IPEX1, A LIBRARY OF DYNAMIC INTRODUCTORY PROGRAMMING EXAMPLES

Linda Lewis
Department of Computer Science
Southeast Missouri State University
Cape Girardeau, Missouri 63701

Robert J. McGlinn
Department of Computer Science
Southern Illinois University
Carbondale, Illinois 62901

This paper describes IPEX1, a library of dynamic Pascal examples for use in an introductory Pascal course. Several modules of examples are designed to help students overcome conceptual difficulties. Additionally, the examples should direct students toward a better understanding of the constructs available in Pascal. Finally, visualizing the execution of the examples should give the students a better feel for what goes on inside the computer as a program executes.

Keywords: computer aided instruction, computer science education, Pascal.

INTRODUCTION

Most computer science instructors use the textbook method to teach programming. Also called the static approach by Ross [8], this method challenges students to read static examples in texts in order to learn about the constructs of a programming language. However, there are two very separate aspects to every program - static and dynamic. As Ross points out,

> The novice programmer must understand and be able to differentiate between these two facets clearly in order to learn programming. Presentation of new programming concepts in a textbook requires the use of examples which are printed in static form. An attempt must then be made to explain in writing what happens when the example is executed on a computer. This is where the textbook fails; explaining something which is inherently dynamic in static textbook form is only marginally helpful [8].

The current solution involves having the instructor trace the execution on an overhead projector or on the blackboard. Of course, students take notes, but again, they have only a static representation of dynamic concepts.

A better approach would be the use of a teaching aid which allows a student to view the execution of a program through a truly dynamic interface - the computer itself. Students should be able to proceed at their own pace, noticing the effects of variable declarations, conditional branching, looping, and other operations. Such a tool should also allow fast and easy repetition of the execution of a given program. This would eliminate the student's need to take notes during the trace of a program's execution, thereby breaking the chain of having a static representation of dynamic concepts.

There are many systems in existence which present lessons dynamically through the computer. Access to lessons through such systems as the well known PLATO system can be rather expensive, and their use often requires special equipment [2]. Others are intended as course supplements, and include lesson plans and a small number of examples in an effort to repeat everything the instructor has presented [1]. However, this type of educational software adds little to what students already have in their notes and textbooks. Still other systems consist of interpreters, mainly for use as graphic support of software development [3,4,7]. Unfortunately, students can make effective use of these systems only after they have acquired the ability to write correct programs.

These deficiencies point to the need for a low cost system that includes a library of correct programs and allows the beginning student to actually see the internal effects of program execution. For example, such a system could be used to view the effects of a new construct before students attempt to use it in their programs. Also, students experiencing difficulty in understanding the flow of control with various decision and looping constructs could benefit from using this type of teaching aid.

It is for these reasons that we have chosen to design and implement a library of dynamic introductory Pascal examples (IPEX1) for use by beginning students in computer science.

© 1985 ACM 0-89791-152-0/85/003/0072 $00.75

PEDAGOGICAL CONSIDERATIONS

The library is divided into seven modules, which are arranged in the order in which the statement constructs are typically presented in an introductory Pascal course. The modules become increasingly more difficult as the student progresses from one to the next. Additionally, within each module, the examples progress in difficulty.

Having taught computer science courses, and in particular introductory courses, for several years, we have become aware of many conceptual difficulties which beginning students often experience. We have attempted, through the use of well chosen examples, to address these difficulties. A brief description of the modules, including the pedagogical considerations which entered into their design, follows:

Module 1 - Input Versus Assignment Statements

Early in an introductory course, many students seem unable to cope with the concept of a variable. They are unable to conceptualize a variable as a location in the computer's memory which stores a value. Additionally, recognizing how and when the value of a variable changes causes much difficulty.

Interestingly, we have observed an anomaly in several students' learning behavior. Early in an introductory course many students do not readily distinguish between using an input statement and using an assignment statement to change the value of a variable. If the input statement is introduced early, then students seem to think that every program must have such a statement. Thereafter, they are uncomfortable with examples that use only assignment statements. This difficulty is so deeply rooted that several students never overcome it.

The examples in the first module are designed to assist the student in overcoming these problems. Each example is intended to visually demonstrate the effects that assignment and input statements have on variables. By repeatedly witnessing these effects, we feel that students will become more comfortable with the notion of a variable.

Module 2 - Input and Output Statements

The major goal of this module is to further acquaint the students with the differences between READ and READLN and between WRITE and WRITELN. We do, of course, recognize that the EOLN and EOF functions do cause students some concern and we have developed examples illustrating the behavior of these functions. However, by necessity, they are presented in a later module on WHILE loops.

Module 3 - Decision Statements

This module is designed to help the students learn the mechanics of the various decision statements (IF-THEN, IF-THEN-ELSE, and CASE).

In this module, as well as in the others, the student should gain an appreciation for the value of tracing the execution of a program. Indeed, the execution of each example is visually traced on the monitor as the computer single steps through it. Since the student can delay the execution of the next instruction as long as s/he wishes the student has ample time to predict the result produced by each statement. Thus, the student can use this as a test of his/her understanding of the concepts involved.

Module 4 - WHILE Loops

In this module we demonstrate some situations where it is most natural to use WHILE loops. In particular, we illustrate the EOF and EOLN logic.

Additionally, we present the two troublesome pathological situations which arise when working with WHILE loops, a loop in which the body never executes and, of course, an infinite loop. Students refuse to believe they can write an infinite loop. We show them how easily it can be done.

Module 5 - REPEAT-UNTIL Loops

Situations which lend themselves naturally to solutions using REPEAT-UNTIL loops are presented.

We feel that significant parts of the learning process are reading and understanding well written solutions to meaningful problems. And so, rather than use obscure examples to illustrate Pascal constructs, we frame the constructs in solutions to typical programming problems.

Module 6 - FOR Loops

One of the topics that causes the introductory students a good deal of conceptual difficulties is array manipulation. Since FOR loops and arrays blend together nicely, a significant portion of this module is devoted to arrays. In particular, the last example (determining election results) does a good job of illustrating the use of two-dimensional arrays.

Module 7 - PROCEDUREs and FUNCTIONs

Undoubtedly, the hardest topic for the beginner is the manipulation of procedures and functions. There are many sources of problems (e.g., transfer of control to and from the subprogram, argument-parameter correspondence, and VAR versus value parameters). The early examples in this module address these topics. The latter examples serve as a review of some important algorithms which the introductory student should be comfortable with before going on to a second course: the linear search, the binary search, the selection sort, and the insertion sort.

OPERATION OF THE SYSTEM

The hardware and software required to run our software are minimal. An IBM-PC was chosen as the computer on which we developed our software with TURBO Pascal 2.0 [9] used as the implementation language. However, the TURBO Pascal system is not required to run our software since all of our code is compiled into machine language and Borland Inc., the company which developed and which markets TURBO, does not require a royalty for programs developed with their system. The machine code occupies 196K of storage on diskette. Hence, a double-sided, double-density disk drive is required. However, since we use the "chain-and-execute" feature of TURBO Pascal, only 41K of main memory is needed to run our software.

Additionally, a printer is recommended, but it is not required. A window in the upper right hand corner of the screen is used to display the program which is "executing". If an example exceeds the size of the window, a listing of the program is offered prior to execution. The student can then use this listing to more easily follow the trace of execution. Our system is designed to detect the availability of a printer, and no listings are offered if the printer is not accessible (i.e., there must be a parallel printer interface card installed, the printer must be turned on, and there must be paper loaded in the printer).

The user does not need experience with either the IBM-PC or its operating system. The diskette is configured so that our software boots automatically from drive A when the computer is powered on (assuming, of course, that the diskette has been inserted into drive A).

Furthermore, many qualities desired in educational software in general were taken into consideration during the design and development of the software. Instructors normally ask the following questions when selecting software:

1. Is it accident proof?
2. Are the directions easy and simple to understand?
3. Are the instructions presented through the computer interactively (no manual to read)?
4. Is help provided following an incorrect response? [5]

Indeed, since every response is checked for its validity, it is not possible for the student to "crash" the system by entering an incorrect response. Of course, if an incorrect response is entered, the error is explained and the user is gently asked to reenter it. The entire system is menu-driven and the directions are presented in a clear and straightforward manner. There is no manual to read; after booting the diskette, the main menu appears and the system is up and running. Finally, suggestions from students to "avoid technical jargon, and stay at our level" [6] were also kept in mind, and the final result, we believe, is a user-friendly, easy-to-use package.

A TYPICAL SESSION

We now demonstrate what the user experiences as s/he works through the menus and examples of our software.

After the system is booted the following menu appears on the screen:

```
          Library of Dynamic Pascal Examples
       Written by Linda Lewis and Bob McGlinn
                     SIU-C, 1984

                        MENU
      0 Exit (quit)
      1 General tips for using this system
      2 Input versus Assignment Statements
      3 Input and Output Statements
      4 Decision Statements
      5 WHILE loops (conditioned controlled)
      6 REPEAT-UNTIL loops
      7 FOR loops (count controlled)
      8 Procedures and functions

   Note:  Each group contains its own introduc-
          tion.

Type item number, and press return.
```

There are nine valid responses. Entering a 0 terminates the execution of the system. Entering a 1 leads to an overall introduction to the use of the system and to some helpful hints on how to make effective use of the system. For example, screen layout is explained, instructions for going through the examples are given, and if a printer is available, a method for obtaining a listing of any screen is described. Of course, each of the remaining choices corresponds to one of the seven modules.

Having selected one of the seven modules, a menu of the examples in the chosen module appears. This menu includes a list of example descriptions as well as options which allow the user to view the introduction for the module or return to the main menu. For example, if the student selects the module dealing with WHILE loops by entering a 5, then the following menu appears on the screen:

```
          WHILE loops (condition controlled)

                       MENU
          0 Return to main menu
          1 Introduction and general hints
          2 Print characters until trailer 'Z' is
            reached
          3 Print even numbers (infinite loop)
          4 Given your input of up to 3 character
            strings, use EOLN and EOF to print them
          5 Given your input of up to 5 integers,
            print their squareroots (using sentinel
            of 0)
          6 Determine the final balance of a bank
            statement

Type item number, and press return.
```

If, at this point, the student selects an example (2-6), the screen is cleared and, if the chosen example is one of the more difficult ones, then a slightly more detailed description of the logic involved in the example is displayed on the screen. For example, if the student selects the last WHILE example (selection 6) which deals with balancing a bank statement, then the following more detailed explanation is presented:

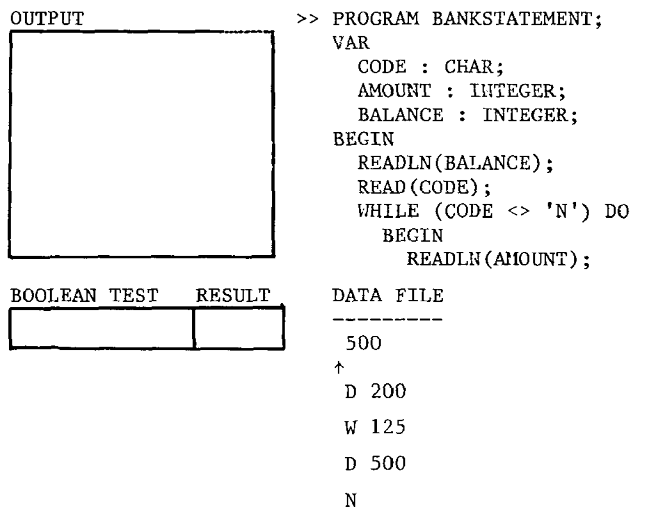
This example determines a final balance for a bank statement. The beginning balance is read first, and each following line of data contains a code letter and a dollar amount. A code of D indicates a deposit and W indicates a withdrawal. The final line contains only a code letter, N, to indicate the end of data.
Press any key to continue.
This particular example is too large to fit on the screen completely, and will be scrolled up as the line pointer advances. To view it as a whole, would you like a listing? (y/n)

This particular example is too long to fit in the window which is devoted to the display of the program, and so if a printer is available, a listing is offered. Most of the rest of the discussion in this section centers around this banking example.

It should be pointed out that most examples allow the students' to enter their own data thereby allowing the program to be run several times using different data values.

In any case, when all preliminary preparations are complete, the screen is initialized and the trace of the example begins. The screen is initialized as in Figure 1 for the banking example.

The static versus dynamic aspects are further emphasized by dividing the screen approximately in half, with the right portion being used for the static aspects (the program itself and the data file), and the left portion for the dynamic effects (e.g., the variables and output).



```
OUTPUT                  >> PROGRAM BANKSTATEMENT;
                           VAR
                             CODE : CHAR;
                             AMOUNT : INTEGER;
                             BALANCE : INTEGER;
                           BEGIN
                             READLN(BALANCE);
                             READ(CODE);
                             WHILE (CODE <> 'N') DO
                               BEGIN
                                 READLN(AMOUNT);
BOOLEAN TEST   RESULT      DATA FILE
                           ---------
                            500
                           ↑
                            D 200
                            W 125
                            D 500
                            N
```

Whenever printing stops, press any key to continue.

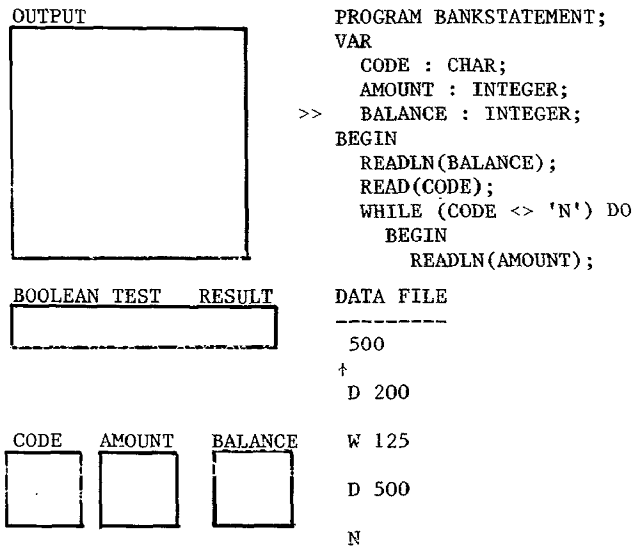
Figure 1. Screen Initialization

Two pointers are shown in a different color than that used for the listings of the program and the data file. The program pointer (>>) indicates which statement is currently "executing", while the data pointer (↑) specifies the next value to be read from the data file.

End-of-line and end-of-file characters are not shown so that the data file appears visually the same as it did when it was created, thus preserving its static nature.

The dynamic effects of execution appear on the left portion of the screen. A window for output is defined, as well as a comparator for those examples using boolean expressions in WHILE loops, REPEAT-UNTIL loops, and decision statements.

During the trace of the example, the system pauses after each statement is simulated (i.e., the program pointer remains fixed on the statement). Pressing return or any other key advances the pointer to the next line and it is then simulated. As each variable in the VAR section is encountered, a labeled storage box is created in the dynamic area of the screen to illustrate the allocation of the corresponding memory location. Figure 2 approximates the appearance of the screen after the three variables in the banking example are "declared".

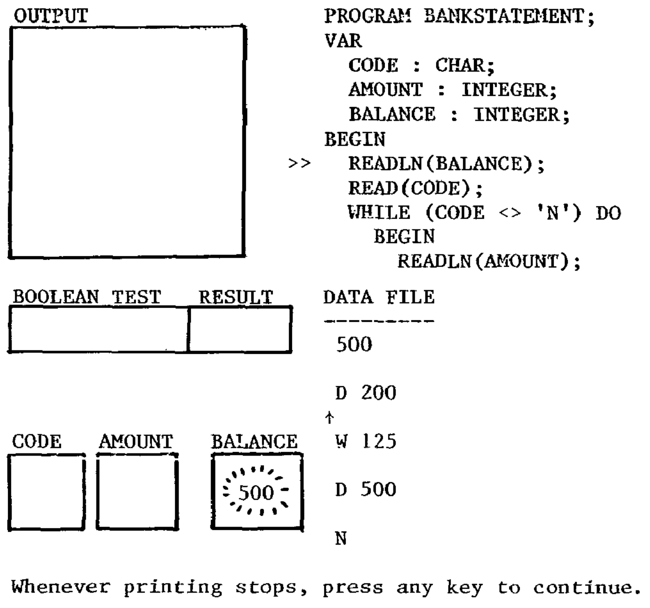


```
OUTPUT                         PROGRAM BANKSTATEMENT;
                               VAR
                                 CODE : CHAR;
                                 AMOUNT : INTEGER;
                            >>   BALANCE : INTEGER;
                               BEGIN
                                 READLN(BALANCE);
                                 READ(CODE);
                                 WHILE (CODE <> 'N') DO
                                   BEGIN
                                     READLN(AMOUNT);
BOOLEAN TEST    RESULT         DATA FILE
                               ---------
                                500
                               ↑
                                D 200

CODE    AMOUNT    BALANCE       W 125

                                D 500

                                N
```

Whenever printing stops, press any key to continue.

Figure 2. Variable Declarations

The "execution" of the initial READLN statement causes the initial balance of 500 (cents) to appear in the storage box labeled "BALANCE". In order to call attention to this change in the value of the variable, its new value is displayed in a color different from that used for the rest of the dynamic display. Additionally, the new value blinks briefly. Of course, the data pointer is advanced to the next line. All of these changes are illustrated in Figure 3.

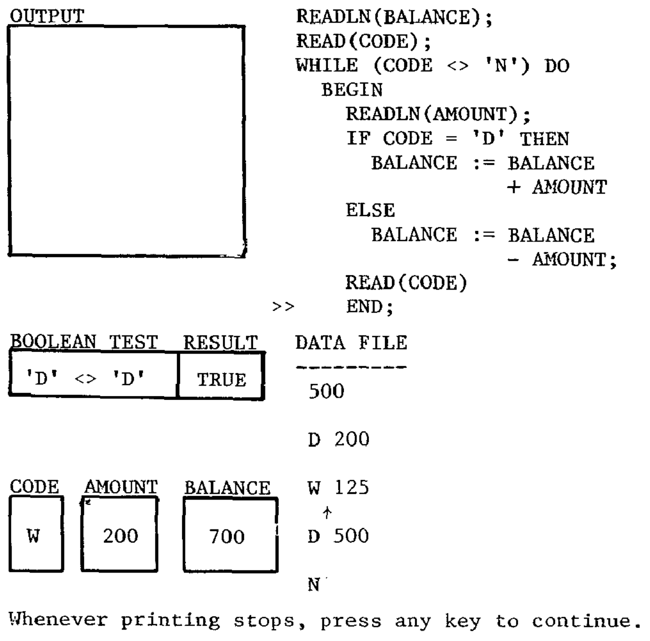


```
OUTPUT                      PROGRAM BANKSTATEMENT;
                            VAR
                              CODE : CHAR;
                              AMOUNT : INTEGER;
                              BALANCE : INTEGER;
                            BEGIN
                         >>   READLN(BALANCE);
                              READ(CODE);
                              WHILE (CODE <> 'N') DO
                                BEGIN
                                  READLN(AMOUNT);

BOOLEAN TEST   RESULT       DATA FILE
                            ---------
                             500

                             D 200
                            ↑
CODE   AMOUNT   BALANCE      W 125
                 500
                             D 500

                             N

Whenever printing stops, press any key to continue.
```

Figure 3. A New Value for a Variable

Each of the remaining lines in the data file, except for the last, corresponds to a deposit (D) or a withdrawal (W). The last line (N) signals the end of the data. After each code is read, the Boolean expression in the WHILE statement tests to see if the code is the trailer, the expression is displayed in the BOOLEAN TEST box, and its value (TRUE or FALSE) is displayed in the RESULT box.

Assuming the code is not N, then the body of the WHILE loop "executes". Eventually, as the statements in the loop are simulated, the remaining statements in the body of the loop will scroll into the program-display window (see Figure 4).

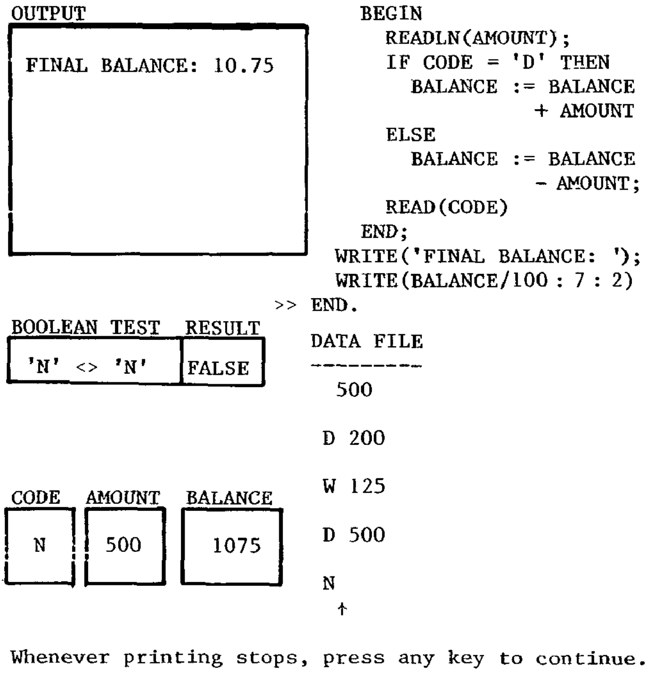


```
OUTPUT                      READLN(BALANCE);
                            READ(CODE);
                            WHILE (CODE <> 'N') DO
                              BEGIN
                                READLN(AMOUNT);
                                IF CODE = 'D' THEN
                                  BALANCE := BALANCE
                                             + AMOUNT
                                ELSE
                                  BALANCE := BALANCE
                                             - AMOUNT;
                                READ(CODE)
                         >>     END;

BOOLEAN TEST   RESULT       DATA FILE
'D' <> 'D'     TRUE         ---------
                             500

                             D 200

CODE   AMOUNT   BALANCE      W 125
                              ↑
 W      200       700        D 500

                             N
Whenever printing stops, press any key to continue.
```

Figure 4. The Effects of Scrolling

Since the body of the loop is small enough to fit in the window, no more scrolling will take place until the statement following the loop is simulated.

After the output statements are simulated the final configuration of the screen is displayed (see Figure 5).



```
OUTPUT                        BEGIN
                                READLN(AMOUNT);
FINAL BALANCE: 10.75            IF CODE = 'D' THEN
                                  BALANCE := BALANCE
                                             + AMOUNT
                                ELSE
                                  BALANCE := BALANCE
                                             - AMOUNT;
                                READ(CODE)
                              END;
                            WRITE('FINAL BALANCE: ');
                            WRITE(BALANCE/100 : 7 : 2)
                         >> END.
BOOLEAN TEST   RESULT
'N' <> 'N'     FALSE         DATA FILE
                             ---------
                              500

                              D 200

CODE   AMOUNT   BALANCE       W 125
 N      500      1075
                              D 500

                              N
                              ↑

Whenever printing stops, press any key to continue.
```

Figure 5. The Final Screen Configuration

Upon pressing return, the user is given the option to repeat the example. If the student chooses not to do so, then the menu for the WHILE loop category reappears on the screen. The student can then select another example or return to the main menu.

Although this example does illustrate most of the features we have incorporated into the layout of the screen, a brief discussion of how procedures and functions are handled is in order. The early examples in that module are designed so that the entire program and all the internal procedures and functions of an example fit in the program display window, thereby allowing the student to visually observe the transfer of control to and from the subprograms. The latter examples are so long that this is not possible, and so we only display the currently executing program, procedure, or function.

The main goal of the procedure and function module is to familiarize students with the transfer of control to and from subprograms, argument-parameter correspondence, and VAR versus value parameters. For this reason, the area of the screen which is normally used for output is used instead for the local variables and value parameters of the subprogram which is currently "executing". In order to convey the notion of a VAR parameter, we display the name of the parameter in parentheses alongside the name of the corresponding argument. This correspondence is further highlighted by using a different color for the name of the parameter.

CONCLUSION

The current approach to teaching programming leaves a gap in the area of understanding the dynamic aspects of a program. The good students are usually able to visualize the dynamic aspects of a program on their own, but less capable students may be greatly helped by using a teaching aid which allows them to see "what really happens inside the computer" during program execution. Such students have much to gain from viewing correct programs as they execute before they attempt to use a given statement construct. In this rapidly growing age of computers, any such teaching aid which may help produce better programmers is certainly worth the efforts of designing, implementing, and, most importantly, using.

REFERENCES

[1] Bitzer, D.L., "Wide World of Computer-Based Education", Advances in Computers, Vol. 15, pp. 239-283, Academic Press, 1976.

[2] Denenberg, Stewart, "A Personal Evaluation of the PLATO System", SIGCUE Bulletin, Vol. 12, No. 2, April 1978.

[3] Dionne, M.S., and Mackworth, A.K., "ANTICS: A System for Animating LISP Programs", Computer Graphics and Image Processing, Vol. 7, pp. 105-119, 1978.

[4] Kramlich, D., Brown, G.P., Carling, R.T., and Herot, C.F., "Program Visualization: Graphics Support for Software Development", Twentieth Design Automation Conference, IEEE, pp. 143-149, 1983.

[5] Olds, H.F. Jr., "The Making of Software", Classroom Computer News, Vol. 1, No. 6, August 1981.

[6] Pepper, Jeff, "Following students' suggestions for rewriting a computer programming textbook", American Educational Research Journal, Vol. 18, pp. 259-269, Fall 1981.

[7] Reiss, Steven D., "Graphical Program Development with PECAN Program Development Systems", Proceedings of the ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environment, pp. 30-41, May 1984.

[8] Ross, R.J., "A Dynamic Library of Interactive Programs", Technical Report CS-81-073, Department of Computer Science, Washington State University, April 1981.

[9] Turbo Pascal Version 2.0 Reference Manual, Third Edition, Borland International, May 1984.