COMPUTER TECHNOLOGY

The PSYCHLAB programming system: A tool for developing experimental control programs

KENNETH J. MAXWELL and ROGER W. SCHVANEVELDT
New Mexico State University, Las Cruces, New Mexico 88003

The PSYCHLAB system is designed to facilitate the task of writing experimental control programs. The system consists of two major components: (1) a collection of procedures and functions to facilitate presenting stimuli, collecting responses, and measuring response times and (2) a high-level language (LAB-TALK) designed to organize and sequence events in experiments. The PSYCHLAB system is formally an interpreter that will interpret programs written in LAB-TALK. The interpreter is customized to perform the detailed functions of each trial in an experiment. The occurrence of these trials is controlled by the LAB-TALK program. Thus, once a particular experimental paradigm has been represented in a customized version of the interpreter, many different experiments in that paradigm can be programmed by variations in the LAB-TALK program.

The PSYCHLAB project is an attempt to simplify and accelerate the process of developing experimental control programs. The PSYCHLAB system provides a common ground on which graduate students and faculty members can implement experiments on Terak 8510/a microcomputers without much duplication of effort among laboratories. The system also represents an attempt to provide a means for nonprogramming researchers to design and implement experiments without having to become very familiar with a complex programming language and the Terak computers. Thus, undergraduate students taking an experimental methods course can use the Terak computers with the PSYCHLAB system to run their independent projects even if they are not skilled programmers.

The PSYCHLAB system takes advantage of the commonalities among experiments. Many experiments are structured around a basic unit called a trial. Although trials differ considerably among experiments, they generally involve presenting stimuli, collecting and scoring responses, and storing data. An experiment commonly consists of several trials organized into blocks, with various characteristics (e.g., stimuli) changing in some way from trial to trial.

Thus, experiments can commonly be analyzed into events at two distinct levels: a micro or trial level that includes the events that take place during a particular

trial, and a macro or experiment level that includes the organization and sequencing of these trials and the control of changing stimuli from trial to trial. The mechanics of controlling the events at either the trial or the experiment level on a computer are quite similar across a wide range of experimental paradigms. The PSYCHLAB system provides a general framework that facilitates the control of these common events.

The system consists of two major components: (1) a collection of procedures and functions to facilitate the precise control of the Terak for specifying trial-level events and (2) a high-level language (LAB-TALK) designed to facilitate the organization and sequencing of experiment-level events. These two components reflect the two levels of structure described above.

To a great extent, the trial and experiment levels are independent of each other. For example, the sequence of events during a particular trial does not constrain the sequencing of the trials themselves. Several different types of trials can be specified for the same experimental structure, and several experimental structures can be constructed for the same trial. The independent specification of these two levels gives the PSYCHLAB system its versatility and power as a laboratory tool.

The PSYCHLAB system is written primarily in UCSD PASCAL, with some of the low-level machine control being accomplished through assembly programs. UCSD PASCAL is a version of the standard PASCAL language developed at the University of California at San Diego.

The PASCAL language is defined by Jensen and Wirth (1974). PASCAL is a structured language. One aspect of that structure is that it is modular. A program consists of a set of procedures and functions and a set of

The authors acknowledge the contribution of Ted Dunning, who was primarily responsible for implementing the PSYCHLAB system, and James McDonald, who designed and programmed the response timing technique used in PSYCHLAB. Requests for reprints should be sent to Roger W. Schvaneveldt, Department of Psychology, Box 3452, New Mexico State University, Las Cruces, New Mexico 88003.

data structures. The procedures are designed to accomplish a subtask and may be written independently from the rest of the program. The independent trial and experiment levels in the PSYCHLAB system are in keeping with the modular nature of structured programming.

The PSYCHLAB system is formally an interpreter that will interpret programs written in LAB-TALK. The interpreter is customized by the user to perform the detailed functions of each trial in an experiment. The occurrence of these trials is controlled by a LAB-TALK program. Thus, once a particular experimental paradigm has been represented in a customized version of the interpreter, many different experiments using that paradigm can be programmed by variations in the LAB-TALK program.

CUSTOMIZING THE INTERPRETER

To implement an experiment using this system, the user must first write a customized version of the interpreter. Familiarity with the PASCAL programming language and the set of PSYCHLAB procedures and functions is a prerequisite. The PSYCHLAB subroutines simplify the specification of common events at the trial level. A subset of the PSYCHLAB subroutines appears in Appendix A. These subroutines are collected in a PASCAL unit that is stored in the PASCAL system library. A unit is a module of code that is compiled separately from the PASCAL program. Units are implemented in UCSD, but not in standard, PASCAL. The PSYCHLAB interpreter is linked to this unit after compilation.

To facilitate the customization process, PSYCHLAB provides a prewritten program (DUMMY) that includes the interpreter and several unspecified user procedures. The user specifies the particular low-level events that take place during a trial in these unspecified procedures. The trial-level events thus become part of the interpreter and are compiled along with the interpreter. The user-specified procedures can be named, using a PSYCHLAB procedure, such that they may be called directly from a LAB-TALK program when they are to be executed.

The primary functions that need to be specified at the trial level include presenting stimuli, collecting and scoring responses, and recording data. To accomplish these tasks, the user needs control of the monitor screen, a timer, the keyboard, and output to a storage medium. Screen control is necessary to present stimuli; it includes both the capability to turn the screen on and off and the capability to write to and erase from the screen. Control of a timer is necessary to precisely control the occurrence of events and to measure response latencies. Control of the keyboard is necessary to provide a response medium for the subject. Control of an external storage medium is necessary to record data.

Trial Specification

The trial block. Trial specification is organized by

two PSYCHLAB procedures that define the beginning and end of a period during each trial when stimuli are presented and responses are required of the subject. The two procedures are TRIAL and ENDTRIAL. PSYCHLAB defines several variables that monitor timing and keypresses during each trial. When TRIAL is executed, these monitor variables are reset or cleared. Also, control of the screen is altered to support the PSYCHLAB subroutines that perform screen functions. Execution of the ENDTRIAL procedure resumes normal screen control. Actual specification of events is done between the execution of the TRIAL and ENDTRIAL procedures.

Presenting stimuli. To present stimuli, the user must have control of the screen and a timer. Execution of the TRIAL procedure allows use of a PSYCHLAB timing variable and screen control procedures. This timer can be used to control stimulus onset, stimulus duration, inter-stimulus interval (ISI), and stimulus onset asynchrony (SOA).

Stimuli are written on the screen with the PUTSTR procedure. PUTSTR requires parameters specifying the screen coordinates and the character string to be displayed. The stimulus may be represented by a variable name. The screen may be turned on and off with a set of procedures that specify whether the stimulus is character or graphic in nature, how long the screen should be turned on, and the portions of the screen to be activated. The Terak screen is divided into three horizontal fields. These may be turned on and off for either graphic or character displays independently.

Writing on the screen and turning on and off the screen are independent control functions: A stimulus may be written on the screen but will not be seen until the screen is turned on; turning off the screen does not erase the stimulus. To erase a stimulus, the system provides a procedure to clear the entire screen of all stimuli. If only a portion of the screen is to be erased, a stimulus consisting of blanks can be written over the original stimulus. Also, the screen is divided into three horizontal fields, and the status of each field (i.e., whether the field is on or off) may be controlled independently.

The timing of events during a trial block is asynchronous. This means that several different things can be ongoing at the same time. Sometimes it is desirable or necessary to have a variety of events occurring, but certain events require that the program wait at a specified point in the program before doing anything else. In this case, it is necessary to specifically instruct the program to wait. Waiting may be accomplished by including a repetition statement that does nothing until the timer exceeds a specified value or until a response is made. PSYCHLAB also provides a SYNC procedure that, when executed, waits for a specified amount of time.

The timing of stimulus durations and intervals during the trial block is controlled by the Terak 60-cycle clock. Thus, the arguments of procedures and functions that

specify the timing of these events have units of 1/60 sec. Events are synchronized with the Terak clock by executing "SYNC(1)," which will pause until the beginning of the next cycle of the clock. This synchronization is performed during execution of the procedure TRIAL.

Collecting responses. The Terak keyboard is used to enter responses. Allowable response keys can be specified by listing and assigning this list to a variable called RESPSET. The remaining keys are disabled during the trial block unless RESPSET is reassigned. Information about responses, such as the number of keypresses, the specific keys pressed, and the times when they were pressed during each trial block, is collected in system-defined variables. The user can use this information to score each response for accuracy and reaction time.

The measurement of response times, unlike that of stimulus presentations, is not determined exclusively by the Terak clock. When a response is entered, a counter accumulates the number of machine cycles between the response and the beginning of the next clock cycle. The duration of this interval is then calculated. Test programs determine the duration of the machine cycles. Millisecond accuracy is accomplished by subtracting the interval duration from the time at the clock cycle following the response.

The response measurements must be stored before the next trial begins, since execution of the TRIAL procedure clears the response variables at the beginning of each trial. The response variables, like the clock, can be reset by the user at any time during the trial with an assignment statement. The purpose of this is to remove information about any extraneous keypresses at unwanted times during the trial.

PSYCHLAB is designed to accept a single character, a string of characters, or a number as a response. String and number responses are collected using specific subroutines designed to control the bookkeeping of those responses. The user needs to specify the type of response by executing the appropriate subroutines when necessary.

Recording data. During an experiment, data may be temporarily stored in variables. However, at some time the data must be written out to an external file so that they may be retrieved after the program has finished executing. This may be done every trial, but it is commonly done either at the end of a block of trials or at the end of the experiment. The OPENDATA function allows the user to create and open a new file on the storage medium. The name of the file is specified in an argument required by the OPENDATA function. A number is returned by this function that is appended to the name of the created file. This allows for separate data files for each subject and enables the user to obtain a subject number for use in counterbalancing conditions.

The data can be written out to the file using standard PASCAL WRITE and WRITELN procedures. The first

argument of these procedures specifies the file to which data are to be written. PSYCHLAB refers to this file as DATAOUT. Once all the data have been recorded, the file may be closed and locked onto the disk using the standard PASCAL CLOSE procedure.

Transportability

Much of the PSYCHLAB system is written in PASCAL and is easily transportable to another machine. Several of the low-level control subroutines, however, are written in assembly code and require rewriting in order to be implemented on another system. These include subroutines needed to define the trial block and to control the screen. The subroutines listed in the first two sections of Appendix A are written in assembly code and require rewriting.

THE LAB-TALK PROGRAMMING LANGUAGE

The LAB-TALK language is designed to facilitate control of experiment-level functions. The primary functions at the experiment level are the sequencing of trials (and other events, such as presenting instructions, presenting feedback, and writing data), passing parameters to the trial-level procedures, and counterbalancing stimuli. These functions are accomplished through the use of a set of intrinsic LAB-TALK subroutines and a set of LAB-TALK operations that may be performed on the intrinsic functions and the trial-level procedures.

All LAB-TALK programs have the same basic structure. The two primary operations are assignment and execution. All LAB-TALK programs end in an execution statement. The word DO is reserved for this instruction. Prior to the DO instruction, the program generally consists of a series of assignment statements. Assignment is used to assign values to parameters or to define a sequence of events. The execution statement causes a sequence of events to be performed.

The LAB-TALK user functions appear in Appendix B, with a short description of each function.

Sequencing Events

Events at the experiment level are entire procedures or functions. The events a user has available come from the LAB-TALK subroutines and the set of user-defined procedures used to customize the interpreter.

In LAB-TALK, a sequence is defined by constructing a vector of the events in that sequence (i.e., the events are simply listed with a space after each of them). Repetition of events in a sequence is accomplished with a repetition operator (*). The end of the sequence is defined by a semicolon (;). The sequence can be assigned to a variable or be defined literally in the execution statement itself. For example, the statement "DO A B C;" causes the sequence A B C to be executed. In this case, A, B, and C may be names of user-defined procedures or LAB-TALK intrinsic functions. These

sequences may become quite long, however, and the sequencing process can be simplified by assigning parts of a sequence to variable names and then sequencing the variables themselves. This practice also facilitates the diagnosis of errors in a sequence definition. For example, the program below will execute the sequence A B C D E F:

```
BLOCK1= A B C D;
BLOCK2= E F;
DO BLOCK1 BLOCK2;
```

Thus, LAB-TALK programs consist of a series of assignment statements followed by an execution statement. Variables such as BLOCK1 need not be formally declared in LAB-TALK; they are defined when they are assigned.

Passing Parameters

Passing parameters is really a cooperative effort between the LAB-TALK program and the user procedures to which the parameters are passed. The LAB-TALK program's function is to set up a parameter structure. This structure contains all of the values of all the parameters needed by a particular procedure. The user procedures at the trial level are able to address this structure via certain PSYCHLAB subroutines in order to choose the values needed at a particular time.

Passing parameters is used in programming as a means of communication between different parts of a program that need to do operations on the same variables or values of variables. Parameters are also a means of producing a more generalized program that may be modified at the upper levels of the program structure. The parameters in LAB-TALK can pass not only the stimuli to be presented on a given trial, but also information about stimulus durations, response intervals, and an assortment of other specifications. This allows high-level modification of the basic trial specification and makes the program more versatile from the viewpoint of the user. The tradeoff is that the user needs to construct and deal with a more complex parameter structure.

Making the interpreter procedures more general by allowing specifics to be passed from the LAB-TALK program is a useful approach to providing a usable system for the novice programmer, since the alternative is to continually modify a very specifically written trial procedure. With the use of parameters, the user can simply specify the information needed in the LAB-TALK program and the program will communicate this information to the user procedures.

All of the user-specified procedures provided with the interpreter are declared with the capacity to accept parameters from the parameter structures constructed in the LAB-TALK program.

In LAB-TALK, the user declares parameters by

listing them, enclosed in parentheses, with the procedure when the procedure is called. This is identical to the way parameters are listed in standard PASCAL. The names given to the parameters have to be assigned to values before the procedure is called. Each parameter may be a single value or a list of values. The values can be assigned directly in the LAB-TALK program itself or be read in from an external file by using the FILE user function. Execution of FILE allows the user to assign an entire list of values to a variable. This is commonly used for stimulus materials that are listed in an external file.

The assignment of parameters in this way establishes a hierarchical parameter structure. The structure includes all of the values of all the parameters for each procedure. The procedures themselves are specified in the interpreter. PSYCHLAB provides the necessary procedures and functions to specify which of the values for each parameter is needed at a particular time in the procedure. The entire parameter structure is generated and read into the Terak before execution of the DO statement. It is not possible to read in sets of stimulus materials between blocks of an experiment using LAB-TALK. Consequently, the number of stimulus materials that can be used in an experiment is limited. If necessary, stimulus materials can be read in between blocks directly through the user procedures.

All of the values assigned to parameters in the LAB-TALK program are of the PSYCHLAB-defined type of VALUE. VALUE is a LISP-type structure implemented in PASCAL. The procedures used to specify which value of a parameter is required also allow the user to change the type of the parameter from VALUE to either STRING or INTEGER.

Counterbalancing Stimuli

Counterbalancing is also a task that requires cooperation between the LAB-TALK program and the trial-level procedures. LAB-TALK user function PERMUTE returns a random sequence of numbers that may be used to randomize stimuli for each subject. The permuted sequence can be assigned to a variable that is passed to a trial-level procedure as a parameter. The trial-level procedure can simply access each random number in turn and use this number as an index for choosing a stimulus from a list. In this way, the stimuli will be chosen randomly.

An Example LAB-TALK Program

Figure 1 presents a representative LAB-TALK program that demonstrates the properties discussed above. The program specifies a training paradigm in which training is terminated when either a performance criterion is exceeded (80% correct) during a particular block of trials or a specified number of blocks (six) is completed. The statement numbers are included only to reference particular lines of the program and are not actually part of the program. Note that a semicolon is used to indicate

```

1.  PERCENT = 80;
2.  NUMBLKS = 6;
3.  NUMTRLS = 10;
4.  ITI = 5;
5.  STIMDUR = 30;
6.  STIMULUS = FILE('STIMULI.TEXT');
7.  LEN = LENGTH(STIMULUS);
8.  ORDER = NUMBLKS * PERMUTE(LEN);
9.  TRIAL = SETUP(STIMDUR,ORDER,STIMULUS) DISPLAY WAIT(ITI);
10. BLOCK = BEEP(2) (NUMTRLS * TRIAL) CHECK(PERCENT) SKIP;
11. DO XFER('INSTRUCT.TEXT')
    (NUMBLKS * BLOCK) ENDSKIP
    XFER('DEBRIEF.TEXT');

```

Figure 1. An example of a LAB-TALK program.

the end of each statement. Braces may be used to insert comments into the program

Statement 9 defines the sequence of events that will be repeated on every trial. Assume that the procedures **SETUP** and **DISPLAY** are trial-level procedures specified earlier by the user in the interpreter, and identified to the LAB-TALK program using **PSYCHLAB** procedures. Assume further that **SETUP** is used to determine the specific stimulus to be used each trial and that **DISPLAY** is used to present that stimulus and collect responses.

The call for procedure **SETUP** in the LAB-TALK program (Figure 1) sets up a hierarchical parameter structure of the type presented in Figure 2. The parameter structure is configured according to the ordered list of parameters listed with **SETUP** when it is called. As a whole, the parameter structure is referred to by the term **ARG**. The second level in the structure reflects the position of each parameter in the parameter list. The values that each of these parameters can assume have been assigned earlier in the program. **STIMDUR** is a constant and is assigned to the value 30 in Statement 5. As mentioned earlier, the timing of stimulus presentations is specified in units of 1/60 sec. Thus, a value of 30 for **STIMDUR** indicates a 500-msec stimulus duration.

STIMULUS is assigned in Statement 6, to the external file "STIMULI.TEXT," using the LAB-TALK subroutine **FILE**. This external file would generally contain a list of stimulus materials. Carriage returns placed in the file define the boundaries of each stimulus. Thus, the variable **STIMULUS** represents the list of stimuli contained in the file to which it is assigned. In this example, **STIMULI.TEXT** contains 10 word stimuli, 3 of which are listed in Figure 2. Access to the individual word is controlled in the **SETUP** procedure using **PSYCHLAB** subroutines that allow the user to specify the coordinates of a desired stimulus in the **ARG** structure. These procedures also enable the user to change the parameter value from type **VALUE** to either

type **INTEGER** or **STRING**. The third level in the **ARG** structure is used to list each of the possible values of the parameters that can take on more than one value, such as **ORDER** and **STIMULUS**.

The values that the parameter **ORDER** can assume are assigned in Statement 8. The LAB-TALK function **PERMUTE** is used to obtain a permuted order of numbers from 1 to the value of its argument. The argument in the example is **LEN**, which is assigned in Statement 7 to the **LENGTH** of the variable **STIMULUS**. The LAB-TALK procedure **LENGTH** returns a value equal to the length of its argument. Since **STIMULUS** consists of 10 elements, **LEN** is equal to 10. **NUMBLKS** is a constant assigned to the value 6 in Statement 2; thus, **ORDER** is assigned to six permutations of the numbers from 1 to 10. These six permutations are concatenated, such that **ORDER** has a length of 60.

In the case of the **STIMDUR** parameter, only one coordinate is necessary, since **STIMDUR** assumes only one value. However, in the case of **ORDER** or **STIMULUS**, two coordinates are necessary to specify a particular value: (1) to indicate the particular parameter, and (2) to indicate which value of that parameter is needed. For example, if the stimulus "BUTTER" was needed, the **PSYCHLAB** procedure used to specify a particular value for the **STIMULUS** parameter would require the coordinates 3,2, indicating the second value of the third parameter.

Since **ORDER** is a permuted list of numbers, it can be used to randomly select stimuli for presentation. The permuted list may be accessed in sequence. In turn, each value of **ORDER** can be used as the second coordinate in specifying a value for **STIMULUS**.

The sequence described in Statement 9 defines a trial as the sequential execution of procedures **SETUP**, **DISPLAY**, and **WAIT**. **WAIT** is used here to specify an intertrial interval (ITI). Note that the stimulus does not have to be passed from the **SETUP** procedure to the **DISPLAY** procedure.

According to Statement 10, a **BLOCK** begins with a tone [**BEEP(2)**], followed by a sequence of 10 trials (**NUMTRLS * TRIAL**). The trials are then followed by the procedure **CHECK**. Assume that **CHECK** is another trial-level procedure defined in the interpreter and is used to evaluate a subject's performance on the preceding trials. The parameter **PERCENT** is passed to procedure **CHECK** and is used to designate a performance

Figure 2. Possible parameter structure for the procedure **SETUP** in Figure 1, including coordinates for each value. The letter "N" represents a random integer between 1 and 10.

level at which the training can end. That is, if performance exceeds the value assigned to PERCENT, then the experiment, or the current phase of the experiment, is terminated. This can be accomplished by setting the PSYCHLAB variable SKIPFLAG to true, in the procedure CHECK, if performance exceeds PERCENT. The LAB-TALK procedure SKIP is executed every block following the procedure CHECK. If the variable SKIPFLAG is true, SKIP will cause the program to ignore everything until the procedure ENDSKIP is reached. At this point, the program resumes normal execution.

Statement 11 is the execution statement. Execution is always the final statement in the program. The LAB-TALK XFER procedure causes the external file identified in its argument to be transferred to the monitor. If performance exceeds PERCENT before the sixth block, the variable SKIPFLAG will be set to true, causing the program to skip to the last XFER procedure and then terminate. If performance never exceeds PERCENT, the program terminates after the sixth block.

MAJOR LIMITATIONS

There are three major limitations to using the PSYCHLAB programming system. The first is that the user must know the PASCAL programming language in order to specify the experimental paradigms. This burden can be relieved by making the paradigms more general (e.g., by allowing specific parameters to be passed to them). Also, over time, the problem should become less acute, as a library of specified experimental paradigms, which the novice programmer can draw upon, is accumulated.

The second problem involves the limitations imposed by the system itself. A primary example is that of space. The code necessary to support the system takes a lot of storage, which limits the amount of code a user can add to the system. This is not much of a problem when the particular experiment being implemented draws heavily on the capabilities of the system, but in the case in which the experiment is using only a fraction of the power of the system, space could become a problem.

This problem is relieved to a degree if the user simply employs the trial-level procedures and functions and disregards the interpreter altogether. This step requires that the user specify the experiment-level events in the PASCAL program. Thus, this step eliminates the possibility of a novice programmer's being able to construct a LAB-TALK program using that experimental paradigm, since the interpreter, which is necessary for LAB-TALK to be understood by the program, is not used.

The moving-window reading task employed by Just, Carpenter, and Woolley (1982) is an example of a paradigm that is difficult to implement using LAB-TALK because of space limitations. This task involves a procedure that presents a passage, one word at a time. Each word is presented in the position it would occupy if the

entire passage were presented on the screen at once. The subject makes a response to display successive words. If LAB-TALK were used to read in the passages, every word in every passage would have to be read in at the beginning of the experiment. This would create a very large parameter structure that probably could not be held in the computer at once.

Third, LAB-TALK is designed to specify experimental paradigms that follow a trial-block format. Some experiments do not follow this format, and consequently, LAB-TALK would be of limited value in implementing them.

COMPARISONS WITH OTHER LABORATORY SYSTEMS

There are a variety of software systems for psychological experimentation that facilitate the use of computers as laboratory tools. These systems differ from each other in scope and intent. Some systems consist of a set of prepackaged experimental control programs (e.g., Fischler, 1980; Keenan & Keller, 1980; Perera, 1981). These programs may be implemented on compatible machines and are useful in setting up an elementary undergraduate methods laboratory. They are also specific and inflexible. Consequently, the systems are limited in scope.

Other systems supply general methods to facilitate users in developing their own experimental control programs (e.g., Burkhardt, 1979; Butler, 1980; Howard, 1981; Kornbrot, 1981; Posner & Osgood, 1980; Post & Fox, 1982). Of these systems, many are exclusively concerned with providing techniques that provide a user easy low-level control of the processor. This control allows for the precise execution of events such as the presentation of stimuli and collection of responses. The user is generally provided a set of subroutines that simplify control functions. The PSYCHLAB subroutines belong in this category.

However, in addition to a set of subroutines for low-level machine control, a number of systems provide the user a framework that is specifically designed for specifying experimental paradigms. This allows the user to work in an environment that is oriented toward an experimenter rather than a programmer. The LAB-TALK language is one such framework. In this way, LAB-TALK is similar to the EXPERIMENT WRITER language (Posner & Osgood, 1980) and the ARTIST system (Kornbrot, 1981).

All three of these systems are alike in that they provide a framework for implementing experiments that consist of discrete trials and blocks of trials. Also, they all provide for users with different levels of experience. The EXPERIMENT WRITER language is different from LAB-TALK and ARTIST in that it is intended to be used to teach students how to investigate empirical questions. The EXPERIMENT WRITER language itself

is not generally used as an aid in developing experimental control programs.

LAB-TALK and EXPERIMENT WRITER are unlike ARTIST in that they emphasize the structural description of an experiment in terms of a vector of objects, with each object representing an event. ARTIST emphasizes a description in terms of experimental design and counterbalancing of stimuli.

The use of ARTIST consists of two phases. The first is a generation phase, during which the display and timing characteristics of the trials are defined. The second is a run phase, during which the data files prepared during the generation phase are read into the computer, information about the particular experiment is provided by the experimenter, and the experiment is executed. These two phases are similar to the construction of a PSYCHLAB program. The generation phase roughly resembles the process of customizing the interpreter, and the run phase is similar to the construction of a LAB-TALK program, in that general information about the experiment is specified.

One significant difference between ARTIST and LAB-TALK is that LAB-TALK can pass parameters from the experiment-level to the trial-level procedures. Thus, the trial specifications can be modified without modification of the trial procedures themselves. ARTIST does not support this capability. Modification of displays and timing during trials must be done by going back to the generation phase. Another difference is that ARTIST is a command-based interactive system, whereas PSYCHLAB is not interactive.

FUTURE ELABORATIONS

Ideally, this system would provide a means of specifying an experiment without any knowledge of programming in PASCAL. The current system attempts to reach this goal. At the experiment level, a laboratory language was developed that is quite simple to use, has very little syntax, and has semantics designed specifically for programs that control experiments. At the trial level, subroutines facilitate control of common tasks. These subroutines attempt to make the specification task easier for the experienced user. The trial level is the bottleneck, for a novice user cannot do the interpreter specification without some knowledge of PASCAL.

The current problem is how to take the step from a PASCAL knowledge-dependent system to a PASCAL knowledge-independent system. One way to approach this problem is to design a system that allows the user to input a specification of what the experiment program is supposed to do. This specification need not be a program but simply a description of the experiment. A system armed with this description and some knowledge about PASCAL and the PSYCHLAB system subroutines could attempt to translate that description into a working PASCAL program.

PSYCHLAB Acquisition

The PSYCHLAB system is available through the PASCAL Users Group. The system includes one 8-in. floppy disk and documentation, and it may be used on Terak 8510/a systems. Orders should be requested from the Terak User's Group, 14151 North 76th Street, Scottsdale, Arizona 85260.

REFERENCES

- BURKHARDT, K. J. The design of a real-time operating system for experimental psychology. *Behavior Research Methods & Instrumentation*, 1979, 11, 507-511.
- BUTLER, F. E. MicroSKED. *Behavior Research Methods & Instrumentation*, 1980, 12, 152-154.
- FISCHLER, I. An on-line laboratory in cognition and perception. *Behavior Research Methods & Instrumentation*, 1980, 12, 116-119.
- HOWARD, J. H. RT-11 FORTRAN subroutines for on-line control. *Behavior Research Methods & Instrumentation*, 1981, 13, 371-373.
- JENSEN, K., & WIRTH, N. *Pascal user manual and report*. New York: Springer-Verlag, 1974.
- JUST, M. A., CARPENTER, P. A., & WOOLLEY, J. D. Paradigms and processes in reading comprehension. *Journal of Experimental Psychology: General*, 1982, 111, 228-238.
- KEENAN, J. M., & KELLER, R. A. Teaching cognitive processes: Software for laboratory instruction in memory and cognition. *Behavior Research Methods & Instrumentation*, 1980, 12, 103-110.
- KORNBROT, D. E. ARTIST: A computer system for creating and running psychological experiments. *Behavior Research Methods & Instrumentation*, 1981, 13, 351-359.
- PERERA, T. B. A laboratory in experimental psychology. *Behavior Research Methods & Instrumentation*, 1981, 13, 195-197.
- POSNER, M. I., & OSGOOD, G. W. Computers in the training of inquiry. *Behavior Research Methods & Instrumentation*, 1980, 12, 87-95.
- POST, T. A., & FOX, J. L. Programming experiments on a TERA 8510/A microcomputer. *Behavior Research Methods & Instrumentation*, 1982, 14, 276-280.

APPENDIX A

PSYCHLAB SYSTEM VARIABLES AND SUBROUTINES

Trial Block Specification

```
TRIAL;  
ENDTRIAL([SCREEN STATE]);
```

These two procedures delimit the beginning and ending of a period during the trial when stimuli are presented and responses are collected. TRIAL resets timing and response bookkeeping variables. Standard read and write procedures cannot be executed within the trial block. The system supports other procedures that perform these functions. SCREEN STATE is an argument that indicates which portions of the screen are to be left turned on after the trial.

Presenting Stimuli

```
PUTSTR(X,Y,STRING);
```

Puts a variable of type STRING at coordinates X,Y on the screen.

```
ALPHA(TIME,[SCREEN STATE]);
```

```
GRAPH(TIME,[SCREEN STATE]);
```

```
TONE(TIME,[TONEBIT]);
```

These three procedures function similarly. ALPHA will cause the specified fields of the screen to be turned on in the character

mode for the specified amount of time. GRAPH will turn the screen on in the graphics mode. TONE will cause a tone to be presented.

ALPHACLEAR;

A procedure that clears the screen of all characters. It cannot be executed during the trial block.

SYNCH(N);

A procedure that causes the program to wait at a particular point for a specified amount of time. The amount of time is specified by the argument N.

Collecting Responses

RESPSET

A variable of type SET that is used to assign the keys the subject may use to respond. The remaining keys are disabled during the trial block.

HOWMANY

A function that returns the number of keypresses during a trial block.

WHICH(N);

WHEN(N);

The WHICH function returns the Nth character that was pressed during the last trial block. The WHEN function returns the time in milliseconds that has elapsed from the beginning of the trial block or the last time the timer was reset, to the Nth keypress.

MAKESTR(STRING);

MAKENUM;

The MAKSTR procedure is used to make a single response of type STRING from several individually entered characters. The MAKENUM function returns an integer composed of several individually entered digits.

Recording Data

OPENDATA('FILENAME');

This function is used to create, name, and open a data file for a particular subject. The function returns a number that is appended to the file name to identify each subject's data. The subject number can also be used in counterbalancing conditions across subjects.

DATAOUT

This is a temporary name given to the data file created by the OPENDATA function. It is used to reference the data file for writing data or to close the data file.

Naming User Procedures

USERDEF('USER PROCEDURE NAME',NUMBER);

This procedure is used to name each of the user-specified procedures in the interpreter so that they may be called directly from the LAB-TALK program.

APPENDIX B

LAB-TALK-DEFINED SUBROUTINES

XFER('FILENAME');

A procedure used to transfer an external text file to the screen and to control the sequence of text screens. Commonly used to present instructions to the subject.

FILE('FILENAME');

A function that allows the user to assign the contents of an external file to a variable. FILE is commonly used to set up a parameter structure.

PERMUTE(N);

A function that returns a randomized sequence of numbers from 1 to N. It can also be used with two arguments to delimit both the lower and upper limits of integers to be permuted.

WAIT(N);

A procedure that causes the program to wait for an amount of time specified by the argument N.

BEEP(N);

A procedure that causes a tone to sound for an amount of time specified by the argument N.

LENGTH(VARIABLE);

A function that returns the number of elements in a variable.

SKIP

ENDSKIP

These procedures make it convenient to set up a criterion in the experiment that determines when certain procedures are to be executed or cease to be executed. This is common in experiments in which a subject's performance level determines when to go on to other procedures. PSYCHLAB provides a boolean variable called SKIPFLAG that may be set to true when a criterion is exceeded. When SKIPFLAG is true, SKIP and ENDSKIP are executed.

(Received for publication July 27, 1982;
revision accepted November 2, 1982.)