Developing a laboratory course using PASCAL on the Apple II

DAVID A. ECKERMAN

University of North Carolina, Chapel Hill, North Carolina 27514

Several resources are available to the person seeking to develop a laboratory course that utilizes microprocessors. Several roles for such equipment are envisioned, and some of the issues regarding types of equipment and programming languages are addressed.

This report is intended as an open letter to psychologists intending to develop a laboratory course for undergraduate students using a microcomputer as course hardware. During the 4 months I have been working on such a task, a number of resources have come to my attention and a number of issues have presented themselves for solution. I should note that I am currently biased toward the Apple II microcomputer and therefore have slanted my coverage of software and other sorts of support toward this device. Although I believe that the Apple is unsurpassed for educational applications, other devices are also excellent, especially the TRS-80, Models I and III.

The University of North Carolina Psychology Department is somewhat less taxed than in the early 1970s by oversubscription of courses. The faculty last year supported a departmental committee's recommendation to reestablish a laboratory course to be required of all majors. Such a laboratory had once accompanied the general psychology course but it was phased out with the growing number of students. The reasons for such an experience in the undergraduate curriculum have continued, however, and we are ready to return to this offering now that the resources are available. Current technology should make it easier to accomplish the goals set for the course, namely, contact with procedures, data, and methods of analysis from a broad range of psychological research topics so that the student will be better prepared to react to research reports.

The current plan is to develop a one-semester (13- to 14-week) 4-h/week laboratory course having as prerequisites the introductory psychology (survey) course and a course in psychological statistics. With approximately 600 majors, the course will serve about 150 students/semester, with approximately nine sections.

This report is based on information obtained in collaboration with R. F. Genovese and R. A. King. This effort was supported in part through NSF Grant SER 78-07046 to the University of North Carolina. Copies may be obtained by writing David A. Eckerman, 122 Davie Hall 013A, University of North Carolina, Chapel Hill, North Carolina 27514.

WHAT CAN THE MICROCOMPUTER DO IN THE COURSE?

The Microcomputer Can Act as the Experimenter

I value my undergraduate laboratory experience, which included running rats, but the cost of maintaining a rat colony and the effort involved in continuing such a venture for 150 sometimes unenthusiastic students must be considered. Our current plan is to maintain a small number of rats and have students do one experiment as a group project with, for example, a flavor (taste) aversion experiment. Although the rat is an excellent subject in physiological psychology and in conditioning and learning experiments, the human may be a better subject for a laboratory in general psychology. A common practice is to have one student as subject and the other as experimenter. Another approach is to use a microcomputer interacting with the subject. If the microcomputer presents the instructions, arranges the experimental conditions, and carries them out, the student can assume the role of subject and then interpret the meaning of the data. Obviously, the topics can vary across a wide range of psychological phenomena.

In the approach described above, the students produce and then analyze the data. There is, of course, an alternative, namely, simulation. Simulations proceed from a model of a situation. The program then generates data consistent with the model. The student's job is to uncover the relations the data reveal about the model and hence, hopefully, about the phenomena being modeled. The rationale is that the time-consuming task of data collection can be abbreviated by simulation, allowing the student to find relations among variables and to test models of relationships. EXPER SIM (see Forbach, 1979) and LABSIM (Eamon, 1980) are two examples of general programs allowing simulation experiments. CONDUIT provides several experiments that use the EXPER SIM program. Are there microcomputer versions of these programs? Levy and Durnin (1981) may have an answer to this question. The extant versions of EXPER SIM use more space and speed than micros currently have to offer, but, with the addition of a modem and a telephone connection, microcomputers make excellent smart terminals on larger computers. In

a sense, the larger programs can execute on microcomputers too if the telephone lines stay "up" and the large computer stays on the line. Although these connections make the laboratory exercise more risky, contact with the large programs may make the risk worthwhile for some exercises.

The Microcomputer Can Act as Tutor

We hope to develop programs that show the student the data and then act as tutors as the student selects and uses a method of analysis. There is much work to be done in developing these programs.

Another tutorial function might aid the student in developing other experimental designs from the exercise just completed. Following an experiment on reaction time, for example, the device could provide a CRT message noting that reaction time is also a function of x , y , and z , three variables that could also be manipulated with the microcomputer. The student could be encouraged to select a variable to manipulate in a further exercise and to propose how to go about this additional investigation. Additional data could be obtained and added to the data for the student's report.

Students in the course will learn very little about the specific microcomputer operations. They will learn how to turn the system on, how to start a program, whom to call when troubles arise, and little else. This limitation derives from the role the course plays in the curriculum. The course is intended for all majors, not only those who will go on to become graduate student researchers. For undergraduates who wish to learn about the device (and for the laboratory instructors), it is important to allow out-of-class experience in programming, file handling, and so on.

The Microcomputer Can Proctor and Test

The student's work is evaluated primarily through laboratory reports. At first, the outline and many of the words are prompted; within one or two reports, the student produces the report with few prompts. The microcomputer is not very useful in evaluating these reports, although it is a competent grader of multiple-choice items.

For exercises requiring preliminary reading, there is a brief qualifying quiz. The student answers questions at the keyboard and, if criterion performance is achieved, is advanced to the experiment itself. If criterion performance is not achieved, another session is scheduled. It will be possible to note for the student which sections of the preliminary material should be reviewed prior to the next session. This use of the computer is much like the role played by a proctor in the personalized system of instruction (PSI) method developed by F. S. Keller and others. This reference should foreshadow later proposals. The microcomputer's capabilities as a stimulus-response device with fast feedback and excellent data storage is very handy.

The Microcomputer Can Serve as a Course Manager

Student attendance, progress, and grades can be logged and analyzed.

MICROCOMPUTERS: AVAILABILITY AND RELIABILITY

There are several very fine microcomputers on the market. The devices are fast and storage is large, but they are not limitless allies. For example, microcomputers are not as capable as the minicomputers that have been relatively common in psychology laboratories. Speed and facility in dealing with external devices are the major differences. However, many microcomputers can be purchased for the cost of a single mini. Rather than play the complex "executive program" game of timesharing, it is much easier to have several microcomputers. Also, the delivery time of microcomputers is typically short compared with that of minicomputers. Minicomputers typically require special installation and environments; micros do not. Microcomputer repair is simpler, maintenance help is easier to obtain, and repair costs are lower. Only one of three Apple II systems with which I am familiar has developed any problem over 3-18 months of service; two North Star Horizon IIs and two TRS-80s I have seen in operation have also had minimal service needs. Repair charges have totaled \$27 for 93 months of operation of all of these systems.

HOW CAN I MAKE THE DEVICE DO WHAT I WANT?

Software is Available Commercially

Much of the available software is advertised and reviewed in the "trade publications" that can be found at many newstands and local computer shops. Common publications include *Byte*, *Creative Computing*, *Dr. Dobbs's Journal (of Computer Calisthenics & Orthodontia)*, *Interface Age*, *Kilobaud Microcomputing*, *Nibble*, *Personal Computing*, and *80 Microcomputing*. Also, catalogs often review software for particular functions. Kantrowitz and Kantrowitz (1980), for example, list educational software available for several microcomputers and are starting a monthly newsletter titled *Microcomputers in Education*. Many of these programs cost from \$10 to \$50. Occasionally, the asking price is above \$200-\$300 (Hewitt, Note 1). In some cases, membership in a users' group allows very useful software to be purchased at very reduced rates. For example, I am in the Carolina Apple Core and have obtained several utility programs (as well as a few games) for approximately \$8. Some software for educational uses is being provided by CONDUIT, a group at Iowa University that sets standards for educational software and distributes many programs at very reasonable costs (see a recent issue of *Pipeline*). CONDUIT's history is with mini- and main-frame computers, but it is now starting to provide software for the Apple II and the

TRS-80 as well. Programs are in a fairly "dialect-free" BASIC language. The cognitive and perception experiments by Levy (CONDUIT No. PSY224; see Levy, 1980) are excellent. I understand that other packages in cognition (Keenan & Keller, Note 2) and in perception (Cavanagh & Antis, 1980) are being developed through CONDUIT for the Apple II. Also I understand that the series of programs developed at the University of Oregon (Posner & Osgood, 1980) is being rewritten in PASCAL with intrinsics especially oriented toward the Apple II. A recent brochure from Shapiro (1980) describes another series of experimental procedures arranged for the Apple II. TRS-80 owners can obtain the series of experimental modules authored by Perera (1981). As Wood (1980) notes, several of the commonly available computer "games" are really psychology laboratory exercises, and we can incorporate them into coursework without too much trouble. A word of caution applies to all commercial software, of course. Regardless of price, software must be thoroughly checked before it can be trusted.

Software is Available "For the Typing"

Behavior Research Methods & Instrumentation, as well as the microcomputer trade journals and other sources of microcomputer information, such as users' group newsletters, provides a wealth of published programs. Such programs are often, of course, in a language that is not quite like the one on your machine, although if you have one of the best-selling models, this problem is reduced. Even if they are in another dialect of your language, though, translation may be possible, or at least the algorithm may be adapted. There are also frequent extensions and/or "fix-ups" noted in these publications that may solve a problem created by a limitation of the software you already have. Programs by Bewley (1979), available through CONDUIT, can be used with microcomputers. Several interesting exercises in cognition are contained in this series of procedures.

You Can Write Your Own Software

Suppose you decide that a particular experimental protocol is ideal to fit into the laboratory course. You will be in the best position to see that the protocol is correctly implemented on the microcomputer. How might you proceed? You can write the program in assembly code that is specific to your microcomputer, or you can write the program in a higher level language. Assembly code executes quickly and can put an efficient programmer in complete charge of the order in which the device does the required steps. Despite the experience of some programmers who find that the higher level languages "write bad code" (Frey, Note 3), I find that the higher level languages execute quickly enough to get a great many things done. However, the time to complete the job includes the time to write and debug the program (as well as, for those who are devoted to big systems, the time to correct job control language errors,

find the system up, and walk to and from the place where input and output occur). There are two higher level languages with which I have experience: BASIC and UCSD PASCAL. There are alternatives to these (FORTH, COBOL, FORTRAN, . . .), but only BASIC and PASCAL will be discussed here.

BASIC is an interactive language that executes code either in a direct "command mode" just after you enter the instruction into the device (useful for doing small calculations) or in a "deferred execution mode" when you enter the instruction RUN. The language is similar in some ways to FORTRAN (it has been called "FORTRAN without tears"). BASIC is easy to learn, and many implementations of the language are very "friendly" (e.g., an error message is given when you try to enter a line of code having a syntax error, and the status of variables can be determined from command mode after a run terminates through error or other means). If you do not know any programming language, becoming proficient in BASIC should take about 1 week's work. Of course, depending on your propensities, it might be a vigorous week. There are a number of good tutorial manuals for learning BASIC (see supplier of your microcomputer). It is easy to sit down and spontaneously generate a program in BASIC to carry out some task. The language is permissive: It does not require that the user declare or define many variables, and it lets the user do things in most any order (dimensioning arrays in midprogram, wending back and forth with "GOTO" statements). These are nice features in many applications.

Although BASIC has many nice features, it has some clear limits as well. First, in most implementations, variable names are really only two characters long. The BASIC interpreter looks only at the first two characters in identifying the variable, even though you may be able to use longer identifiers as you type in the program. Thus, "response" and "reinforcer" cannot be used to indicate two different variables. To the device, they are both RExxx. Some implementations (e.g., North Star BASIC) are not even this flexible, insisting that variable names be a letter and a number and no more. Another limitation is speed. Because BASIC interprets the program each time it is run, there are several extra machine steps in each program step. A "compiled" language does this translation once, and, when executed, the program need not be retranslated from the higher level language to machine code. Evidently, BASIC compilers will soon be available and will allow the BASIC program to be compiled into machine code and thus executed more quickly. This limitation will then be eliminated. It will be worthwhile, however, to determine just how much faster the compiled program will run. I understand that current compilers do not improve the speed as much as hoped.

Perhaps the greatest limit of BASIC is psychological rather than mechanical or electronic. By encouraging extemporizing and by permitting meandering through

the program, it is very possible to produce a program that is error prone and impossible to figure out later. Large programs in BASIC can become nightmares of intertwined conditional "GOTO LINE #X" statements, with variables named according to obscure and forgotten conventions. These are the issues that PASCAL seeks to solve. Also, because there are so many different dialects of BASIC, the "transportation" of a working program from one type of machine to another is very difficult. That creates a problem in the present tense, since the perfect program for an application may appear in Dialect X while you have Dialect Y. It is a greater problem, however, when one considers the speed with which things are changing in this field. As technology continues to develop, cheaper and more powerful devices will appear that will be ideal for your application. Who knows, however, which dialect of BASIC will be provided? You may have to start software development over. CONDUIT has sought to establish standards for BASIC programs that should minimize this problem (see Peters & Johnson, 1978). If the influence of CONDUIT were as great as deserved, there would most likely be no real problem. However, there is a real possibility that you will suffer from having to translate between BASICs. I understand that CONDUIT may also support PASCAL programs in the future, partly because of this issue.

PASCAL is a relatively recent addition to the higher level language world. It was designed to solve problems that users chafed under in other languages. It is said to be the "best statement of what structured programming is all about" (Bowles, 1980). There are reasonably good teaching materials for the language: Bowles (1977, 1980), Findlay and Watt (1978), Gorgono (1978), as well as others. Instead of letting the programmer extemporize and meander, PASCAL insists that variables be declared at the start of the program. Variable names can have eight defining characters (and the language retains any additional characters beyond the eight), so that you are encouraged to use variable names that indicate the meaning of that variable. Further, types of variables other than the common ones of Boolean, integer, real, and character (or string) may be declared and thereafter referred to in shorthand. Thus complex records can be addressed simply in the program text. PASCAL is very "building-block" oriented, encouraging the programmer to develop small routines (procedures, functions) to accomplish parts of the task. By clearly separating global (defined throughout the program) and local (to a procedure or function) variables and by allowing the same "name" to apply to a global and a local variable (the language keeps track of which variable is being addressed), PASCAL encourages the programmer to build routines that can be carried intact from program to program by linking them into the new program or by placing them in a library of routines available to any program. Testing each building block before entering it into the main program reduces errors in the long run.

While insisting that the terms of a program be entered in a particular order, PASCAL makes this formatting of the program text relatively easy. No line numbers are needed, as they are in BASIC. Instead, the text can be put on as many or as few lines with as much or as little indentation as the programmer desires. Statements are separated by semicolons; spaces or carriage returns used in formatting the program are ignored by the compiler. In most implementations of the language, programs are entered using a text editor that comes as part of the PASCAL system. This text editor allows one to find, replace, insert, delete, copy, and so on, text at any place in a text file (e.g., a program). This same text editor is useful in text processing. While PASCAL insists that things occur in a particular order in the final program it eases the task of arranging things in that order. There are a number of characteristics that make PASCAL a very desirable language. The grandest hope for PASCAL is, however, something about which we will not have proof for several years: An effort is being made to assure that implementations of the language will continue to minimize differences from the original. There are two or three dialects of PASCAL currently available (UCSD PASCAL, PASCALM, TINY PASCAL, "standard" PASCAL). There are hopes of reducing the differences between these implementations and of restricting the creation of new versions. This effort has transportability as its goal. Transportability will be gained, of course, partly through resisting the temptation to add new twists to the language. Some accommodations will, hopefully, be made when sufficiently striking new additions are suggested. What separates PASCAL from most other languages is that there seems to be a sincere effort to make rational judgments on these issues. Time will tell. There are also some limitations to the language when compared with, for example, FORTRAN. Variables cannot be dynamically dimensioned. Although "pointers" may be used to build arrays as you go or to store sparsely filled matrices compactly, programming using pointers is a complex affair. The language requires quite a bit of free memory (e.g., 24K 16-bit words) to operate efficiently. There is no shorthand way of exponentiation. Disk files must be blocked rather than being strung among the free blocks on the disk. These limits are not severe, however, when contrasted with the clarity and ease of writing the PASCAL programs. One special advantage of this clarity for the laboratory course will be the ease of communicating with colleagues about what the protocol in fact accomplishes and the ease of adjusting the programs if or when other variations are later desired.

Some references are especially useful to the person seeking to write software for the laboratory course: Lincoln and Lane (1980), Reed (1979), and Thompson (1979). The fact that all these articles are from *Behavior Research Methods & Instrumentation* represents a recommendation to the reader that further help is very likely to be found in the pages published in this journal.

REFERENCE NOTES

1. Hewitt, J. *Advanced stat pak for Apple computer*. Unpublished manuscript, Psychology Department, University of Missouri at Kansas City, 1980.
2. Keenan, J. M., & Keller, R. A. Computer lab in memory and cognition: Instructor's manual. Unpublished manuscript, 1980.
3. Frey, P. Personal communication, 1980.

REFERENCES

- BEWLEY, W. L. Cognitive psychology: A computer oriented laboratory manual (instructor's manual, student manual). Wentworth, N.H: COMPress, Inc., 1979.
- BOWLES, K. L. Microcomputer problem solving using PASCAL. New York: Springer Verlag, 1977.
- BOWLES, K. L. Beginner's guide for the UCSD PASCAL system. New York: McGraw-Hill, 1980.
- CAVANAGH, P., & ANSTIS, S. Visual psychophysics on the Apple II: Getting started. *Behavior Research Methods & Instrumentation*, 1980, 12, 614-626.
- EAMON, D. B. LABSIM: A data-driven simulation program for instruction in research design and statistics. *Behavior Research Methods & Instrumentation*, 1980, 12, 160-164.
- FINDLAY, W., & WATT, D. A. PASCAL: An introduction to methodical programming. Potomac, Md: Computer Science Press, 1978.
- FORBACH, G. B. EXPER SIM: Review and update. *Behavior Research Methods & Instrumentation*, 1979, 11, 519-522.
- GORGONO, P. *Programming in PASCAL*. Reading, Mass: Addison-Wesley, 1978.
- KANTROWITZ, M., & KANTROWITZ, B. E. *Queue: Educational software for Apple II, Atari, Pet, and TRS-80: Catalogue 3*. Fairfield, Conn: Author, 1980.
- LEVY, C. M. Getting an on-line departmental teaching laboratory on-line. *Behavior Research Methods & Instrumentation*, 1980, 12, 111-113.
- LEVY, C. M., & DURNIN, M. W. Modeling on microcomputers. *Behavior Research Methods & Instrumentation*, 1981, 13, 201-203.
- LINCOLN, C. E., & LANE, D. M. Reaction time measurement errors resulting from the use of CRT displays. *Behavior Research Methods & Instrumentation*, 1980, 12, 55-57.
- PERERA, T. B. A laboratory in experimental psychology using the TRS-80. *Behavior Research Methods & Instrumentation*, 1981, 13, 195-197.
- PETERS, H. J., & JOHNSON, J. W. *Conduit author's guide*. Iowa City, Iowa: CONDUIT, 1978.
- POSNER, M. I., & OSGOOD, G. W. Computers in the training of inquiry. *Behavior Research Methods & Instrumentation*, 1980, 12, 87-95.
- REED, A. V. Microcomputer display timing: Problems and solutions. *Behavior Research Methods & Instrumentation*, 1979, 11, 572-576.
- SHAPIRO, R. *The cognitive testing station*. Edgecombe, Maine: DIGITRY, 1980.
- THOMPSON, G. C. Behavioral programming with the APPLE II microcomputer. *Behavior Research Methods & Instrumentation*, 1979, 11, 585-588.
- WOOD, L. E. An "intelligent" program to teach logical thinking skills. *Behavior Research Methods & Instrumentation*, 1980, 12, 256-258.