGEST TOTAL NUMBER OF NOMINEES IN A RUN = n.

21) MAXIMUM NUMBER OF NOMINATIONS PER NOMINATOR IN A SOCIO RUN = n.

22) SPSS RESPONDENT ESTIMATE = e.

23) SPSS NUMBER OF RUNS = n.

24) MAXIMUM NUMBER OF SUBRESPONSES IN AN SPSS RUN = n.

25) FACTOR ANALYSIS CONTINUATION NUMBER = c.

26) FACTOR ANALYSIS NUMBER OF RUNS = n.

27) MAXIMUM LENGTH OF ID STRING = l.

28) IDENTITY OF DELIMITERS = $d_1, d_2, \quad d_n$.

29) SUBRESPONSES PER QUESTION = $s_1, s_2, \quad s_n$.

30) RESPONSE TYPES PER QUESTION = $t_1, t_2, \quad t_n$.

31) ITEM_CT QUESTION NUMBERS PER RUN = $q_1, q_2, \quad q_n$.

32) ITEM_CT RESPONSE TYPES PER RUN = $r_1, r_2, \quad r_n$.

33) CLASSIFY QUESTION NUMBERS PER RUN = $q_1, q_2, \quad q_n$.

34) CLASSIFY RESPONSE TYPES PER RUN = $r_1, r_2, \quad r_n$.

35) CLASSIFY THESAURUS USED PER RUN = $t_1, t_2, \quad t_n$.

36) NOMINATOR QUESTION NUMBERS PER SOCIO RUN = $q_1, q_2, \quad q_n$.

37) NOMINATOR RESPONSE TYPES PER SOCIO RUN = $r_1, r_2, \quad r_n$.

38) NOMINEE QUESTION NUMBERS PER SOCIO RUN = $q_1, q_2, \quad q_n$.

39) NOMINEE RESPONSE TYPES PER SOCIO RUN = $r_1, r_2, \quad r_n$.

40) NUMBER OF NOMINEES PER SOCIO RUN = $n_1, n_2, \quad n_n$.

41) CLASSIFY RUNS ACTED UPON BY FACTOR ANALYSIS = $c_1, c_2, \quad c_n$.

42) ITEM_CT RUNS ACTED UPON BY SPSS = $i_1, i_2, \quad i_n$.

43) SELECT-OUT RESPONDENT ID'S = $id_1, id_2, \quad id_n$.

44) SELECT-IN RESPONDENT ID'S = $id_1, id_2, \quad id_n$.

45) THESAURI DECLARATIONS.

THE3AURUS (n).

CATEGORY (n-1) = $c_1, c_2,$ c_n.

CATEGORY (n) = $c_1, c_2,$ c_n.

46) END OF INSTRUCTION CARDS.

APPENDIX FOUR

JOB CONTROL CARDS

The following is a sample of the job control language necessary for the utilization of STGPROC. This job control language is only an example, effective at the University of Oregon Computation Center at the time of the development of STGPROC, and is set up to read the program and all data from IBM cards. There are many alternatives to this (such as reading from tape) and it is suggested that the researcher consult his resident computation center for the job control most effective for his STGPROC runs.

```
//job.name  JOB  job.number,user.name,MSGLEVEL=1

//JOBLIB DD DSNAME=SYS1.PL1,DISP=(OLD,PASS),UNIT=2314,VOLUME=SER=WRITER

// EXEC PROC=PL1LFCLG,PARM.PL1L='SIZE=999999,NA,X,NST,LC=61'

//PL1L.SYSLIN DD SPACE=(TRK,(10,4))

//PL1L.SYSUT1 DD SPACE=(TRK,(30,2))

//PL1L.SYSUT3 DD SPACE=(TRK,(20,2))

//PL1L.SYSIN DD *,CARDS=1300,BLOCK=5

(place the program here)

/*

//LKED.SYSLIB DD VOLUME=SER=WRITER,UNIT=2314

//GO.OUT DD SYSOUT=B,SPACE=(80,(1500)),DCB=(RECFM=FB,LRECL=80,          C

//              BLKSIZE=800)
```

```
//GO.IN DD *,CARDS=181,BLOCK=21

(put instruction cards here)

//GO.INDATA DD *,CARDS=3000,BLOCK=21

(Put STGPROC respondent data here)

/*
```

APPENDIX FIVE

BASIC STGPROC PROGRAM OUTPUT

The output from the STGPROC program varies greatly, according to what options have been utilized within the program run. Generally speaking, the output takes the following form.

1. The information appearing on the instruction cards utilized for the run is reproduced for convenience in checking what has been spedified for the run.

2. The subroutine options utilized during the program run are listed.

3. The specific output follows, subsection by subsection. Each subsection is plainly marked, and a page is skipped between subsections.

   3a. If individual data are to be output, respondent by respondent, they preceed any tables created by the various subroutines that present the data totals for that subsection (such as ITEM_CT, or SPSS).

   3b. If error messages are to be output, this preceeds any tables created by the various subroutines that present the data totals for that subsection (such as CLASSIFY or FACT_AN).

   3c. The output from the various subroutines totaled for the subsection in question,proceeds--the runs (from one up in increments of one) from each type of subroutine appearing, one after the other, with a blank page between types of subroutine output.

4. All paper output is clearly labelled and identified by means of page headings and subheadings, as shown in the appropriate textual examples.

5. All punched card output decks are clearly labelled and identified on the first card of the deck.

Typed by Rebecca L. Goodrich