Behavior Research Methods & Instrumentation
1975, Vol. 7 (5), 464-470

COMPUTER TECHNOLOGY

An extension of Kieras' general experiment programming system

HENRY M. HALFF, KARL W. SCHOLZ, and JANET H. WALKER
University of Illinois at Urbana-Champaign, Champaign, Illinois 61820

An extension of Kieras' general experiment programming system for computer-aided experimentation is described. The extension includes a support system for program preparation and data recovery and an efficient experiment control system. The result is a system that can run multiple subjects independently with a minimal amount of effort expended in preparation for the experiment. The implementation procedure for this system is outlined, and the paper concludes with a discussion of the type of facility which could make best use of the system.

This paper describes an extension of the programming language and operating system described by Kieras (1973) for computer-aided experimentation. Kieras' general experiment programming system (GEPS) was initially implemented on an IBM 1800 to control human experimentation at a number of CRT/keyboard terminals. There are two basic features of Kieras' original system: An assembler translates programs written in a high-level source code into a compact object code. An operating system then runs multiple terminals independently through the use of a simple monitor which schedules an object-code interpreter to the various terminals.

The version of GEPS described here is currently running at the University of Illinois in a setting similar to the original implementation. We have extended the system in two major ways. First, we have developed an interactive support system for stimulus preparation, data recovery, and debugging. Second, the operating system has been changed from a simple round-robin system to a multiple-queue system which makes more efficient use of I/O wait time.

The purpose of this paper is to describe the extended system, to outline the implementation procedure, and to describe the type of facility for which the system is appropriate. It is fitting to both begin and end with the latter topic. The first questions which come to the mind of a potential user probably concern the kind of experiments that can be run by the system. Unfortunately, a full discussion of these questions hinges on a more complete knowledge of the system itself. We can, however, give some examples of experiments that have been run with GEPS. At Michigan, GEPS has controlled research in psychophysics, problem solving, computer-assisted instruction, and, no doubt, in other areas. At Illinois, GEPS has been in service for about 20 months; and during that time it has controlled experiments in verbal learning, syllogistic reasoning, retrieval time from semantic memory, group decision making, preference and judgment in risky decisions, and has even presented stimuli an experiment tracking the eye movements of subjects who were reading pointers on numerical scales. These examples are clearly limited by hardware and should not be taken as representing the full range of applicability of the system.

A much better view of the generality of GEPS can be gained by viewing the system more as a way of implementing a language than as the specification of a language itself. GEPS languages largely consist of a set of macroinstructions or commands. This set will vary from installation to installation as dictated by the demands of the research and the restrictions imposed by software. While some (e.g., Castellan, 1973, 1975; Pilla, 1973; Wood, Sette, & Weiss, 1975) might argue that such a philosophy encourages reinvention of the wheel, others (e.g., Lewis, Osgood, & Hebert, 1973; Pitz, 1975; Polit, 1974) have made good arguments for our approach. In any case, our purpose is not to offer a final resolution to this problem but rather to provide an easy implementation for those who choose to invent their own wheels. We begin with a description of the GEPS system at Illinois.

Funds for development of the system were provided by the Research Board of the Graduate College of the University of Illinois, by Grant OEG-572-0016(509) from the Office of Education to the first author, and by Grant SOC-7305697-01 from the National Science Foundation to James Davis of the University of Illinois. The authors are especially indebted to David Kieras of the University of Michigan for his advice and assistance in developing the system, to James Greeno of the University of Michigan and Frank Restle of Indiana University for providing software developed in their laboratories, and to Peter Polson of the University of Colorado for suggestions made in reviewing an earlier version of the paper. Requests for reprints should be sent to Henry M. Halff, Department of Psychology, University of Illinois, Champaign, Illinois, 61820. Janet H. Walker is now at Stanford University.

GEPS AT ILLINOIS

At Illinois, GEPS consists of two separate subsystems. A *support system* allows the user to prepare and maintain stimulus materials and programs and to recover data from experiments. An *experiment control system* actually runs the experiments.

The Support System

By way of introduction to the details of this system, we should point out, as have Bailey and Polson (1975), that the functions played by the system are neither dispensable nor trivial. The power of any on-line system is clearly limited by the facilities for getting stimuli and programs into the system and for getting data out. Some (e.g., Churchill, Naess, & Olivier, 1971; Spear, Overgard, & Christian, 1975) have solved these problems by imposing particular formats on the user and developing utilities which deal only with those formats. Others (e.g., Kieras, 1973) provide complete flexibility in running the experiment but leave the user essentially on his own with regard to support services. While both of these alternatives may be defensible, it is entirely possible to develop easily used facilities for program and stimulus preparation and for data recovery and, at the same time, maintain the full flexibility of a higher level programming language.

The support system consists of a collection of utility programs and a monitor. The utilities include a text editor, programs for stimulus preparation, debugging, and data recovery, and the GEPS language assembler. The monitor serves as the overall supervisor for the entire system, decoding commands entered by the user, and implementing the commands by transferring control to the appropriate utility program. As each utility completes the specified operation, it transfers control back to the monitor.

The most heavily used utility in program preparation is the text editor. This utility is used to create and maintain in text form virtually all of the materials necessary to run the experiment. These materials appear to the user as decks of sequenced punched cards but are maintained on the disk using the linked allocation scheme characteristic of many list processors (cf. Knuth, 1973, pp. 251-270). Thus, the editor can easily perform the tasks of inserting, deleting, shuffling, resequencing, and outputting the material with a minimal amount of actual data transfer.

The central role of the editor in the system cannot be overemphasized. In preparing for an experiment, the user will first punch drafts of his programs, stimulus materials, and data-recovery specifications onto cards. These items are read into a special text file which is accessible to the editor. Each program, set of stimuli, etc., is kept in a separate *text block* in the file and is accessed via a user-assigned name. One recent experiment, for example, required four blocks containing programs for delivering instructions, running the experiment proper, terminating the experiment, and monitoring the subjects' progress. Two other blocks contained the test of the instructions and that of two free-recall lists. Since the text blocks are both necessary and sufficient specifications of the procedure of the experiment, debugging consists of using the editor to adjust their contents until the experiment runs properly. Before the experiment can even be tried, however, each text block must be properly translated and/or formatted for the experiment control system.

Several utilities are available for these chores, including the GEPS language assembler. The need for such utilities is justified by the fact that, apart from the terminals, the experiment control system has access to only one external device, the disk. Since the format of material read or written by this system is dictated more by the demands of efficiency than by readability, a battery of utilities formats and recodes programs, stimuli, and data. In performing these tasks, the utilities have direct access to the stimulus materials and format specification maintained by the text editor. In addition, the data-recovery utility allows output of data to a number of different devices, including the text editor itself.

There are two programs responsible for stimulus preparation. One of them, STIMI, formats stimuli according to user specifications and places the formatted material in a file accessible by the experiment-control system. The text-block input to this utility consists of a few lines of formatting information followed by the stimuli themselves. The output consists either of fixed-length or variable-length records. In the latter case, each record is headed by a single word giving its length. While fixed-length records are best suited to random access, the utility also provides for random access to variable-length records by optionally including a list of record locations as one record of the file. An example of an input text block and the corresponding output file for this utility are illustrated in Figure 1. Note the simple yet flexible FORTRAN-like format specification and the facility for using a sequence of different formats. In this example, numeric and text information are isolated in separate records; while STIMI allows intermixing of such information, the combination of control and display information is often best accomplished by another utility, TEXTI.

TEXTI has the ability to compile large amounts of text, such as instructions or CAI materials, into the body of a variable-length record while assembling associated control information into the record header. A single GEPS language instruction can then display the text to the subject and, at the same time, present control information to the program running the subject. TEXTI lacks the powerful formatting ability of STIMI, but it is in general simpler to use.

Just as the system can deal with a wide range of stimuli, so also can it deal with a wide variety of data types. Data is collected in variable-length records, and each record is headed by the record length, program and terminal number, a user-defined format code, the time, and the date; the last three items are optional. The body of the record is completely determined by the user; it may contain numeric information, text, or a mixture of both. The data-recovery routine operates in conjunction with a text block which specifies the format of the data both as it appears on the data file and as it should appear on the output devices. As the example in Figure 2 illustrates, both input and output formats are similar to these used in STJMI. In addition to compiling the body of the data record, the utility assigns a subject number to the records from each subject and, at the user's option, recovers the header information as well. Finally, the data-recovery utility is responsible for arranging the output records in subject-by-subject order since, in the usual case, they are written to the data file in essentially random order.

We turn now to the most important utility in the support system.

The GEPS language assembler. The assembler is responsible for translating the source code which specifies the experimental procedure into a compact object code which can be implemented by the experiment control system. To define the experimental procedure, the user writes one or more programs which will run the experiment at a terminal and one or more programs which will allow the experimenter to monitor

```
Format Specification Text Block
--------------------------------------------------------------------
FILE GEPS1 1          File name and number.
RECORD 1 2I 1A        Data record Type 1--two integers, one text word.
RECORD 2 3A 1I        Data record Type 2--three text words, one integer.
FORMAT 1 2I5 A2       Output Format 1.
FORMAT 2 A6 2X I4     Output Format 2.
SEQUENCE 1,1 2,2      Use Record 1, Format 1 with Data Code 1,
                      and Record 2, Format 2 with Data Code 2.
TERM -1               A -1 marks the end of a subject.
====================================================================
Data File (one 16-bit computer word per line)
--------------------------------------------------------------------
                      Length      3
        Header  Program, Subject  1, 2
                     Data code    1      Record 1
                                  5
                       Contents 7235
                                 HI
                      Length      4
        Header  Program, Subject  1, 2
                     Data code    2
                                 TH      Record 2
                                 ER
                       Contents
                                 Eb
                               5726
====================================================================
Output (First field is user-assigned subject number.)
--------------------------------------------------------------------
bbbb1bbbb5b7235HI     Record 1
bbbb1THEREb5726       Record 2
--------------------------------------------------------------------
Spaces are shown us "b".
```

Figure 2. Example of format specifications for, input to, and output from data recovery utility.

```
Input Text Block
--------------------------------------------------------------------
FILE GEPS1 1          File name and number.
FORMAT 1 2I1 X5 I3    First data format--three numeric fields.
FORMAT 2 A8           Second data format--one character field.
SEQUENCE 1 2 2        Use formats in Sequence 1, 2, 2, 1, 2, 2, . . .
FINIS                 End of format specifications
1 bbbbb275            Record 1 --written with FORMAT 1.
WORD1                 Record 2 --written with FORMAT 2.
WORDbTWO              Record 3 --written with FORMAT 2.
====================================================================
Stimulus File (one 16-bit computer word per line)
--------------------------------------------------------------------
        Record length    3
                         1
        Record contents  0       Record 1
                       275
        Record length    3
                        WO
        Record contents RD       Record 2
                        1b
        Record length    4
                        WO
                        RD       Record 3
        Record contents
                        bT
                        WO
                         .
                         .
                         .
--------------------------------------------------------------------
Spaces are shown as "b".
```

Figure 1. Example of STIMI input and output.

and intervene with the procedure at any terminal. The details of the language are presented in Kieras (1973). Since we have modified Kieras' version of the language only to the extent of adding convenience features such as implicit allocation of variables, we need only mention the most fundamental aspects of the language.

As with most languages, a program consists of data-specification statements and executable instructions. The data-specification statements serve to define the names and extent or contents of the variables or constants manipulated by the program during the experiment. Due to the time-sharing nature of GEPS, the user must also distinguish between constants and variables in his specifications. Constants, whose values never change during the course of an experiment, are placed in a *constants area* which is shared by all terminals using the program. Variables, whose values may change from time to time for any subject, are allocated to *variable areas* which are replicated for each subject.

Each of the executable instructions consists of a command followed by a list of operands. The command specifies the type of operation which the instruction performs, and the operands specify the locus of the operation in the form of parameters or references to the constants or variable area. Flow control is accomplished by allowing some operands to be labels which prefix executable instructions.

Our command vocabulary can be roughly divided into six classes: arithmetic and logical, string processing, flow control, terminal I/O and timing, Disk I/O, and interterminal communication. It would be clearly inappropriate to discuss all 73 commands of the Illinois GEPS system, but the description of a subset of these commands is given in Table 1 to provide the reader some feel for the language. As a further aid in this regard, we have rewritten Kieras' (1973) program for the Sternberg's (1966) memory-scan experiment. This program, presented in Figure 3, uses the Illinois version of the language, but retains the procedure used by Kieras except where noted. In examining these materials, the reader should keep in mind that they represent only a small fraction of the system's capabilities and that even the set of commands in Table 1 may not be appropriate for every installation.

Table 1
Brief Description of a Subset of Illinois GEPS Language

	Data Definition Statements
NC *name value*	Numeric constant
SC *name [contents]* *	String constant
SV *name length*	String variable
SN *name parent <offset>***	Subnumber
	Commands
ADD *increment sum*	Replace *sum* with *increment* + *sum.*
CURSE *position string*	Move the cursor to *position* and display *string.*
DATA-OUT *file string*	Output *string* on disk file number *file* with format code *code*; go to *@eof* on end of file.
DELAY *time scale*	Delay *time scale* milliseconds; *scale* = 1 if omitted.
ERASE	Erase terminal screen.
GEN-RANDOM *range value*	Set *value* to a random number between 0 and *range* – 1.
INPUT *response length RT scale*	Wait for subject's response; put the response in *response*, the number of characters typed in *length*, and the reaction time (in *scale* milliseconds) in *RT*; *scale* = 1 if omitted.
LOOP *@label index limit increment*	Add increment (1 if omitted) to *index*; transfer control to *@label* if *index* > *limit.*
MOVE *source target*	Replace contents of *target* with those of *source.*
PERMUTE *string*	Randomly permute the words of *string.*
SCAN *source target offset*	Set *offset* to offset of first occurrence of *target* in *source*; if *target* is not in *source*, set *offset* to –1.
SET *target source*	Replace the contents of the first word of *target* with those of the first word of *source* (0 if omitted).
START-TIME *flag*	If *flag* is 0 or omitted, start the reaction-time timer immediately; otherwise, start it with the next display instruction.
STOP	Stop execution.
Target = *expression*	Replace first word of *target* with value of infix *expression.*
Subscripts	Refer to substring of *string.*
String<offset;length>†	Offset and length both specified.
String*<offset;>*	Offset specified; length is rest of *string.*
String*<;length>*	Offset is 0; length specified.

**Character contents of strings are delimited by brackets. **Offsets are from beginning of parent string, starting at 0. †Use of a semicolon as separator specifies offset and/or length in words; use of a comma specifies bytes.*

```
GEPS1 3 STERNBERG EXPERIMENT
NC OUTFILE 1                              (OUTPUT FILE NUMBER)
SC DIGITS [ 1 2 3 4 5 6 7 8 9 0 ]         (STIMULI TEXT)
SV SET 20                                 (20-BYTE STIMULUS SET)
SV RESP 1                                 (1-BYTE RESPONSE)
SV DATA 8                                 (DATA RECORD)
          SN S DATA<0>                    (POSITIVE SET SIZE)
          SN POSNEG DATA<1>               (0--POSITIVE, 1--NEGATIVE)
          SN EC DATA<2>                   (0--CORRECT, 1--ERROR)
          SN RT DATA<3>                   (REACTION TIME)
**********************************************************************
START
          MOVE DIGITS SET                 (FETCH STIMULI)
          ERASE
** LOOP ON TRIALS
          SET TRIAL
@TRIAL    GEN-RANDOM 5 S                  (CHOOSE POSITIVE SET SIZE)
          ADD 1 S
          GEN-RANDOM 1 POSNEG             (CHOOSE PROBE TYPE)
          PERMUTE SET                     (CHOOSE PROBE AND POSITIVE SET)
          PROBEPOS=POSNEG*S               (FETCH PROBE)
          SET PROBE DIGITS<PROBEPOS>
          PERMUTE SET<;S>
** LOOP TO DISPLAY MEMBERS OF POSITIVE SET
          SET X
@DISPLAY CURSE 1208 SET<X;1>              (DISPLAY STIMULUS)
          DELAY 1200                      (FOR 1200 MSEC)
          LOOP @DISPLAY X S               (END OF DISPLAY LOOP)
**
          CURSE 1208 [X]                  (DISPLAY WARNING SIGNAL)
          DELAY 5000                      (FOR 5 SEC)
          START-TIME 1                    (START RT TIMER)
          CURSE 1208 PROBE                (DISPLAY PROBE)
          INPUT RESP 0 RT                 (COLLECT RESPONSE)
          ERASE
** "R" KEY IS DESIGNATED AS POSITIVE, "I" AS NEGATIVE.
** THIS PROCEDURE DIFFERS FROM KIERAS' SINCE HIS LAB HAS PUSHBUTTONS.
          SCAN [RI[ RESP X                (DECODE RESPONSE)
          EC = X.EOR.POSNEG               (EVALUATE RESPONSE)
          DATA-OUT OUTFILE DATA 1 @EOF (OUTPUT DATA)
          DELAY 5000                      (5 SEC ITI)
          LOOP @TRIAL TRIAL 20            (END OF TRIALS LOOP)
**
@EOF      STOP
```

Figure 3. Sample program for memory-scan experiment.

GEPS object code is essentially a more compact form of the source code. An object program has three main parts: the constants area, described above; a set of string definition vectors defining the area, location, and extent of each string; and an instruction list which consists of the code for each executable instruction. This structure has three important implications. First, GEPS maximizes the extent of information sharing among terminals. Two terminals using the same program will share the same constant area and instruction list. Second, the assembler is independent of the experiment-control system. That is, the assembler can be hosted in any language and need not be impelemented on the same machine as the experiment control system. Indeed, the same assembler could service several quite different experiment control systems by changing the vocabulary of commands to suit the needs of each. Third, the executable part of a GEPS program has a natural unit of service in the instruction. As is pointed out in the next section, the experiment control system takes advantage of this unit to deal with many of the problems of time-sharing in a real-time environment.

We conclude the discussion of the support system with a description of the interactive debugging facilities. We have mentioned above that the text editor can be used to easily repair bugs once they are found. The other aspect of debugging, tracking down the bugs, is accomplished in a variety of ways. The most common method involves using the GEPS language itself and, in particular, its wide variety of display commands. Special debugging tools include an extensive assembly listing, a snapshot facility, and informative error messages at execution time as well as during assembly. The assembly listing provides maps of both the variable area and the program. The latter map can be used in connection with execution-time error messages to locate the instruction causing the error. The map of the variable area is used in conjunction with the snapshot facility, which allows the user to bring down the experiment control system at any point for inspection of the subject table (see below) and variable area of any subject. Variable-area maps then allow identification of the contents of any variable.

The utilities described above have greatly increased the efficiency of the Illinois system over Kieras' original implementation. Our support system allows the user to prepare, execute, and recover data from an experiment while seated at a single remote terminal. The need for unnecessary card shuffling and for learning assembly language or even FORTRAN has been completely eliminated.

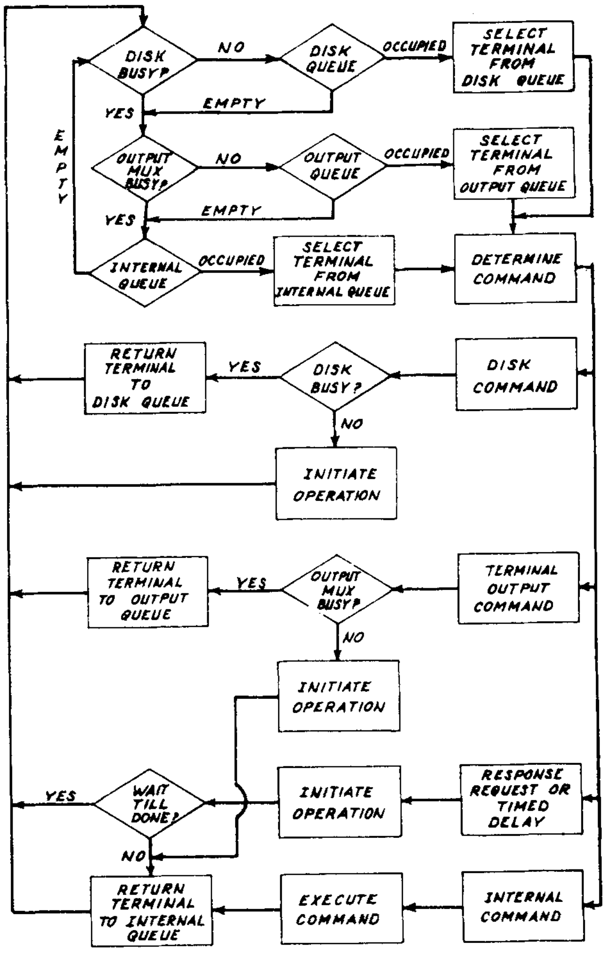

Figure 4. Flowchart of the experiment control system.

The Experiment Control System

The experiment control system has two main parts: an interpreter and a dispatcher. The interpreter executes the experiment at each terminal by interpreting the object programs, and the dispatcher schedules the interpreter among the various terminals. The individual instruction is the unit of scheduling. Thus, on each cycle, the dispatcher selects a terminal, and the interpreter executes a single command for that subject before turning control back to the dispatcher.

Our interpreter is little changed from Kieras' original version. Briefly, the interpreter consists of a set of *command service routines*, each of which executes a particular command, and a set of *operand service routines*, which relieve the command service routines of the burden of locating and fetching operands. Flow control is accomplished by giving each terminal a pointer to the current position in the instruction list. This pointer is updated by the command and operand service routines and is maintained along with other status information in what Kieras calls a *subject table.*

The dispatcher in our system is a great improvement of Kieras' original round-robin system. The heart of the dispatcher, as illustrated in Figure 4, is three queues. Terminals are placed in a disk queue while awaiting a disk operation, in an output queue while awaiting the use of the output multiplexor to the terminals, and in an internal queue while awaiting the interpreter for an operation which does not involve the disk or output multiplexor. Terminals are selected for service from the disk and output queues as the associated devices become nonbusy; terminals are taken from the internal queue so long as both the disk and output multiplexor are either busy or not needed. In practice, all terminals are returned to the internal queue upon completion of each command; the command service routines are responsible for replacing a terminal in the disk or output queue if the requested device is busy. In addition, both Kieras' and our system have a facility for removing a subject from service while awaiting the end of a timed delay, disk operation, or completion of a subject's response; these terminals are returned to the internal queue upon completion of the operation.

This system effectively combines the principle of using the command as a unit of service with a multiple queue system which makes effective use of all I/O wait time. Thus, we have been able to avoid the problems of reentrant programming, and, at the same time, vastly increase the efficiency of the system.

IMPLEMENTATION

In this section we will outline the procedure for implementing GEPS and then conclude with a discussion of the type of facility which could most benefit from such an implementation.

Implementing GEPS can be a relatively painless chore. The experiment control system can be written as soon as basic software is available for interrupt servicing, organizing secondary storage, etc. The extent of such facilities need not be great. Indeed, the experiment control system gains in efficiency as it gets closer to the hardware. In our system, for example, both the clock and output multiplexor are under direct control of the experiment control system, and two levels of system software (for bookkeeping and buffering) intervene between this system and the disk. Terminal input is handled by a single interrupt-service routine which reads, edits, and disposes of input, a character at a time. Thus, our resident system, IBM's TSX, is almost entirely ignored by the experiment control system.

When the experiment control system is working, experimentation can proceed concurrent with development of the support system. During this development, users can translate their own programs into GEPS object code, and special purpose assembly-language programs can be written to serve the other purposes of the support system. The support system can then be built up gradually, starting with the monitor, assembler, and text editor. Utilities for stimulus preparation, data recovery, and debugging can then be added to the system. The editor is often supplied as standard system software for small computers, and, in fact, many aspects of the support system may take more advantage of general system software than does the experiment control system. This differential reliance of the two systems on resident software has important implications for the type of installation best suited for GEPS'

Turning then to those factors which bear on the suitability of GEPS for an installation, we first consider the characteristics related to hardware and then conclude by discussing more general characteristics of the setting.

The ideal hardware configuration for GEPS is a central computer, with a reasonable amount of random access secondary storage, tied to a number of smaller computers which run individual laboratories (cf. Gillman, Lapin, & Buckley, 1975; Scholz & Halff, 1974). The central computer could house the entire support system, leaving the experiment control system appropriate to each lab in its small computer. A minimal configuration for GEPS would be a bare computer interfaced to at least two experiment stations which would run concurrently. It would hardly be worth implementing the system on any less elaborate configuration.

A discussion of the more general characteristics of a setting suitable for GEPS is an appropriate way of summarizing three main characteristics of the system. First, GEPS is a general system and thus requires a certain amount of time and effort to implement. However, if the demands on a lab are such that a general system is appropriate, GEPS should be considered as one of the easier to implement. Second, GEPS demands that the experimental procedure be formulated as a sequence of operations controlling information flow. But if the syntax of GEPS is appropriate, the semantics of the language are largely up to the users since they determine what the actual commands are. Finally, GEPS is dedicated to the idea of software sharing at all levels. At the subject level, different subjects will all be using the GEPS object programs and command service routines; thus GEPS will work best in computers servicing a number of identical stations. At the user level, different users will be using the same vocabulary of commands; thus GEPS will work best when servicing a relatively homogeneous group of users. However, a single support system operating from a central computer could efficiently service a number of heterogeneous experiment control systems in peripheral computers.

In summary, GEPS, as described here, offers an extremely attractive way of implementing a time-sharing system for computer-aided experimentation on a small computer in such a way as to greatly facilitate the process of preparing for, running, and recovering data from experiments.

REFERENCES

Bailey, D. E., & Polson, P. G. Real-time computing in psychology at the University of Colorado. *American Psychologist*, 1975, **30**, 212-218.

Castellan, N. J. Laboratory programming languages and standardization. *Behavior Research Methods & Instrumentation*, 1973, **5**, 249-252.

Castellan, N. J. The modern minicomputer in laboratory automation. *American Psychologist*, 1975, **30**, 205-211.

Churchill, S., Naess, L., & Oliver, W. P. CAN-4, an advanced author language for CAI, computer based testing, and psychological experimentation: PDP-9 implementation. *Behavior Research Methods & Instrumentation*, 1971, **3**, 95-99.

Gillman, C. B., Lapin, D. J., & Buckley, P. B. Hierarchically distributed processing for psychology. *Behavior Research Methods & Instrumentation*, 1974, **6**, 149-154.

Kieras, D. A general experiment programming system for the IBM 1800. *Behavior Research Methods & Instrumentation*, 1973, **5**, 235-239.

Knuth, D. E. *The art of computer programming*, Vol. I, *Fundamental algorithms* (2nd ed.). Reading, Massachusetts: Addison-Wesley, 1973.

Lewis, J. L., Osgood, G. W., & Hebert, J. J. Pleiades: Real-time-sharing control in the behavioral science laboratory. *Behavior Research Methods & Instrumentation*, 1973, **5**, 365-370.

Pilla, M. A. Co-existing with on-line systems. *Behavior Research Methods & Instrumentation*, 1973, **5**, 80-82.

Pitz, G. F. Building a programming language for a small computer: Reinventing the wheel. *Behavior Research Methods & Instrumentation*, 1975, **7**, 42-46.

Polit, A. C. A software environment for problem-oriented systems. *Behavior Research Methods & Instrumentation*, 1974, **6**, 267-273.

Scholz, K. W., & Halff, H. A decentralized computer network for supervision of multiple psychological laboratories. *Behavior Research Methods & Instrumentation*, 1974, **6**, 139-143.

Spear, T. L., Overgard, D., & Christian, T. W. The CLIPR display terminal experiment system. *Behavior Research Methods & Instrumentation*, 1975, **7**, 107-112.

Sternberg, S. High-speed scanning in human memory. *Science*, 1966, **153**, 652-654.

Wood, R. W., Sette, W. F., & Weiss, B. Interfacing the experimenter to the computer: Languages for psychologists. *American Psychologist*, 1975, **30**, 230-238.

(Received for publication April 9, 1975;
revision received June 23, 1975.)