*A person-to-computer communication system for application program writing by nonprogrammers is discussed.*

*Called A Program Generator* (APG), *the interface system is built upon the authors' previous developments of a tutorial system that is briefly discussed.*

*Principles and applications of* APG *are presented and illustrated in terms of actual applications.*

# A program generator

**by W. D. Hagamen, D. J. Linden, K. F. Mai, S. M. Newell, and J. C. Weber**

Computers are machines that play a unique role in modern technology in that they tend to amplify man's mental power rather than his physical power. If they are to be considered an "intellectual appendage," why are they so difficult to use? Note that we are not asking why it is difficult to understand how they work, but merely why there is a communication gap between man and machine. This question recalls some words written by one of the authors over thirty years ago, while trying to explain why knowledge of the human brain lagged so far behind the physical sciences: "Something in the Mind's Complexity, does not like to admit its Simplicity."[1]

**man-machine communication**

In this paper we use our medical environment as an example of man-machine communication. Here we find that there are three alternative ways of bridging the communications barrier as we have observed it.

- Teach the physician and medical administrator the language of the machine, i.e., teach them to become programmers. We have not been too successful with this approach.
- Provide them with their personal interpreter or programmer. Not only is this expensive, but it also serves to remove them from direct control over what they want to accomplish.
- Teach the machine to speak the language of the physician, i.e., English.

We have chosen to teach the machine to speak the language of the physician. Techniques for our man-machine dialogues form the main theme of this paper.

## A Tutorial System

An instructional language we developed,[2,3] called A Tutorial System (ATS), was—in many respects—a precursor of A Program Generator (APG) that we present in this paper. ATS consists of two parts: (1) The author interrogation program, which asks questions of the author of a particular computer-mediated tutorial. The answers to these questions are stored as program data. (2) This program data drives the program that supervises the tutorials, and makes each tutorial unique.

The tutorial supervisor (TUTOR) does not accumulate file data, except for a minimal amount of information regarding the nature of the interaction with the student. Instead, it first evaluates the syntax of the student's input to determine whether he is answering a question, making a comment, or asking a question. It then evaluates the meaning of the student's input and responds appropriately. This response is determined by a combination of the factual and logical data that the author provides (program data) and by TUTOR's access to the rules of human discourse.

Another way of saying this is that TUTOR performs the primary function of carrying on a normal conversation. Certain basic rules of human discourse are built into the function of the supervisor, which are then applied to the program data supplied by the author and to the input from the students to create a considerable aura of intelligent verbal behavior.

ATS can, for example, select the most appropriate answer to a student's question, based on its available stored information, even though the question had not been anticipated by the author. It can recognize and deal with such things as ambiguous and conflicting input (questions or answers), and handle negating words. The use of personal pronouns (it, he, they) is acceptable, and meaning is assigned to these words according to the context of the discussion. It also has the ability, to about the extent that people have it, to interpret correctly the intended meaning of misspelled words.

On the basis of experience with ATS and TUTOR, we have incorporated these fairly powerful features of human-machine communication into a noninstructional application. In the new system, a modified version of TUTOR performs the front-end function of user data input, which we designate and later define as ENTER.

The ability to interpret misspelled words is made use of in ENTER when the user is referring to items on a list. This ability is also used in the various output functions when the user has to refer to predefined text. When the program can silently correct the user's occasional errors, it makes his job easier.

For the most part, the other features mentioned, which are essential to a free-format tutorial discussion, are not really needed in medical application programs, the primary function of which is to store and access data. The reason they are not needed relates to the distinction to be drawn between filling out a questionnaire and producing a free-format composition.

## A program generator

The data processing needs of Cornell University Medical College, and an increasing percentage of those of its affiliated hospitals, currently are being served by using APL. To facilitate both the writing and subsequent maintenance of the large number of application programs involved, a conversational interface has been developed that automatically generates and documents the application programs, and permits their subsequent maintenance or modification in an equally conversational mode. These operations are performed using A Program Generator (APG) which is discussed in this paper.

APG has been designed to enable people with analytical habits of thought, who know what they want to do, to write application programs by answering a series of questions in English, with adequate default options at every point. Thus the user does not have to be a computer programmer.

**use of the program generator**

Although the stereotyped conversation imposed by APG might be considered to be a programming language in its own right, it differs from other languages in the very important respect that the program generator interrogates the user. Thus the initiative is assumed by the program, not by the user. Neither does he have to compose a series of declarative statements that often require complicated rules of syntax, nor does he assume responsibility for the resulting product's functional cohesion.

Despite such considerations, the average physician or medical administrator is no more inclined to write his own computer applications than he is to type his own letters. Even though the programmer uses a typewriter-like terminal, the analogy goes still deeper. The person who wants a letter written usually knows what he wants to say. He also has a working knowledge of the capabilities of a typewriter, although his secretary is more skilled in its use. However, just as one dislikes being burdened with details of spelling, punctuation, grammar, and format, so with the program generator one does not want to be concerned with similar details of validity checks, sequence, and format. Nevertheless, the ultimate user is brought closer to the actual process of program design. He proofreads and edits the results, and quickly learns what computers can and cannot do effectively.

Perhaps the most tangible benefit of giving the user these insights into the nature of data processing is the awareness that the greatest costs are incurred with data input. Thus it behooves the user to make as much use of these data as possible. The standard advice we give someone after receiving their initial specifications for a program is "dream a little."

APG does not produce APL code. Rather, the generator asks the programmer a series of questions, the answers to which are stored as data. These data drive the basic APL functions common to all programs, and thereby make each program unique. To avoid confusing these data with those that are subsequently entered via the application program itself, we shall refer to the data that drive the programs as *program data.* Information entered through the application programs is called *file data.*

**data-driven programs**

We use the term *data-driven programs* to describe the distinction between special-purpose programs that are individually coded for each application and general-purpose programs that are modified by a series of stored instructions for each application. This distinction should be familiar to any computer professional. The analogy is similar to the distinction between a hard-wired program and the stored computer program, which provides the basis for modern data processing. Just as computers are capable of performing only a limited number of simple operations, and it is the sequence of these operations that is controlled by a stored set of instructions or program data, so application programs perform a limited number of functions that are controlled by a series of stored instructions.

**text processing**

Quite apart from APG, we have our own text editing and formatting package[4] which receives wide use within the Cornell University-N.Y. Hospital Medical Center for activities that range from writing letters to printing books. Many of the features of this program have been incorporated into APG. Two of these seem worthy of mention.

On the input side, there is what we call a "user-defined shorthand," which may be used whenever character data are asked for. The user simply types in an asterisk followed by any abbreviation that seems intuitively appropriate. The person is asked, on completion of the entry, to specify the meaning of any newly entered abbreviations. Thus *ENT could be used to stand for "otorhinolaryngology." The entry is not only printed out as "otorhinolaryngology," but it is also stored that way in the file. This is important so that one user is not restricted to another person's convention. Such abbreviations are not limited to single words, but they can represent text of any length. Thus *NYH might represent "The New York Hospital." This user-defined shorthand conserves considerable input time in such situations

as the biographic section of a personnel file, which includes many repetitions of long names of institutions attended, medical specialties, and addresses. It also tends to reduce the incidence of typing errors.

On the output side we find significant use of a hyphenation algorithm that automatically hyphenates words according to the rules of common English usage, for purposes of right justification of formatted text. Hyphenation is particularly useful where the column width is narrow.

The ease of using APG derives not so much from its natural language communication as from the interrogative nature of the dialogue. The user, rather than being required to structure and sequence a series of statements, is asked explicit questions that have adequate default options at every point. Clearly defined options appeal to most users.

The programming field abounds with languages that require a series of procedural statements in pseudo-English, such as COBOL. To encourage such verbosity would be to abuse the power and brevity of APL. The pressure is still on the user to learn the rules of syntax of the language, and to compose his statements to produce a functional whole. The goal we set for ourselves, and feel we have achieved, is quite different, i.e., to guide the user every step of the way.

**efficiency of data-driven programs**

We have little doubt that our use of a general-purpose program for writing application programs can reduce programming and maintenance costs and provide a satisfactory level of quality. A natural question, however, might be, to what extent (if any) must we lose efficiency of data input, data storage, and processing time because of the generality of the data-driven functions? Optimal conditions for one application may not be optimal for all applications.

APG is general purpose only to the extent that it contains the basic functions necessary for any application. The functions have been fine tuned to the best of our ability. However, the program data that drive these functions make each program unique and special purpose in every sense of those words.

Selective branching and a multiplicity of validity checks increase the efficiency of data input. The facility for variable-length records and for combining consecutive data fields into the minimum number of integers offers the maximum degree of data compaction. The ability of the user to override such features by storing each data element separately and even redundantly (in attribute blocks), permits him to make the optimal tradeoffs between

storage requirements on the one hand and access and CPU times on the other.

The response times in the particular APL environment in which we work are well within the range required for terminal-based applications. This includes the interrogation of large files. Volume reports are produced on a high-speed line printer.

## System environment and experience

Since this experimental computing system involves one of the most extensive uses of APL within a medical environment, it seems appropriate to include a brief description of the computing facilities and the scope of the applications. The computer is an IBM System/370 Model 145 with APL files and microcoded assistance. Thus there is no problem with workspace size. Not only are the program data and file data stored in files, but the myriad of APL functions that comprise the program generator are also stored. When any of the named top-level functions is executed, it automatically reads into the workspace all the necessary subfunctions and program data and expunges unnecessary functions and data.

We variably use a typewriter terminal, CRT, paper tape (for laboratory instruments), and cards for data input—depending on the application. Also used are a high-speed line printer for volume outputs and the typewriter terminals or CRTs for other types of displays.

***file management***

Most of the program data are stored outside the programmer's workspace, in a file area set aside for this purpose. What remains in the workspace is basically a series of descriptors that define the contents of the program data to the programmer functions.

Since application programs vary greatly in size, we have found it useful to allow the programmer functions to define their own storage areas dynamically, rather than define fixed area for each application ahead of time. In our present implementation, each program is initialized with an external storage area of approximately 4K bytes, and this is added to as needed from a file pool shared by all the APG programs on the system.

Control of access to both program data and file data is provided by reference to the user's APL account number. Program data may be modified only from the sign-on number by which it was originally entered. Furthermore, file data may only be modified or displayed from sign-on numbers for which authorization has been entered from the programmer's user number.

Communication between separate APG programs is automated so that one program may make use of information entered into the system from another, either to modify its behavior (e.g., branching) or to update its own file data. All such uses of multiple files depend on access authorization, which can only be given from the user number that originally establishes each file.

Overall control of the APG system is vested in one account number, which may execute functions to allow entry of a new program into the system and monitor each program's use of storage area for accounting purposes.

We were given three major accounting-type programs written in COBOL. These are planned to be converted to APL. The only reason for mentioning the existence of these programs is that we use conversational APL programs to interrogate the COBOL files. Whether one is talking about APL or COBOL files, he finds that the need for volume reports is significantly reduced by the facility of being able to interrogate the files in a conversational manner to obtain current information whenever it is needed.

**programming times**

The time required to produce the average application program using APG is less than ten percent of that required to write the same program by hand. However, the greatest single benefit is the automatic detailed documentation and subsequent ease of program maintenance, utilizing the various editing functions.

When we first, naively, became involved in providing a programming service, we saw no problem in writing special-purpose programs. Each application offered new challenges, and often new solutions for previous problems. Because of the conciseness of the APL language, our programming times and costs were a fraction of previously estimated costs.

The ten-percent time estimate did not include the time required for documentation. Adequate documentation of APL programs is regarded as time-consuming work and has typically not been done. When one is on top of each program, this causes no problems. However, a year and twenty to forty application programs later—especially when some of the people involved are no longer with us—program maintenance and modification become a very real problem.

## Program generator concepts

APG consists of the following two major classes of APL functions:

- *Programmer functions* that create program data.
- *User functions* that collect, process and output file data.

Table 1 Programmer and user functions

| *Purpose of functions* | *Programmer functions* | *User functions* |
|---|---|---|
| Data input | GENENTER | ENTER |
| | GENBRANCH | |
| | GENFILE | |
| | GENDISPLAY | |
| Editing | GENEDIT | EDIT |
| | | EDITFILE |
| Data output | GENREPORT | REPORT |
| | GENSUMMARY | SUMMARY |
| | GENDISPLAYSUM | |
| | GENSTATS | STATS |
| | | SEARCH |

The programmer and user functions may in turn be broken down into the following three general categories:

- Functions that relate to data input and file organization.
- Functions that edit either program or file data.
- Functions that relate to data output.

Since programmer and user functions often perform parallel tasks, it is appropriate to assign similar names to related pairs of functions.

**programmer and user functions**

Table 1 contains the function names called directly either by the person who creates an application program or by the user of an application program.

GENENTER creates most of the program data related to subsequent file data input, storage, and retrieval. More specifically, this function determines how the questions to be asked in ENTER and EDIT appear to the user; validity checks to be imposed on the data; where the data are to be stored; names to be assigned to units of data; and how the output data are formatted.

GENBRANCH defines conditions under which a question, or series of questions, will be skipped or repeated in ENTER and EDIT.

GENFILE establishes file organization.

GENDISPLAY prints the documentation of the program data as defined by the three functions GENENTER, GENBRANCH, and GENFILE.

GENEDIT is used to make changes in the program data previously defined by GENENTER, GENBRANCH, or GENFILE.

ENTER is executed by the user of an application program to input the data constituting new records.

EDIT is used to change (or add to) records previously entered by the user.

Four user functions produce four different types of data output. Three of these user functions require the prior execution of a parallel programmer function to provide the program data to drive them.

REPORT provides the output of the answers to specified questions in tabular format. Fields to be printed, sequence of sorting, criteria for data selection, and totals are defined by the prior execution of the GENREPORT function.

SUMMARY composes and prints file data in prose form, according to rules of structure defined in GENSUMMARY. GENDISPLAYSUM displays (documents) the program data that drive SUMMARY, and GENEDITSUM permits modification of this program data.

STATS prints cumulative statistics stored in counters. The options for selection, sorting, totals, and formatting are similar to those in GENREPORT. However, because the data are stored in a separate part of the file (counters), a separate programmer function—called GENSTATS—is used to specify these options.

SEARCH permits the user to select from the file all information that satisfies any criteria he can state in terms of answers to specific questions.

### Functions related to data input

**brief and verbose modes**

If a person is unfamiliar with any of the programmer or user functions, explicit instructions are required to indicate his options at every point. However, after one has used a program a sufficient number of times, such repetitious printing of these options consumes unnecessary time. Thus there is a brief and a verbose mode for displaying the text (options) associated with each question in every programmer and user function.

One can switch from brief to verbose mode, or vice versa, simply by typing **x** whenever the keyboard unlocks. The program continues in that mode until an **x** is typed again.

In the Appendices, examples of terminal interaction are described and shown. Outputs from the computer begin at the left margin of the page, and inputs from the terminal are indented six

spaces, so that the reader may easily distinguish machine from user.

The examples show the use of the program generator to write a program called GRANTS AND CONTRACTS. This is a program currently used by Cornell University Medical College for the management of Federal and Foundation research funds.

There are two basic modes of operation of the program generator. In the example shown in Appendix 1, the user has typed the name of the function GENENTER, to which the system responds that this is Question 1. The system asks the user to respond to the message LT:. Since this is too cryptic for him, the user types **x** to signal a more explicit statement of what is expected. The instruction is repeated in its verbose form: *Long text:*. The user answers this, and—still in verbose mode—is asked to respond to *Brief text:*. If the user had wanted the brief and verbose text to be identical, he could have entered a carriage return, which would have caused the long text to be copied as the brief text. If he had wanted the brief text to be omitted and replaced by the question number, which is always displayed in ENTER and EDIT, he could have entered a space before the carriage return. If a user of ENTER or EDIT is in verbose mode, the GRANT ID NO: tells him what he is expected to enter. If he is in brief mode, he sees only ID:.

The first question in any application program has certain special restrictions. It must be a number—preferably a unique number—that is used for the identification of logical records. In Appendix 1, the number is the grant identification number. Otherwise, it could have been a social security number, patient history number, item number, or sequential identification number, depending on the nature of the application and the judgement of the programmer.

Because this was the first question, the facts that it requires numeric input, consists of one entry, and contains no decimals are stated for the user. He is then asked for maximum and minimum values, which serve as validity checks on the file data input via ENTER.

GENENTER proceeds to Question 2, still in verbose mode, and continues in that mode until the user again types **x**. Since the user has progressed past the first question, he has options in addition to entering long text. By typing **c**, he may define a question as a calculation. Such questions do not require any input from the keyboard, and, therefore, need no brief or long text to tell the user what to do. Instead, the user does not have to do anything. The file data are obtained by performing calculations that have been defined by the programmer on any fields from

any questions already answered. The other option is to type **r** for *repeat*, which means that he wants to copy some question that already exists. *Repeat* is used when subsequent editing of such a question would be quicker than specifying the entire question. The user exists from GENENTER by entering a carriage return in lieu of *Long text*, **c**, or **r**.

**data types**

For all questions except Q1, GENENTER asks for the data type. For purposes of the user interfacing with the program, we currently distinguish five types of data: binary, numeric, date, character, and list. Actually, all file data are converted to APL integer data before being written into the file.

*Binary data* are those that may be selected by numeric code from a predefined list of options, and they are binary in the sense that an option is either present or absent, as indicated by the numbers entered. Data such as primary diagnoses, complications, and associated conditions are often treated as binary data. Apparently unrelated items such as sex, marital status, and ethnic group can be handled as a single question, since GENENTER asks whether the data should be divided into mandatory or exclusive sets. A *mandatory set* is one from which at least one choice must be made. An *exclusive set* is one from which only one choice may be made. These restrictions are then imposed as validity checks on the entering data. Such data are binary in the additional sense that they can be represented as a binary string. Such strings are then converted to the minimum number of APL integers necessary to contain the information, i.e., thirty bits are represented as a single base ten integer. (We reserve the thirty-first bit for another purpose.) Appendix 2 shows one use of the binary data type in the GRANTS AND CONTRACTS program.

GENENTER begins execution with the next sequential question number. Appendix 2 starts with Q2. Two formatting symbols are used for defining the long text. The ¢ indicates a carriage return. The s̲ is replaced by a space when printing by the user functions ENTER and EDIT. Actually, s̲ is an s overstruck with an upper case F. The convention of using s̲ is used here for exposition so that it may be visualized without an APL Selectric ® Typing Element.

*List data* are similar to binary data, but differ in the following respects. The user may enter either a name (character data) from a predefined list or the corresponding numeric code. At the programmer's option, a user may add to the number of items on the list. List data have the added advantage that where the number of options is very large, partial display of the list is permitted, rather than requiring that all information be contained in the verbose text. An example of list data is given in Appendix 3.

*Character data* (illustrated in Appendix 4) are APL literal data. Here we ask the maximum number of characters allowed and whether an empty vector is permitted. As with all types of data, character data are converted to APL integers before being stored.

*Dates* are treated as separate data types for formatting purposes. Special functions are used for calculating time differences in days, weeks, months, or years between two dates or with respect to dates available from the system. Again GENENTER asks for the number of entries required and the earliest (minimum) and latest (maximum) values permitted. Appendix 5 shows an example of dates taken from the GRANTS AND CONTRACTS program, in which we have already defined Q5 and Q6.

*Numeric questions* are asked by GENENTER to determine the number of entries (numbers required), maximum and minimum values permitted for each entry, number of decimal places to be stored, and whether (when there is more than one entry) certain of these may be encoded together for purposes of data compaction, i.e., represented as a single integer in a different base numbering system. An example of a numeric question is shown in Appendix 6.

Calculations between fields are performed in ENTER (or EDIT) and therefore have to be specified in GENENTER. In ENTER, calculations replace questions, i.e., they represent questions for which the input is internally, rather than externally, supplied. Interfield calculations are contrasted here with calculations performed on individual fields, e.g., selection and sorting, which are performed in the functions related to data output.

A calculation means any operation that can be performed on any number of fields from any number of questions, as long as these questions have already been answered. Many of the common calculations such as summation, multiplication, percentage, and averaging, as well as certain calculations involving dates, have been anticipated. In these instances, the user simply indicates the desired calculation, and the fields in which questions should constitute the input. He answers questions regarding the field width, decimal positions, heading, and whether the result should be stored in a sorting block. An example of an interfield calculation is shown in Appendix 7.

**print format and name specification calculations**

GENENTER requires the definition of three types of names. One type is mentioned under "list data," i.e., the literal text that comprises the lists. Another is the heading that should be displayed in the output program whenever that field is printed. The third is any text associated with the various options in a binary type question, again for display purposes. In Q2 of Appendix 2,

the heading is GRANT TYPE, the texts associated with the three answers are NEW, RENEW, and CONT.

The formatting of each output field is also determined at this time. Numeric data are always flush right. However, literal text—whether a heading, element of a list, or text associated with the binary choices—may be flush left or right or may be centered in the print field. The width of each column is calculated as the greater of the length of the heading or the width of the input field. Column width is presented to the user, who is asked whether it is acceptable. A **no** answer requires that the user specify the width that is desired. If the heading exceeds the width of the input field, giving a smaller width causes the heading to be formatted on more than one line. It may be desirable to print literal data themselves on more than one line. The average name in a list of people may be only fifteen characters. One name of thirty characters, however, would make the field unnecessarily wide. Thus if the width were specified as fifteen, longer names would be displayed on two lines, without splitting the words.

**sorting blocks and logical records**

GENENTER also asks for each question whether the user wants the data to be stored in a sorting block, which refers to how data are stored in a file. All APL data types (binary, integer, decimal, and literal) are first converted to the integer type before being written into the file. Thus each logical record consists of a series of numbers. Successive logical records are concatenated without demarcation. All the normal processes of selection, sorting, producing totals, etc. may be performed on the data stored in this fashion. However, if it is known ahead of time that such functions are to be performed frequently, having the fields stored in a separate block or series of blocks speeds the process. This represents redundant storage, i.e., data are stored in the logical vectors and in the sorting blocks.

**selective branching**

GENBRANCH is the second programmer function that is related to data input. This used to be part of GENENTER, but it was found that most people were not in a position to use it fully until they had finished their program and had seen the documentation.

Selective branching may occur on the basis of the presence or the absence of data in any field. Quantitative conditions may be defined explicitly or by range.

Branching most frequently consists of skipping a set of questions. In Q3 of Appendix 8, if the grant had not been funded, Q8 and Q11–14 would have been omitted since they pertain only to grants already funded.

Branching (illustrated in Appendix 9) may also consist of adding questions, which means repeating certain questions because

questions that do not exist cannot be addressed. Questions are automatically repeated if an error check occurs as a result of the parameters defined in GENENTER, e.g., too large or too small a number. However, selective branching may be used to correct other errors detected as a result of calculations in some other part of the program. Text may be defined for every branch. Thus the reason the question is repeated is stated for the user of ENTER or EDIT.

**GENFILE and variable length records**

Before file data can be entered, the person who is writing the program must execute GENFILE. The program already knows the number of possible entries in a logical record, and how many sorting blocks and counters one has requested. Thus a file that consists of fixed-length records can be constructed simply by asking for an estimate of the number of logical records anticipated. (This number may be changed at a later date.) All details of the file organization become part of the documentation. The user is also asked whether he wants an actual, or a simulated file. (A simulated file resides in the workspace.) Very few of our applications fit in an APL workspace, but the simulated file is a very useful feature in the early stages of testing an application program.

The user is also asked whether he wants fixed- or variable-length records. A variable-length record is one in which any unused space is compressed out before being filed. When read back into the workspace, the logical record is restored to its original length.

ENTER and EDIT have built into them the option of typing **u** (for unknown), which distinguishes in the file data between the absence of something—as might be indicated by a 0—and information not available or not tested. The **u** is stored in the file as the largest four-byte number (2*31)-1, which is used for no other purpose. This unique integer is also inserted automatically into the logical record where questions were skipped, and into those parts of character data fields that contain only spaces. The **u** or "not applicable" data are compressed out with variable-length records.

The average data compaction that results from the use of variable-length records in our applications is over fifty percent. Given this situation, one might wonder why the user is even given the option of fixed-length records. The reasons are that some applications are by their nature relatively fixed and that the compression and subsequent expansion increases the apparent I/O time.

**ENTER**

ENTER is executed by the user of an application program to input file data. Each time ENTER is used, a new logical record is

added to the file. All validity checks defined in GENENTER are imposed on the entering data, and the specified calculations are performed automatically. If, because of selective branching or entering the **u** option, any data necessary for a calculation are absent, the calculation is not performed, i.e., its result is unspecified (**u**). However, this is *not* the case with user-defined calculations (FN1, FN2, etc.) because the programmer may want to make use of the **u** variable in the calculation.

## Editing functions

The essential difference between ENTER and EDIT is that ENTER adds a new logical record each time it is executed, whereas EDIT modifies logical records already in the file. EDIT asks which logical records to edit—accessed by means of the ID number—and which questions to change. EDIT simply writes over existing data. Note also that EDIT may be used in lieu of ENTER to add data to an existing record. If, for example, data are being entered about patients on periodic visits, ENTER is used only for the initial visit. Questions relating to subsequent visits are skipped as defined in GENBRANCH. These data are then added for subsequent visits by using EDIT.

GENEDIT is used to modify program data. Not only does it edit data produced by GENENTER, but it also changes conditional branches. It is possible to add new questions, erase questions, and to change the sequence of questions. By design, GENEDIT modifies only the sequence in which questions are asked. It does not change the position of the data in the logical vector, nor does it affect selective branches already established, i.e., the program automatically changes the numbers associated with these program links so that they perform the functions originally intended. The QN numbers shown on the right of the first line of the display for each question are included so the user can monitor the changes that have been made.

EDITFILE is executed when the user gets a message to do so, which occurs under either of the following conditions: (1) If questions have been added via GENEDIT after file data already exist, then the length of each logical vector must be increased, i.e., the file must be expanded. This is done by EDITFILE, and **u** data are inserted into the newly created slots. (2) If the user adds data to a file that has variable-length records, the new data will have to be stored in a new place because the existing logical record does not have enough room for it. This occurs automatically, and the old record is wiped out. In time, the file becomes filled, because of the unused space where erased records were located. EDITFILE also compresses out these empty spaces.

## Functions related to data output

**REPORT and GENREPORT**

The reports we are called upon to produce fall into four general categories. In the first of these, the selected fields represent the columns, and the selected logical records represent the rows. The user function that prints the file data is called REPORT. The function that creates the program data to drive REPORT is called GENREPORT, and is illustrated in Appendix 10.

In Appendix 10 one can see the use of both the numbers associated with each field, and assigned text (e.g., NEW) to indicate sequences and selections. Logical expressions such as SINCE and GREATER THAN are comparable to reserved words in COBOL.

The numbers in parentheses under Options represent the field widths plus one space to aid the user in estimating the number of fields that fit on a page. If the number of fields needed exceeds the number that fit on a page, the user is told the number of spaces, and he is required to delete one or more fields. The number of spaces between fields is a global variable.

When REPORT is subsequently executed, it first asks a question:

```
   report
Continuous or manual printing c/m?
   m
```

At this point the keyboard unlocks to permit the alignment of the paper. A carriage return causes printing to start. Manual rather than continuous printing means the program pauses after fifty-four lines, or after each DEPT, whichever occurs first. If additional lines are required on a given page, one will be printed for each space that is entered before the carriage return. Each page begins with the report header lines. The DEPT, or whatever category has been selected, has the statement "(continued)" appended, where appropriate. A sample tabular report is shown in Figure 1.

**formatted prose**

A quite different type of report is one that generates formatted prose from the file data. An example of formatted prose is a patient discharge summary shown in Figure 2. Such summaries are required as a permanent part of every patient's hospital record. Preparation of discharge summaries is a time-consuming and costly chore for the physician; consequently, the summaries are not always written in sufficient detail. Thus this service not only frees the physician's time, but the quality of the summaries is improved.

GENSUMMARY prepares the program data to run SUMMARY. It may be helpful to think of a summary as a string of predefined

Figure 1 Tabular report format

GRANTS AND CONTRACTS

06/27/74 NEW SINCE 01/01/74 page 1

ANATO

| PRINCIPAL INVESTIGATOR | GRANT STATUS | AGENCY | PURPOSE | PERCENT FUNDED | TOTAL |
|---|---|---|---|---|---|
| Able, M. | FUNDED | NFME | STUD-TEACHNG | 70 | 34953.00 |
| Baker, J. | FUNDED | NIH | GEN-RESEARCH | 100 | 702493.00 |
| Baker, J. | PENDNG | HEW | INST-TRAING | - | 1214531.00 |
| Baker, J. | REJCTD | NSF | GEN-RESEARCH | - | 102705.00 |
| Jones, S. | PENDNG | NIMH | GEN-RESEARCH | - | 9725.00 |
| Smith, R. | FUNDED | NIH | RES-TRAINING | 85 | 99705.37 |
| FUNDED | 3 | | | 85 | 837151.37 |
| PENDNG | 2 | | | - | 1224256.00 |
| REJCTD | 1 | | | - | 102705.00 |
| TOTAL | 6 | | | 85 | 2164112.37 |

text containing vacant slots to be filled with file data during execution of SUMMARY. However, to give syntactic and semantic integrity, multiple conditions must be defined to modify the predefined text, as well as the file data.

It is important to realize that GENSUMMARY need be executed only once to create a general-purpose SUMMARY for each medical discipline. Once created, SUMMARY may be used for thousands of patients. If modifications are desired, on the basis of experience with the program, they may be made easily with GENEDITSUM.

SUMMARY operates on units called paragraphs and entries. A paragraph is defined as a unit for formatting purposes. Each time a new paragraph is defined, GENSUMMARY asks a series of questions. Is the format columnar or paragraph? If columnar, it asks how many columns, the width of each, and the space between columns. In either case, GENSUMMARY has to know the print width, indentation for the initial line (which can be zero, or a negative or positive number of spaces), and the formatting symbol that indicates whether the paragraph or column should be centered, flush left or right, or both, i.e., variable spacing with automatic hyphenation. It also asks for the number of lines. A line is a unit of text followed by a formatting symbol.

The SUMMARY for obstetrical patients happens to consist of fourteen paragraphs. Everything under DISCHARGE NOTE could have been one paragraph consisting of four lines (formatting symbols), since each has the same format (a negative indentation of three characters). However, mainly for ease of editing the program data, it has been divided into three paragraphs.

Figure 2 Example of formatted prose

```
09/29/73W  10/16/73T  SP1  OBS                          Smith, Mary
23 years                                                    1234567
ADMITTED: 09/24/73                          DISCHARGED: 09/29/73

PRIMARY DIAGNOSIS: Full term operative delivery.

ASSOCIATED CONDITIONS: none.

OPERATIONS AND/OR  PROCEDURES:  Right  mediolateral episiotomy
   and repair.  Low-mid Dewees forceps  extraction;  position:
   left occiput anterior.

SURGEON: Jones DATE: 09/25/73 ANESTHESIA: Local, nitrous oxide
   and oxygen.

COMPLICATIONS: Fetal distress -- cause: unknown.

CONDITION ON DISCHARGE: Patient: well.
    Infant's Hist. No.: unavailable
              Condition: well
                   Sex: female
                Weight: 2660 grams
-------------------------------------------------------------
DISCHARGE NOTE:
1. REASON FOR ENTERING HOSPITAL:  This 23 year old registered,
   black, married female, G2, P0, AB1, LC0, presented in early
   labor,  without vaginal bleeding, with intact membranes, at
   39 wks gestation.  Her past  history included cone  biopsy,
   anemia and hyperemesis.
2. PERTINENT PHYSICAL,  X-RAY, AND LAB FINDINGS: Admission BP:
   115/75;  cervix 2 cm dilated,  80% effaced;  uterus: 40 wks
   gestational  size;  regular;  adnexa clear;  presentation:
   vertex;  engagement:  engaged;  station on admission +1;
   nitrazine: neg;  FH: normal; no edema; reflexes: normal. No
   significant x-ray findings. HCT on admission: 33.3%; HCT on
   discharge: 31.5%;  blood type: O pos;  not immunized; VDRL:
   neg; FTA: not done; sonogram: 20 wks; placental location by
   sono: anterior; liver values normal.
3. COURSE IN HOSPITAL:  Labor was characterized by fetal  dis-
   tress.  The membranes were artificially ruptured during la-
   bor. She underwent the operative delivery stated above. The
   indication  was  fetal  distress.  Labor  stimulated  with
   pitocin; indication: insufficiently strong labor. Total la-
   bor:  14 hrs;  second stage:  2 hrs.  She subsequently ex-
   perienced  no problems.  The infant had  an apgar of 8 at 1
   minute and 9 at 5 minutes, and subsequently did well during
   the mother's stay.  The patient's specific therapy included
   pitocin and ergotrate.
-------------------------------------------------------------
DISCHARGE INSTRUCTIONS:
Disposition:  To family plan. clinic in 4 weeks.
Medications:  1. Ferrous sulfate 300 mg p.o. t.i.d. 200
              2. NYH Vitamins 1 p.o. daily 100
Send copy to: 1. A. B. Jones
                        Signed:_____M.D.
                               Name of physician: A. B. Jones
```

Thus DISCHARGE NOTE: and the text starting with 1 constitute the two lines of this paragraph. The two succeeding paragraphs each consist of one line.

The basic unit within a paragraph is called an entry. For each entry, GENSUMMARY provides six primary options: OUT, QTEXT, BRANCH, DROP, SKIP, and FN (meaning the definition of a special APL function to perform any calculations omitted in GENENTER).

QTEXT permits the following definitions: LTEXT (which precedes the file data); RTEXT (which follows the file data); the file data fields (if any) to be called; conditions for printing the concatenation of LTEXT, DATA, RTEXT; and what should be printed if the data are null (when the set is not mandatory).

In Figure 2, the third paragraph is an example of QTEXT. The ASSOCIATED CONDITIONS: is the LTEXT, and the period (.) is the RTEXT. The input question on which the associated data are based is a binary one, with no mandatory sets. There happen to be eighteen choices, including that of **other** or free text option. The user of GENSUMMARY indicates that if none of these choices were stored, the word **none** would be printed as DATA. In other situations, the absence of data could result in the suppression of printing of the whole sequence of LTEXT, DATA, and RTEXT. When there are several associated conditions, these are concatenated together and separated by commas—unless some other delimiter is indicated—and the last two elements in the string are separated by **and.** The text associated with each data field is created by GENENTER and stored as program data. None of the options under QTEXT is mandatory. Thus DISCHARGE NOTE: is both an entry and a line, and consists of only LTEXT.

BRANCH supplies the program data that permits SUMMARY to skip over (not print) any number of specified entries. The conditions for branching can be defined as any logical or numeric relation between any of the data elements in a patient's record. If the primary diagnosis were not some form of delivery, then no information about an infant would be printed.

A DROP entry is not illustrated in the obstetrical SUMMARY, but is frequently used in those from other services. It means to drop a specified number of characters from the beginning or end of a piece of text. It is usually associated with conditional branch. Consider an example such as: "She experienced diabetes as a complication." If there were more than one complication, the "a" would have to be dropped, and an "s" concatenated to "complication."

A SKIP entry simply means a blank line; SUMMARY inserts an extra formatting symbol for each such entry. This is not only used where one wants a blank line between paragraphs, but may also be used in columnar format where the number of items is not equal for each column. **Medications:** is one column, and the number and each medication represents a second column. In column one, the user of GENSUMMARY must supply a SKIP entry for each medication after the first. The program keeps track of this so imbalances cannot occur.

The special function (FN) entry provides the means of performing calculations that could have been done in ENTER. An exam-

ple would be calculating the duration of stay from the dates of admission and discharge. We usually prefer to perform such calculation in ENTER and store the result as data. However, the option of specifying calculations is provided in GENSUMMARY. It was not used in this instance.

Substituting the correct personal pronoun for **the patient** is achieved by supplying both entries, and branching to the appropriate one on the basis of the information contained in the field related to the sex of patient. The facility to substitute **an** for **a** when the following text begins with a vowel is supplied automatically.

**cumulative statistics**

Cumulative statistics should not be confused with analysis of statistical data. Rather, they represent simple frequency counts or tabulations, with appropriate totals and groupings. These statistics deal with such things as the number of patients who had certain diseases, procedures, and complications. The statistics serve the functions of summaries for internal use and governmental reporting. They also provide the basis for, rather than summaries of, research projects.

GENSTATS asks for the specification of the options for selection, sorting, totals, and formatting in a manner similar to GENREPORT. However, because the data are stored in a separate part of the file (counters), a separate programmer function is needed. STATS is the user function that prints the cumulative statistics stored in the counters.

It is important to realize that STATS and SUMMARY usually go together; i.e., if a clinical service has one program, it usually has both. They access the same data base. We tend to look upon the cumulative statistics as providing a service primarily for the department and certain physicians who are interested in clinical research. The discharge summaries are also a service for the hospital. More importantly, by saving time, they insure the cooperation of the physicians who may see no immediate rewards from the tabulations, but who have to provide the input.

Since the two programs go together, one may conceive statistics as being a long vector of counters. Every time patient data are entered into the system, the appropriate counters are incremented. The counters are defined in GENENTER as calculations, and the locations of the counters in the file are documented in the GENDISPLAY. The counters are cumulative, but there are parallel vectors for weekly, monthly, and yearly reports. Each report is printed and initialized at the appropriate interval. The raw data, however, that form the basis for the discharge summaries are kept only for about six months, at which time they are stored in tape archives.

**file searching**

Probably the greatest single benefit that accrues from permanently storing raw clinical data in a file, be this on disk or tape, is the ability to request correlations of information that were not originally anticipated. This facility provides a powerful tool for clinical research, peer review, and overall management of the clinical department.

Clinical research often consists of studying the charts of patients with certain combinations of clinical features. For example, someone may want to study the effects of treatment on the infants born to all patients with a history of conditions A or B during the first trimester of pregnancy. The basis of such searches is the ANDing and ORing of various parameters. The usual desired output is a list of history numbers that direct one to the patient charts in the records room.

Peer review is the evaluation of the quality of medical care provided by individual physicians. The criteria for data selection include such things as procedures or treatments used, the indications for such treatment, and the effect on the patient. The output of these searches usually is a list of names of physicians. Finally, a conversational function, called SEARCH, permits the selective interrogations of files. SEARCH is illustrated in Appendix 11.

## Concluding remarks

We believe that the data processing needs of a major medical center do not differ fundamentally from those of government or industry. Experience in our environment is that data processing is playing an increasingly prominent role in the full spectrum of services. Not only is the number of applications increasing, but they are also becoming more diverse and imaginative as users become more directly involved in the process.

A Tutorial System (ATS) was the first program that provided a flexible intellectual tool for our medical teaching staff.

A Program Generator (APG), derived in part from ATS, is designed to provide the same facility in clinical, research, and administrative operations. A version of APG has been operational since about October, 1973. It has been used for every application since that time because of the speed of programming, the automatic documentation, and the ease of program maintenance. To date we have provided a full range of administrative programs including accounting, ordering and billing, payroll, and personnel. The persons who request these application programs are increasingly being brought into direct control of the final product as they see the program generator being used. The ultimate goal is for the programming staff to provide advice only.

APG was developed to facilitate medical application programming as well as to involve both the physician and medical administrator more directly in the decision-making processes. Initial programming time has decreased by a factor of ten, and—because of automatic documentation and editing functions—the cost of program maintenance or modification has decreased even more. Ease of programming derives not so much from the use of a natural language interface as from the interrogative form of man-machine communication. Rather than being required to structure and sequence a series of statements, the user is asked explicit questions, with adequate default options at every point.

The precursor of a fully automated medical records system including clinical laboratories is now operational. We expect this system to grow in breadth, depth, and utility as we increasingly provide a data processing facility that allows flexible definition and redefinition of the requirements for medical records by the clinical staff and administration.

Cornell University Medical College has a free-access policy for computer usage by faculty and students. An increasing amount of teaching and clinical and laboratory research is being performed with the assistance of the data processing facility. We have begun an educational program to orient both students and staff toward using the computer in helping to solve their everyday problems in teaching, research, and clinical practice.

Such data-driven APL programs are used to meet the data processing needs of Cornell University Medical College and many of those of its affiliated hospitals.

ACKNOWLEDGMENTS

This research was begun while Dr. Hagamen was a Visiting Fellow at the IBM Systems Research Institute. The work was supported in part by Grant Nos. 50-68 and 67-73 from the National Fund for Medical Education, Grant No. 1 008 PE 00480 from the Department of Health, Education and Welfare, and Grant No. MH25621 from the National Institutes of Mental Health. The computing facilities of the Integrated Data System Laboratory, the IBM Systems Research Institute, and the Philadelphia Scientific Center were made available to the authors as Research Fellows of the Systems Research Institute. The authors wish to acknowledge the assistance of Mr. S. W. Dunwell, and Drs. K. E. Iverson and E. S. Kopley. We also wish to thank the SRI students who offered constructive suggestions during the early development of the program generator.

CITED REFERENCES

1. W. D. Hagamen, *The Functioning Brain of Man*, Chas. Pfizer & Co., Inc., New York, N.Y. 10017 (1966).

2. W. D. Hagamen, D. J. Linden, H. S. Long, and J. C. Weber, "Encoding verbal information as unique numbers," *IBM Systems Journal* **11,** No. 4, 278–315 (1972).
3. W. D. Hagamen, D. Linden, M. Leppo, W. Bell, and J. C. Weber, "ATS in exposition," *Computers in Biology and Medicine* **3,** No. 3 205–226, Pergamon Press, Elmsford, New York (1973).
4. D. J. Linden, W. D. Hagamen, and J. C. Weber, *CORTEX—An APL Text Editing System*, Cornell University Medical College, New York, New York (1973).

## Appendix 1: Brief and verbose modes

```
      genenter
Q1 ***************
LT:
      x
Long text:
      GRANT ID NO:
Brief text:
      ID:
Q1: numeric input
Q1: 1 entry
Q1: no decimals
Give max value permitted
      999999999
Give min value permitted
      10000000
Print format y/n?
      y
Entry 1 Heading:
      ID:
Print width 9  y/n?
      y
Width 9
Q2 ***************
Long text (or c=calculation, r=repeat):
```

## Appendix 2: Binary data

```
      genenter
Q2 ***************
Long text (or c=calculation, r=repeat):
      GRANT TYPE:¢
      ss 1=NEW¢
      ss 2=RENEWAL¢
      ss 3=CONTINUATION
Brief text:
      GT:
Data type: binary, numeric, date, character, list
      b
How many choices?
      3
Is 0 permitted y/n?
      n
Define exclusive sets, from which only one choice
is permitted (0=none)
      1 3
Define mandatory sets, from which at least one choice
is required (0=none)
      1 3
Print format y/n?
      y
Heading 1:
      GRANT TYPE
```

```
Text for each choice:
1
      NEW
2
      RENEW
3
      CONT
Print width 10 y/n?
      n
Specify desired width:
      5
Width 5
Do you wish to store this input in a sorting block y/n?
      y
Q3 ***************
Long text (or c=calculation, r=repeat):
```

## Appendix 3: List data

```
      genenter
Q3 ***************
Long text (or c=calculation, r=repeat):
      GRANT STATUS:
Brief text:
      GS:
Data type: binary, numeric, date, character, list
      l
How many entries?
      1
Give list separated by / . Additions can be made later
(p=previous list):
      FUNDED/PENDNG/REJCTD
Max number of characters - up to 20
      6
Print format y/n?
      y
Entry 1 Heading:
      GRANT STATUS
Print width 12 y/n?
      n
Specify desired width:
      6
Width 6
Do you wish to store this input in a sorting block y/n?
      y
Q4 ***************
Long text (or c=calculation, r=repeat):
```

## Appendix 4: Character data

```
      genenter
Q4 ***************
Long text (or c=calculation, r=repeat):
      PRINCIPAL INVESTIGATOR:
Brief text:
      PI:
Data type: binary, numeric, date, character, list
      c
What is the maximum number of characters permitted?
      24
Is an empty vector permissible y/n?
      n
Print format y/n?
      y
Heading:
      PRINCIPAL INVESTIGATOR
Print width 24 y/n?
      n
Specify desired width:
      12
Width 12
Do you wish to store this input in a sorting block y/n?
      y
Q5 ***************
Long text (or c=calculation, r=repeat):
```

## Appendix 5: Dates

```
      genenter
Q7 ***************
Long text (or c=calculation, r=repeat):
      PROPOSED STARTING DATE:
Brief text:
      PSD:
Data type: binary, numeric, date, character, list
      d
How many entries?
      1
Give latest date permitted (n=no limit, c=current):
      n
Give earliest date permitted (n=no limit, c=current):
      c
Print format y/n?
      y
Entry 1 Heading:
      PROPOSED STARTING DATE
Print width 22 y/n?
      n
Specify desired width:
      8
Width 8
Do you wish to store this input in a sorting block y/n?
      n
Q8 ***************
Long text (or c=calculation, r=repeat):
```

## Appendix 6: Numeric questions

```
      genenter
Q10 ***************
Long text (or c=calculation, r=repeat):
      AMOUNT REQUESTED:
Brief text:
      AR:
Data type: binary, numeric, date, character, list
      n
How many entries?
      1
How many decimal places shall be stored?
      2
Give max value permitted
      10000000
Give min value permitted
      5000
Print format y/n?
      y
Entry 1 Heading:
      AMOUNT REQUESTED
Print width 16 y/n?
      n
Specify desired width:
      11
Width 11
Do you wish to store this input in a sorting block y/n?
      n
Q11 ***************
Long text (or c=calculation, r=repeat):
```

## Appendix 7: Interfield calculation

```
      genenter
Q13 ***************
Long text (or c=calculation, r=repeat):
      c
Take input from which Q's?
      11 12
```

```
Q11 has 10 entries
Which entries go into this calculation?
      6
OP CODE?
1=counter
2=sum
3=difference
4=product
5=quotient
6=average
7=percent
8=equality
*******
26=FN1, etc.
      8
Give the length of the result to be stored in V
      0
Comment?
      Checks calculated total (Q12)¢
      against entered total (Q11)
Q14 ***************
Long text (or c=calculation, r=repeat):
```

The purpose of the calculation in Appendix 7 is to check on the accuracy of previously entered data. If the two figures are not identical, the data must be reentered. The result is not stored in the logical vector (V), but is used later to effect a branch back to question 11. For this reason, questions normally asked regarding decimal positions and print format are omitted.

The list of possible calculations is infinite. Therefore, it is not desirable to store functions that are seldom used. We presently store twenty-five such calculations, eight of which are shown in the (abbreviated) OP CODE in Appendix 7. If the needed calculation is not on the list, the user writes his own APL function. The only restrictions are that the functions be named FN1, FN2, etc., have an explicit output that represents the result of the calculation, and create no global variables. Each such user-defined function is appended as the last line of the function CALCULATE. Fields that are defined as inputs for the functions become their right arguments. The user indicates FN1 by entering OP CODE 26, FN2 by OP CODE 27, etc.

Fields involved in the calculation, as well as the OP CODE, are included automatically in the documentation. With a user-defined calculation, the OP CODE reads FN1, FN2, etc. The purpose of the comment is to describe the function FN.

## Appendix 8: Documentation

The following is a display of the documentation of the GRANTS AND CONTRACTS program we have been generating. All fourteen questions are shown, even though we have illustrated only seven questions with GENENTER. The V referred to under **Storage:** is the vector of integers that represent the logical record.

```
  gendisplay
Title: GRANTS AND CONTRACTS
Total entries (characters & numbers): 80
Input bytes: 256
Stored bytes: 152 (including Sorting Blocks)
Total number of Questions: 14
Questions whose answers affect Branching: 3 13
Calculations: 12-14
Binary Questions: 2
Numeric Questions: 1 10 11
Date Questions: 7 8
Character Questions: 4
List Questions: 3 5 6 9
Questions stored in Sorting Blocks: 2-6 8 9 12

File organization:
Rsecurity: 123456789  Wsecurity: 987654321
Queuing code: grantcon
Block 1 of this file is block 1 of the dataspace
File size: 107 blocks
Index: blocks 1-8
Records: blocks 9-59
Record length: Variable up to 26
File capacity: from 1004 to 2048 records
Counter: none
Sorting: blocks 60-107


Q1************************(QN1)
Long text:
  GRANT ID NO:
Brief text:
  ID:
Input: 1 numeric entry; no decimals.
Max. value = 999999999
Min. value = 10000000
No encoding
Storage: V(1)
Branching: none
Print format:
1. ID1:  999999999 Width 9

Q2************************(QN2)
Long text:
  GRANT TYPE:¢
  ss1=NEW¢
  ss2=RENEWAL¢
  ss3=CONTINUATION
Brief text:
  GT:
Input: 3 binary choices; zero NOT permitted.
Exclusive sets: 1-3
Mandatory sets: 1-3
Storage: V(2)= (Base 2) encode choices 1-3
Input also stored in sorting block
Branching: none
Print format:
1.   GRANT TYPE1  Width 5
     1  NEW1
     2  RENEW1
     3  CONT1

Q3************************(QN3)
Long text:
  GRANT STATUS:
Brief text:
  GS:
Input: 1 item from list 3
Max. size of entry on list = 10 characters
Storage: V(3)
Input also stored in sorting block
Branching:
1. Answer ≠ 1
  Branch text: none.
   Skip Q's 8 11-14
Print format:
1. GRANT STATUS1: (list reference) Width 6
```

```
Q4************************(QN4)
Long text:
  PRINCIPAL INVESTIGATOR:
Brief text:
  PI:
Input: 24 characters; empty vector NOT permitted.
Storage: V(4-8)
Input also stored in sorting block
Branching: none
Print format:
PRINCIPAL INVESTIGATOR1: (literal text) Width 12

Q5************************(QN5)
Long text:
  DEPT:
Brief text: SAME
Input: 1 item from list 5
Max. size of entry on list = 5 characters
Storage: V(9)
Input also stored in sorting block
Branching: none
Print format:
1. DEPT1: (list reference) Width 5

Q6************************(QN6)
Long text:
  AGENCY:
Brief text: SAME
Input: 1 item from list 6
Max. size of entry on list = 5 characters
Storage: V(10)
Input also stored in sorting block
Branching: none
Print format:
1. AGENCY1: (list reference) Width 6

Q7************************(QN7)
Long text:
  PROPOSED STARTING DATE:
Brief text:
  PSD:
Input: 1 date (M,D,Y)
Latest date: no limit
Earliest date: CURRENT
Encoded storage:
  V(11)= (Base 10000 100 100) encode Y,M,D
Branching: none
Print format:
1. PROPOSED STARTING DATE1: MM/DD/YY Width 8

Q8************************(QN8)
Long text:
  STARTING DATE:
Brief text:
  SD:
Input: 1 date (M,D,Y)
Latest date: no limit
Earliest date: CURRENT
Encoded storage:
  V(12)= (Base 10000 100 100) encode Y,M,D
Input also stored in sorting block
Branching: none
Print format:
1. STARTING DATE1: MM/DD/YY Width 8

Q9************************(QN9)
Long text:
  PURPOSE:
Brief text: SAME
Input: 1 item from list 9
Max. size of entry on list = 15 characters
Storage: V(13)
Input also stored in sorting block
Branching: none
Print format:
1. PURPOSE1: (list reference) Width 12
```

```
Q10************************(QN10)
Long text:
  AMOUNT REQUESTED:
Brief text:
  AMOUNT:
Input: 1 numeric entry; 2 decimal places
Max. value = 10000000.00
Min. value = 5000.00
No encoding
Storage: V(14)
Branching: none
Print format:
1. AMOUNT REQUESTEDr: 10000000.00 Width 11

Q11************************(QN11)
Long text:
  ssSALARY¢
  ssWAGES¢
  ssEQUIP¢
  ssOTHER EXPENSES¢
  ssINDIRECT CHARGES¢
  ssTOTAL¢
  ss2ND-YR¢
  ss3RD-YR¢
  ss4TH-YR¢
  ssOTHER YEARS
Brief text: NONE
Input: 10 numeric entries; 2 decimal places
Max. values = 2000000.00 5000000.00 1000000.00 1000000.00
2000000.00 10000000.00 2000000.00 2000000.00 2000000.00
6000000.00
Min. value = 0.00
No encoding
Storage: V(15-24)
Branching: none
Print format:
1. SALARYr:  2000000.00 (entry 1) Width 10
2. WAGESr:  5000000.00 (entry 2) Width 10
3. EQUIPr:  1000000.00 (entry 3) Width 10
4. OTHER EXPENSESr:  1000000.00 (entry 4) Width 10
5. INDIRECT CHARGESr:  2000000.00 (entry 5) Width 10
6. TOTALr:  10000000.00 (entry 6) Width 11
7. 2ND-YRr:  2000000.00 (entry 7) Width 10
8. 3RD-YRr:  2000000.00 (entry 8) Width 10
9. 4TH-YRr:  2000000.00 (entry 9) Width 10
10. OTHER YEARSr:  6000000.00 (entry 10) Width 10

Q12************************(QN12)
Calculation:
Input:
  Q11 entries 1-5
Operation: sum
Result: 1 element; 2 decimal places
Storage: V(25)
Input also stored in sorting block
Branching: none
Print format:
1. TOTALr: Width 11

Q13************************(QN13)
Calculation:
  Checks calculated total (Q12)¢
  against entered total (Q11)
Input:
  Q11 entry 6
  Q12 entry 1
Operation: equality
Result: 0 elements
Result not stored
Branching:
1. Answer = 0
  Branch text:
  Total does not check. Re-enter Q11.
  Add Q11
Print format: not defined
```

```
Q14************************(QN14)
Calculation:
Input:
  Q10 entry 1
  Q12 entry 1
Operation: percent
Result: 1 element; no decimals.
Storage: V(26)
Branching: none
Print format:
1. PERCENT FUNDEDr: Width 7
```

The text formatting symbols are shown in GENDISPLAY, but not in ENTER or EDIT. The reason for this is that the symbols as well as the text may be changed in GENEDIT. In addition to ¢ for carriage return and s for space, c, l and r stand for centered, flush left, and flush right, respectively. Flush left is the default option. As previously indicated, these symbols are actually the letter over-struck with an upper case F. They are shown here with the APL underline, so they will be more readable.

## Appendix 9: Selective branching

```
      genbranch
Which Q?
      3
BT:
 GS:
OK y/n?
      y
Answers defined explicitly or by range: e/r?
      e
Which answers cause the branching?
      1
p/a? p=branch occurs in PRESENCE of answer;
     a=branch occurs in ABSENCE of answer.
      a
Which Q's are to be added?
      0
Which Q's are to be skipped?
      8 11 12 13 14
Branch text

More branches for Q3 y/n?
      n
Which Q?
      13
 *Calculation*
OK y/n?
      y
Answers defined explicitly or by range: e/r?
      e
Which answers cause the branching?
      0
p/a? p=branch occurs in PRESENCE of answer;
     a=branch occurs in ABSENCE of answer.
      p
Which Q's are to be added?
      11
Which Q's are to be skipped?
      0
Branch text
      Total does not check. Re-enter Q11.
More branches for Q13 y/n?
      n
Which Q?
      0
```

```
      search
What do you want to search for? (s=stop)
      INFANT WEIGHT
Max. value (u=unlimited):
      2500
Min. value:
      0
and/or? (s=stop)
      a
      INDUCED LABOR
Max. value (u=unlimited):
      5
Min. value?
      1
and/or? (s=stop)
      s
Press RETURN to begin
```

## Appendix 10: GENREPORT

```
      genreport
Enter what you want printed in print sequence.
Options y/n?
      y
  1=ID (10)
  2=GRANT TYPE (6)
  3=GRANT STATUS (7)
  4=PRINCIPAL INVESTIGATOR (13)
  5=DEPT (6)
  6=AGENCY (7)
  7=PROPOSED STARTING DATE (9)
  8=STARTING DATE (9)
  9=PURPOSE (13)
 10=AMOUNT REQUESTED (12)
 11=SALARY (11)
 12=WAGES (11)
 13=EQUIP (11)
 14=OTHER EXPENSES (11)
 15=INDIRECT CHARGES (11)
 16=TOTAL (12)
 17=2ND-YR (11)
 18=3RD-YR (11)
 19=4TH-YR (11)
 20=OTHER YEARS (11)
 21=TOTAL (12)
 22=PERCENT FUNDED (8)

First will be heading and produce total:
      5 4 3 6 9 22 21
Which should produce subtotals (0=none):
      3
Total (0=none):
      21
Average (0=none):
      22
Enter sort sequence (s=same as print):
      s
Selections (0=none):
      2 7
Selections for each choice:
2
      NEW
7
      SINCE 1/1/74
```

## Appendix 11: SEARCH

```
      search
What do you want to search for? (s=stop)
      INFANT WEIGHT
```

```
Max. value (u=unlimited):
      2500
Min. value:
      0
and/or? (s=stop)
      a
      INDUCED LABOR
Max. value (u=unlimited):
      5
Min. value?
      1
and/or? (s=stop)
      s
Press RETURN to begin
```

Reprint Form No. G321-5007