TEACHING PROGRAMMING TO THE DEAF

Rockford J. Ross
Computer Science Dept., Washington State University
Pullman, Washington, 99164

ABSTRACT

For certain groups of handicapped persons the field of computer science offers challenging, high-paying careers which are more accessible than other careers. Programming can be done at cathode ray tube terminals which provide a dynamic, visual environment needed by the hearing impaired, and which can be placed in locations convenient to those with mobility impairments. Unfortunately, most colleges and universities are not prepared to teach such students, particularly the deaf. In this paper an ongoing research project to design an online, dynamic, student-controlled library of programming language examples for use in teaching programming languages is described. This system promises to be useful in meeting the needs of handicapped students.

It is apparent that for many handicapped persons the career field of computer programming is ideal. Most programming today it done with the aid of cathode ray tube (crt) terminals. For the deaf, the dynamic visual environment presented on a crt is suited to their needs. People with various mobility impairments have found working at a crt terminal to be quite convenient, since terminals provide the complete power of the computing system being used and can be placed in accessible locations (e.g. at home). However, such students must be trained to program; the standard teaching methodologies employed by the vast majority of universities and colleges in programming courses prove inadequate for such students. Programs at special institutions tailored to the needs of these students are apparently also not available [1,3,9,10,11,12,13].

In the remainder of this paper we will focus on the particular problems faced by the deaf in learning programming. The project described holds promise of helping not only the deaf but also students with weak backgrounds and some with other physical handicaps.

The way programming is taught in the Computer Science Department at Washington State University is probably typical of most large public institutions. The introductory course for computer science students is CptS 210. This four-credit course is divided into a three-hour lecture and a three-hour laboratory. In the lecture section, which meets three times a week for one hour, general background material is presented, and new Pascal programming language concepts are introduced; in the laboratory, a teaching assistant (TA) further explores the Pascal concepts presented in the lecture and provides the students with guided

experience in writing Pascal programs in a much smaller group setting. (Pascal is widely accepted as the best available language for teaching programming. See e.q. [5].) To provide the students with outside sources of help the lecturer and the TA's establish office hours outside of class. Also, advanced students (usually juniors or seniors) provide programming language consulting on a formal basis from 9:00 am to 10:00 pm most days of the week. Thus, care has been given in providing students with good sources of help, both in a formal classroom setting and outside of class.

For good students this approach is satisfactory. Marginal students, however, often have trouble grasping programming concepts which should not pose much difficulty. But for the deaf, this standard approach is nearly useless. Most of the problems faced by the weaker students and especially the deaf stem from what we term the textbook or static approach to teaching programming languages. Students are forced to learn about programming language constructs by reading static examples in textbooks or from the blackboard in class. There are, however, two distinct parts to a computer program, one of which does not lend itself well to this textbook approach: the static expression of the program in some programming language, and the dynamic execution of that program on a computer. The novice programmer must understand these two facets clearly in order to learn programming. Presentation of new programming language 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.

The current method of resolving the problem is to have the instructor, TA, or consultant provide the "dynamic interface" by playing computer at the blackboard. While this is helpful, the students must once again write the examples down in their notes in static form, and the problem repeats itself. The tremendous proliferation of introductory programming language texts is no doubt in large part a fruitless attempt to correct these deficiencies and address student complaints.

For the deaf these problems are nearly insurmountable. While they are able to attend class, they lose the essential advantage provided by the classroom environment -- the dynamic explanations of the instructor. Most often, they must depend on a professional notetaker, who usually has no computer science background, for information on the lecture. Needless to say, by the time the student reads these notes the information content is minimal. The whole experience is

extremely frustrating; textbooks are really the only sources of help for such students outside of intensive individual instruction.

It was while working with a deaf student in a few programming courses that the idea of developing a dynamic library of programming language examples was first considered. The student owned a microcomputer and was attempting to create a few such examples for his own use in understanding certain concepts presented in class. The underlying idea is quite simple: an online aid which allows a student to select examples of specific programming language concepts for dynamic display on the screen would give students the needed "dynamic interface" for understanding programming. Under his own control and at his own pace, the student could watch the execution of a program unfold; changing variable contents and conditions causing branches would be highlighted during the "execution" of each statement.

THE DYNAMIC LIBRARY

Work on a pilot program to test the concept of providing dynamic programming language examples on crt screens was begun in the summer of 1980 by a Master's student under the direction of the author [6]. The result is known as LOPLE (Library of Programming Language Examples). The LOPLE system consists of the actual dynamic library as well as various library maintenance routines which allow an instructor to add new examples to the library and delete undesired examples. A more complete description of LOPLE can be found in [6 .7]. Here we will only describe how LOPLE works from the student's point of view.

In a typical situation a student is contemplating a new programming assignment. Unsure of the new concepts to be used (for example, looping), the student may first sign on to a crt terminal and request entry into LOPLE. At this point, he has four choices: he may browse through an index of keywords describing programming language concepts included in the library (e.g. if-then-else, repeat, while, etc.) looking for the concepts of interest. This index is displayed on the screen and may be re-displayed at any time. Once an index entry has been chosen, e.g. while, typing "while" will cause a short description of each library entry which includes a while construct to be listed on the screen. Typing the number of the desired example from this list will cause that example to be displayed on the screen, ready to be "walked through" by the student. Of course, if the student already knows that he wants to view examples of the while construct, he need not first review the index. Similarly, if he recalls a particular example number from a previous session which he wishes to review, typing that number directly will cause this example to be displayed. Typing "stop" at any point causes an exit from LOPLE.

Figure 1 shows how a typical example would appear on the screen.

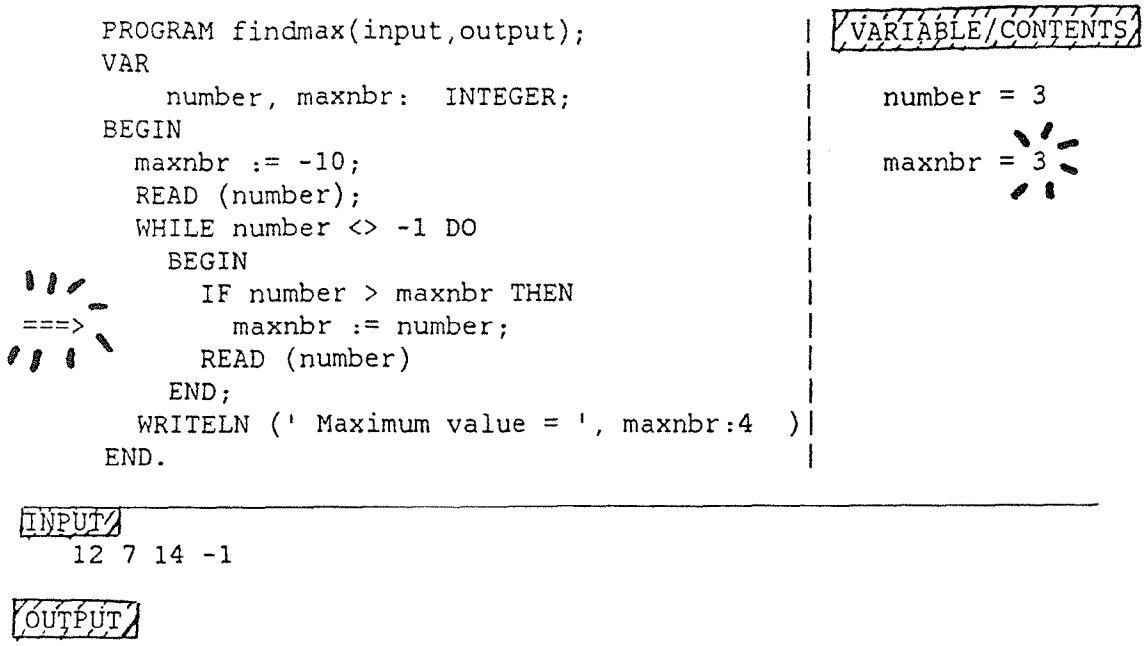

Figure 1

(The description accompanying the figure was difficult to write and is probably not trivially easy to understand because of the very problem addressed by this paper: we are attempting to present and describe something which is dynamic in static form). Notice that the screen is divided into three sections. One section contains the program text. At the bottom is the input/output area which displays the values read in by the program and the results printed during execution. In the upper right-hand corner the variables used in the program are displayed along with their current values. Use of inverse video, blinking, and dashed lines highlights various items of interest. Finally, there is a blinking instruction pointer which points to the current line being "executed".

The strings highlighted above with surrounding dashes represent blinking symbols on the crt; those in shaded boxes represent string displayed in inverse video on the crt.

In this figure the whole loop is being executed for the first time. The first input value, which was a 3, has been read and assigned to the variable "number"; it no longer appears in the input stream (the next input value is 12). Since the condition of the IF statement is true this first time through the loop, the then statement is executed. In fact, this is the statement being "executed" in this snapshot, as evidenced by the symbol "==½". The value in "number" is assigned to the variable "maxnbr", as shown by the blinking 3 in the VARIABLE/CONTENTS column.

Student Feedback

LOPLE has been running since February, 1981, at WSU. A few laboratory sections of the introductory Pascal programming language course, CptS 210 were chosen to test this first version (unfortunately, no deaf objective evaluation of LOPLE has been possible, the subjective opinions of both the teaching assistants and students who have used it were quite favorable. An informal survey conducted at the end of the course gave predictable answers: all students were intrigued by LOPLE. The good students found that they received only a little additional help by viewing the examples (they were able to visualize the dynamics of a program in their heads without outside aid); however, they did enjoy looking ahead at advanced examples not yet covered in the lecture. Less capable students were helped a great deal by LOPLE (a number of these students commented that they would have been totally lost without the use of LOPLE). In one laboratory where statistics on the use of terminal time were kept by the system, a midsemester check showed that they had already spent an average of average of 1.2 hours viewing LOPLE examples, even though it was not required of them. The same group had spent an average of 11.1 hours of time at the terminal entering and debugging programs (which were mandatory). (The relatively poor typing abilities of many students make detailed comparisons between these numbers impossible without further study).

While these informal results cannot be used to accurately predict the success of such a system in teaching the deaf, they certainly offer hope. One thing is clear: the dynamic library approach provides instructors with substantially more help in meeting the needs of the handicapped than traditional teaching aids.

Future Directions

LOPLE was conceived and implemented as a test project. As a result it was not designed to be transportable. It makes use of many features peculiar to the computing environment at Washington State University; it would undoubtedly not run on other computing systems without extensive modification. It is also an expensive program to run, a fact which has precluded more exhaustive testing of the project.

The results of this pilot project, however, have been encouraging enough that outside funding sources have been sought to develop the project into a practical tool. A grant from the Foundation for the Advancement of Computer Aided Education has provided the author with two Apple microcomputers for this purpose. Currently two Master's students are working to develop a LOPLE-like system to run on these small computers. Work is expected to be completed on these projects by June, 1982. Another student is beginning work to implement a similar system to run on the central Washington State University computer facilities under CMS, an interactive processor which is available at many educational institutions. This combination of microcomputer-based and large computer-based LOPLE programs should allow the system to be used by many people.

Summary

Work the LOPLE has shown that the dynamic presentation of programming language concepts at crt terminals can be very useful in helping students understand the dynamics of a computer program. Students with a weak background and handicapped students stand to profit most from the use of such a system.

Various general CAI systems already exist for presenting lessons dynamically at a monitor. Some are undoubtedly being used in different locations (see, for example, the editor's note in [8] but have not been documented or reported. Those like PLATO require the use of specialized hardware and can be quite expensive to use [see e.g. 2]. Others which involve the use of interpreters [see, e.g. 4] assume that the students are able to write a correct program which can be viewed while being interpreted at the screen. LOPLE augments this idea by providing correct programs for students to view "in execution" before they attempt to write their own programs. Continuing experimentation with LOPLE shows that efficient, useful versions can be implemented inexpensively on microcomputers and host-based systems.

Use of LOPLE-type systems should provide incentive to those with various handicaps to seek careers in computer science. Their wide-spread use will make the teaching of programming to these students easier.

References

1. Computer, January 1981. Special edition on computer based aids for the handicapped.

2. Deneberg, Stewart. A Personal Evaluation of the PLATO system, SIGCUE Bulletin, Volume 12, Number 2, April 1978.

3. Gallaudet College. A Guide to College/Career Programs for Deaf Students, Gallaudet College, December, 1978.

4. HSCS - A High School Computer Science Curriculum for the 1980's. Description sheet, University of Tennessee and the National Science Foundation. Describes INTERPAS, a Pascal based interpreter which runs on APPLE computers.

5. Merritt, Susan M. On the Importance of Teaching Pascal in the IS Curriculum. ACM SIGCSE Bulletin, VOlume 12, Number 1, February 1980, pp 88 - 91.

6. Rezvani, Saiid and Ross, Rockford J. A Dynamic Library of Interactive Programming Language Examples. Technical Report CS-81-073, Department of Computer Science, Washington State University.

7. Ross, Rockford J. LOPLE: A dynamic Library of Programming Language Examples. Technical Report CS-082-085. Department of Computer Science, Washington State University.

8. SIGCUE Bulletin. Association for Computing Machinery, Special Interest Group on Computer Uses in Education, April/July 1981.

9. SIGCAPH. Special Interest Group on Computers and the Physically Handicapped, Association for Computing Machinery, recent editions.

10. Von Feldt, James. An Overview of Computers in Education, Technical Report, National Technical Institute for the Deaf. Rochester, New York, March 1977.

11. Von Feldt, James, Introduction to Computer Application in Support of Education Technical Report, National Technical Institute for the Deaf. Rochester, New York, August 1977.

12. Von Feldt, James. A National Survey of the Use of Computer Assisted Instruction in Schools for the Deaf, Technical Report, National Technical Institute for the Deaf, Rochester, New York.

13. Watson, Paul. Utilization of Computers with the Hearing Impaired and Handicapped, Technical Report, Gallaudet College, 1979.

DPI Difference

* Offers both training and employment
  -- Exposure to "Real World" programming jobs via work on programming
     service contracts
  -- Provides work history and two to five years of experience through
     demanded by prospective employers

-- Exposes individual DPI employees to prospective employers through
face-to-face contact during contract task performance

* "Flex" -time
-hours
-jobs
-locations -- homebound work stations

* Ability to "tailor" training to individual employer needs

* Provides a demonstration vehicle to break down imagined "barriers"
to employment of disabled persons

* The DPI approach is a model for other metropolitan areas where DPI's
facilities can be replicated through cooperative efforts of industry,
business, handicapped and professional associations, chambers of
commerce and local state and federal government agencies

* Continuation of any DPI branch office will not depend on contributions
to any great extent, but will be largely self-supporting via contract
revenues.

* Uses "recycled" - buildings
- equipment
- furniture

WHY DPI SHOULD SUCCEED

* There is a world-wide shortage of computer programmers with demand
far exceeding supply for as long as anyone dares forecast.

* There is an average backlog of 2 years of programming work in data
processing shops, much of which can be "contracted out".

* Programming is uniquely suited to persons with disabilities:
-- Requires little mobility/physical dexterity
-- Largely an individual contribution, intellectual job
-- Many aids exist to accomodate various handicaps:
"Talking" Terminals
Optacon Readers
Braille Printers/Text

Speech Synthesizers
Special Equipped Vans
Teleconferencing
-- Distributed Processing

* Programming offers a dignified, highly remunerative career to disabled persons

* After start-up, DPI will be largely self-supporting through contract revenues and will require a minimum of reliance on contributions or grants.

* Last but not least, EVERYONE WINS
  -- Taxpapers (Local, State-Federal)
  -- Insurance Companies
  -- Data Processing Industry
  -- Other Industries & Businesses
  -- Government Agency D.P. Shops
  -- Families of Disabled Persons
  -- Disabled Persons Themselves