1982

Interactive screen preparation for visible program execution

B G. Hill
University of Wollongong

INTERACTIVE SCREEN PREPARATION FOR VISIBLE PROGRAM EXECUTION

by

B.G. HILL

Preprint No. 82-17

P.O. Box 1144, WOLLONGONG N.S.W. 2500, AUSTRALIA
tel (042)-282-981
telex AA29022

INTERACTIVE SCREEN PREPARATION for Visible Program Execution.

B.G.Hill

Royal Australian Naval College,
HMAS CRESWELL,
Jervis Bay, 2540.

ABSTRACT

The most proficient way to teach introductory computing science is currently a topic of widespread debate in the academic community. One suggestion has been to offer a dynamic algorithmic reading experience prior to the actual writing of algorithms, and methods of providing this have been implemented. The "dynamic reading" lessons give the student the opportunity to gain a clear understanding of basic computing techniques prior to the stage of actual use. The development of these lessons can be eased with computer assistance. The Interactive Screen Preparation package enables lesson authors to concentrate on the educational aspects of lesson preparation, and minimise the time consuming and mechanical aspects of the task.

KEY WORDS AND PHRASES: dynamic display, interactive screen preparation, introductory computing, visible program execution.

CR CATEGORIES: 1.5, 3.3, 4.4

INTRODUCTION

1. Visible Program Execution

The last decade has seen an exponential increase in the number of people who have been exposed to the task of programming computers. A wide variety of techniques have been used to introduce computing skills. Approaches include the mathematical, where programs are developed and concurrently proven correct, algorithmic design techniques, structured development, single language instruction and hybrids of these and other approaches. The goals of instruction are often unclear, and no approach is near achieving a consensus of opinion. A student's first contact with introductory computing may range from the formal tertiary lecture theatre situation to the perusal of some magazines of very dubious worth. With such a plethora of educational avenues, it is inevitable that many prospective computer enthusiasts are receiving a poor introduction, and it is likely that in many cases, the educational methodology leaves much to be desired.

Dromey [1] has suggested that a dynamic computer program reading experience be provided prior to the first writing of algorithms. A team under Dromey's direction has developed the Visible Program Execution (VPE) package, which consists of a collection of subroutines enabling simple algorithms to be "executed" whilst values of the variables are dynamically displayed on the terminal screen. Additionally, it is interactive in that at each program execution step, the student can be requested to provide the appropriate variable value. The use of this package is expected to lead to a greater understanding of the basic laws of the composition of computer algorithms (sequence, selection, iteration and modularity).

At the present time, some thirty elementary algorithms have been included in the VPE library at the University of Wollongong. These algorithms are all implemented in the Pascal programming language, as this university uses Pascal as its implementation language at the introductory level. Algorithms could just as easily be displayed in FORTRAN, BASIC, COBOL or C. The VPE package itself is implemented in Pascal to enable wide circulation of the package to universities and schools where it may be of assistance.

2. Author difficulties

The author of a VPE lesson is faced with two tasks. The first is to prepare the algorithm for display, and to decide on the presentation of the various variables on the terminal screen. The second is to integrate the various VPE procedure calls into the algorithm, and to provide the necessary parameters to these procedure calls. Both aspects tend to be

OVERVIEW OF ISP

In order to fulfill the first goal of ISP, it is necessary to enable the VPE author to construct on the screen a display that will appear exactly as in the finished VPE lesson. The VPE author has five constructs to position on the screen. They are:

a. the text of the executed algorithm

b. any number of boxes. These labelled boxes are used to display the values of individual integer, boolean, real or character variables.

c. any number of array boxes, to display the values of whole arrays of the above types.

d. any number of condition displays, used to question and display the values of booleans in conditional clauses of the control structures.

e. a prompt display. In VPE lessons, the prompt (which reads 'Press RETURN') gives the student user the opportunity to inspect the current state of execution of the algorithm. VPE authors use this feature when an inspection is thought advisable, but where the student does not need to provide a variable value. Progress is continued by the student depressing the RETURN key.

The text of the algorithm must first be prepared, and stored in a file. The algorithm should not exceed the screen limits which, in the case of the terminals at the University of Wollongong, are 23 rows by 70 columns (the ten left most columns are reserved for the display of an indicator which points to the program statement that is about to be executed). The algorithm, with the accompanying statement indicator, is always displayed on the screen with a bias to the top and to the left. All other constructs are positioned at the discretion of the author, although experience has shown that there are favoured "zones" for each category of construct if a balanced display is to be obtained.

The package includes the following elements.

a. An optional demonstration. In addition to the actual demonstration of the features available, this incorporates an introduction, some instructions to assist in the use of ISP and some advice on techniques to gain maximal use from the package.

b. The opportunity to enter new constructs, to change any detail of existing constructs, and to delete existing constructs.

changed is simplified by presenting to the author a list of all known labels (usually variable names) of constructs of the given class, and accepting the typed label as response. This response is checked for correctness. Although this may require duplication of effort due to incorrect entry, it was found to be the quickest method when the labels were short. When the labels were long and more likely to lead to typing errors (for example, a condition label might read 'Is a[index] < a[index-1] ?'), the author is presented with the choices individually and prompted for acceptance on each occasion.

3. Construct change

Prior to actually using ISP, the VPE author should have a clear idea of how the finished display is to look (automation does not substitute for creativity). After the initial entry of the constructs, changes may be necessary, depending on how the result compares with the original ideas, and ISP enables such changes to be made simply and promptly. Should the display prove to be too congested, the author may wish to delete some aspect from the display. All constructs are deletable at the author's discretion.

4. Validity checking

All entered construct data is checked for validity, with respect to type and size, and with respect to the current screen display. If the given information leads to a "can't fit" diagnosis, then a further check is made to ensure that the construct is legal (that is, that it can be displayed somewhere on the screen). If this is not the case, a message to this effect is given, the latest details entered (that is, those that lead to this "disaster") are deleted from the record and the author is re-positioned at the last valid operation prior to this disastrous entry or change.

5. Condition handling

There is one situation where one construct can be displayed at the same position as another. There is a convention in VPE programming that a condition construct is first displayed at the correct stage of program execution, and is retained on the display, with the valid response, until either the condition may have become invalid (due to an assignment to one of the variables constituting the boolean condition) or until the screen area could be more profitably used to display another condition. Multiple conditions can therefore be displayed in the one area of the screen at different points in time. Although conditions can be displayed concurrently in different screen areas (as with the box and array constructs), the re-use of the one screen area presents an attractive screen balance in many lessons.

VPE package itself). The data entry checking and editing software, although basic in principle, comprises a significant portion of the whole. Early in development, it became obvious that the ISP program was too large for the University's Unix software to handle (that is, it would exceed 32 Kbytes of object program), and steps would be necessary to limit the program size. This had an effect on the design of the data structures necessary.

2. Data Structures

2.1. Construct data records

The number of constructs utilised in a VPE lesson obviously varies to meet the needs of the particular lesson. Of those lessons currently implemented, the number of constructs varies from 2 to 14. ISP needs to retain information about every detail of each construct, and the details needed vary with each class of construct. The obvious candidate for maintaining these details was the Pascal variant record. Rather than use an array of, say, 25 such records, with the consequent under utilisation of much of the array most of the time, it was decided that a simple singly linked linear list would be more efficient.

Each of these records takes the following form:

row datum
column datum
label of variable, or content of boolean condition
length of the label, calculated on entry of the label
(needed when deleting the display of the construct)
length of each box, preset for all except individual array box length and real number box length
pointer to the next record

The variant portions are:

array:	the number of individual boxes
box:	the type of box (integer, boolean, real)
	is the label highlighted?
condition:	is the prefix 'CONDITION:' required?
	is this condition currently displayed?

2.3. Prompts and questions

The prompts and questions are stored as arrays of strings, and displayed in the "command line" when required. Initially, these strings occupied about 13Kbytes. This was reduced by a factor of five by dividing them into files containing only those questions relating to each of the five classes of construct, and only loading those questions for the class of construct currently being positioned. This has the unfortunate side effect of causing a delay of 1 to 2 seconds whenever a new class of construct is to be positioned.

3. Algorithms

The overall control flow is:

```
1. if author wishes to see an introductory demonstration
   then
            provide it
2. display author's algorithm
3. repeat
            find class of construct of interest
            repeat
                    allow additions, changes or deletions to
                    items in this class of construct
            until
                    actions for this class are completed
   until
            author is satisfied with the display
4. translate accumulated data into a skeletal Pascal program
```

3.1. Demonstration

The demonstration section of ISP provides

a. an introduction

b. an explanation of the basic screen layout, with the row and column convention discussed

c. an explanation of the use of the questions and answers

d. an explanation of the constructs available

e. a step by step demonstration of the constructs available, with advice on the best ways to utilise each

The VPE author has the choice of viewing only selected parts of the demonstration, or of the whole. The demonstration builds a balanced screen from a displayed algorithm, with the result that, if all steps are taken, then on completion of the demonstration, the screen has been prepared

example, which concerns the initial steps in the entry of a new construct.

```
get row and column datum point
place minimal dummy values for all other components
check this minimal configuration for legality
if not "disastrous"
then
        while not legal
                allow positional change
                check for legality
```

It will be recalled that "disaster" refers to the attempted entry of a construct which can not legally be positioned in any portion of the display. The above cycle, along with any additional checks necessary (for example, variable labels are checked for a maximum length and for uniqueness), is then repeated with the addition of one extra author input parameter at each incremental stage until all details of the construct have been entered, after which the construct is displayed on the terminal. Although this may appear to be overly protective to the VPE author, the actual checks are very rapid (as indicated in the section on data structures), and there is no noticeable delay between an item of input and the prompt for the next.

3.4. Translation

When the VPE author is satisfied with the screen display, the accumulated information is translated into a skeletal Pascal program. An example of the output program is shown in Appendix C. There are a number of components of each construct that need to be translated into the Pascal program. These values are defined as constants. The name given to the constant is generated from the label displayed by stripping it of all non alphanumeric characters and converting remaining characters to upper case. One of five suffixes is then attached. These are ROW (the datum row), COL (the datum column), LONG (the length of a real box, or the length of a condition message), NUM (the number of individual boxes in an array) and DELTA (the length of the individual array boxes). As an example, an integer box bearing the label "a[i-1]:" will produce the Pascal constants AI1ROW and AI1COL. The number of constants created is dependent on the type of construct.

The algorithm to generate these constants in the output file is:

differ from that of the VPE display.

b. this restricted a meaningful choice to either the top line or the bottom line.

c. the top line is almost certain to include text (the first line of the displayed algorithm) and thus is unlikely to be a candidate for the display of variables.

d. the bottom line rarely has text displayed, and is a more likely candidate for the display of variables.

e. entry of text on the command line uses a carriage return to indicate end of entry, in keeping with most computer software. A carriage return on the bottom line of the screen causes the screen to scroll up one line, causing the top line to be lost, and the display no longer follows the "what you see is what you get" principle. To avoid this occurrence, the cursor is kept clear of the bottom line of the screen unless a construct is specifically required to be drawn there.

The incorporation of some features has provided a friendly and helpful user interface, and enabled authors to be more productive. The validation of all input at the time of entry traps most typographical errors at their source. On the rare occasions that these are not trapped (for example, a valid but incorrect value), the immediate display of the construct makes the error obvious, and the ease of changing details permits rapid correction. Also, the input validation techniques permit rapid and easy experimentation with the position of any construct, and enable the VPE author to prepare a balanced display quickly.

Additionally, author input has been kept as simple as possible. The number of author entries has been minimised by utilising a wide variety of questions and prompts, most of which require only a single keystroke response. By making each question only relevant to the current position in the interactive session, author input has been minimised without forcing a response to a tiresome series of mostly irrelevant questions, a practice which has been noted in some commercially available software in other fields.

These features have brought favourable comment from a number of VPE authors who have used ISP.

EVALUATION

To verify the correctness of ISP, all the VPE programs at the University of Wollongong were re-created using ISP. In addition to demonstrating the "user-friendly" aspects outlined above, this exercise showed that a considerable

APPENDIX A

INSTRUCTIONS FOR THE USE OF ISP

Assumptions

It is assumed that any person using this package is familiar with the VPE package, and has actually worked through a selection of the algorithms therein. Additionally, the person should have some knowledge of teaching techniques, and will have understood the reasons for the VPE algorithms using the display techniques that they do, as well as understanding the algorithms themselves.

Disclaimer

This package is not intended to teach the technique for preparing VPE lesson material. It merely makes the task of preparing such lessons easier.

Introduction to ISP

The ISP package contains a section consisting of an introduction, some instructions to the novice user, some advice on techniques, and a demonstration of the available features. These can be studied by responding to the opening question "do you wish to see a demonstration of the structures available (Y/N) -" appropriately. All instructions are suppressed in the actual information gathering package in order to eliminate such annoying features from distracting the experienced user.

Use

In actual use, there is no need for any operating instructions. The VPE author merely needs to respond appropriately to the questions and prompts provided. Inappropriate responses (up to, and often including, the level of deliberate sabotage) are filtered out, and the author given another opportunity. Only relevant questions and prompts are presented - there is no need to provide any irrelevant information.

A hint

When an attempt is made to place a construct in a position that would entail overwriting some feature already present on the screen, or would exceed the screen boundaries, the user is warned with a message such as "the box at 6, 16 will not fit", and is given the opportunity to reposition the construct. Usually, the reason for the error will be obvious, and repositioning can be accomplished by

APPENDIX B

AN INTERACTIVE SESSION

A view is shown of the screen display at the conclusion of a simple interactive session. In this example, only one construct of each type is shown. The algorithm obviously needs boxes for n, i, a[i] and halfmax as well as for max, but these have not been included in order to simplify the example. On the next page, the prompts and questions, with the appropriate responses, are shown for the session which produced this screen display. Changes to screen presentation are described in braces.

SCREEN APPEARANCE

```
enter or change - Boxes, Arrayboxes, Conditions, Delay or Finished? f
          var    i, max    : integer;
                 halfmax   : real;
          begin  {finds maximum in array and halves it}
          max := a[1];
          for i := 2 to n do                                    -----
               if a[i] > max then                      max: | 6,65|
                     max := a[i];                           -----
          halfmax := max / 2;
          end;



          ---- ---- ---- ---- ---- ---- ---- ----
  ARRAY: |16  | 9  |    |    |    |    |    |    |
          ---- ---- ---- ---- ---- ---- ---- ----
           i
                                                               -----
                                  CONDITION: [ Is a[i] > max ? ] |20,70|
                                                               -----
Press RETURN.22, 0
```

APPENDIX C

THE FILE PREPARED BY ISP FROM THE SESSION SHOWN IN APPENDIX B

Note that as the actual listing can use up to 120 columns, some lines have been truncated.

```
{       VISIBLE PROGRAM EXECUTION SERIES

        A simulation of the steps involved in "executing" a computer program.

        This program simulates . . . . . .

}


program explain (input, output, trial, cminstructions, tellfile);

const   ARROWROW        = ?;            {arrow pointer controller.}
        ARROWCOL        = 0;            {arrow pointer controller.}
        MAXROW          = 6;            {row for the box labelled 'max:'.}
        MAXCOL          = 65;           {column for the box labelled 'max:'.}
        ARRAYROW        = 16;           {row for the array labelled 'ARRAY:'.}
        ARRAYCOL        = 9;            {column for the array labelled 'ARRAY:'.}
        ARRAYDELTA      = 5;            {displacement between adjoining boxes in
        ARRAYNUM        = 8;            {number of individual boxes in the array
        COND1ROW        = 20;           {row for the condition having the questio
        COND1COL        = 70;           {column for the condition having the ques
        COND1LONG       = 15;           {length of the condition having the quest
        SAFEROW         = 22;           {row for the delay construct showing 'Pre
        SAFECOL         = 0;            {column for the delay construct showing '

{       Many of the declarations in the next lines may not be
        required in the specific context of this program.
        Please edit out those that are unnecessary. Those
        not marked by ** must remain.
}

type    chstring     = array [1..60] of char;
        chrname      = array [1..20] of char;
        filename     = text;
        nelements    = array [0..20] of integer;
        details      = record
                         row, col, labelength : integer;
                         value                : boolean;
                       end; {of record}                  {**used in proc.i}
        recarray     = array [1..10] of details;         {**used in proc.i}
        realdetails  = record
                         row, col, labelength, boxlen : integer;
                         value                        : boolean;
                       end; {of record}                  {**used in realnum.i}
        realrecarray = array [1..10] of realdetails;     {**used in realnum.i}
```

```
setarrow (ARROWROW, ARROWCOL, step);     {initialise arrow display.}

{At this point of your program you should insert the program that is being
 simulated, interspersed with the necessary VPE procedure calls to make it
 appear that the program is being executed. If your program includes the
 display of conditions, the necessary procedure calls are outlined below.}


{The following procedure calls will be used in your program.
 Use your editor to place the correct parameters in the positions
 nominated by question marks in the parameter list, and to place
 these procedure calls in the correct spot in your program.}

getsetcondition (?, ?, ?, ?, ?, ?, step, ?, ?, COND1ROW, COND1COL,
                                    'Is a[i] > max ?', true, ?, qmode);
erasecondition (COND1ROW, COND1COL, COND1LONG, true);

{As well as being used extensively in the procedures which produce the
 apparent execution of the VPE program, a delay construct is normally
 placed at the very end of the program in order to give a clean
 finish when the student completes the allotted task.}

delay (SAFEROW, SAFECOL);
tellofsuccess ('trial', failure);              {report user success.}
end.    {main program explain}
```