Getting More Oomph from Programming Exercises

Alan L. Tharp
Computer Science Department
North Carolina State University
Raleigh, North Carolina 27650

ABSTRACT

Much attention has been given to the content of introductory computer science courses, but based upon a perusal of introductory textbooks, it appears that somewhat less attention has been given to the programming exercises to be used in these courses. Programming exercises can be modified to provide a better educational experience for the student. An example of how atypical programming exercises were incorporated into an introductory programming language course is described.

Keywords and Phrases: Programming exercises; introductory courses; motivation; practical knowledge; applications; new technology.

INTRODUCTION

Originally the primary goal of introductory computer science courses was to teach the syntax of a programming language; the programming exercises then quite naturally stressed language syntax. As the field of computer science evolved, the emphasis in introductory courses gradually shifted from syntax to the basic principles of programming; the programming language became a vehicle to illustrate the programming concepts and the exercises then emphasized programming fundamentals. The focus of the following discussion is on modifying programming exercises even further to provide greater benefit to the students.

Paraphrases of typical programming exercises (not especially good nor especially poor) randomly chosen from introductory computer science textbooks are found in Figure 1. One close scrutiny, one may argue that some do not adequately illustrate either programming language constructs or concepts. One might even argue that a few of the exercises are just plain DULL to a large number of students. However, even if the goals of teaching language syntax and concepts were met with these programming exercises, additional goals could be achieved.

Motivation is a tremendously important component of instruction. It is now even more necessary in the discipline of computer science, because an increasing number of students have had prior computer experience upon entering college; with uninteresting or non-challenging programming exercises, many students lack incentive and become disenchanted. In addition, many students are now entering the field of computer science for reasons other than their being enthused about the discipline; such students require greater motivation.

Malone (1980) studied the motivational factors of learning in the context of instructional design; he comments that "if students are intrinsically motivated to learn something, they are likely to spend more time and effort learning, feel better about what they learn, and be more likely to use it in the future." In his study, he categorizes the motivational factors into three groups: challenge, fantasy, and curiosity. In discussing the challenge factor, he reports on one of Csikszentmihalyi's (1975) observations on what makes an activity intrinsically motivating — "There should be a clear criteria for performance; one should be able to evaluate how well or how poorly one is doing at any time." Papert (1972) also argues that it is preferable for a student to discover for himself if something is correct. One factor common to several of the programming exercises in Figure 1 is that the student <u>cannot</u> determine by herself if the answer is correct. Another consideration relevant to motivation is that the skill being taught (in our case programming) does not need to be the primary goal for an exercise but rather a means to achieving the primary goal.

Programming exercises could be improved to be more motivating.

- Write a program to convert an integer number to its Roman numeral equivalent and vice versa.

- Write a program which when given N, counts the number of primes $\leq$ N.

- Write a program to print the numbers from 1 to 10 in one column and the numbers from 10 backward to 1 in another column to the right of the first. Hint: The sum of the two numbers on any line is 11.

- Write a program using the Euclidean algorithm which will find the greatest common divisor of two positive integers.

- Write a program to compute an employee's pay using the formula

  PAY = GROSS - TAX - DEDUCT where

  TAX = TAXRATE * GROSS and

  GROSS = RATE * HOURS

  The values for RATE, HOURS, TAXRATE and DEDUCT are inputs to the program.

- Write a program to read a page of text (50 lines of 80 characters) and print the words that are used on that page. Words are to be printed in alphabetical order with each word appearing only once; words that are used more than once will have a frequency count (e.g., AND 12). The maximum length of a word may be assumed to be 10 characters.

Figure 1
Representative Programming Exercises

A dilemma in computer science education is that much information necessary for completing practical computer science tasks is not considered worthy of university credit, (e.g. reading a file from a tape or learning to use an interactive computer system; the activities in this classification may vary from campus to campus). How, then, is a computer science major to acquire this basic knowledge if it is not presented in a course? Currently the student may have to acquire it on his own or from another student and although these methods are sufficient in many cases, the student may obtain incomplete or incorrect knowledge. (Of course no guarantees can be made that such deficiencies would not occur if this material were presented formally.)
Programming exercises could be improved to provide much of this practical knowledge.

Computer science is a diverse field; how is a student to be introduced to its many subareas? Most curricula allow students a choice of electives, especially in their final year or two. If a student has not been introduced to the elective areas, how does he know if he is interested in a course without having to actually take it?
Programming exercises could be improved to provide an introduction to the many elective areas of computer science.

Computer technology is changing rapidly. One of the primary purposes of this symposium is to introduce new technology to the computer educator. How can this information in turn be introduced rapidly into an undergraduate curriculum? How can the students at least be exposed to these concepts? When a new technology surfaces, there may be a significant time lapse before it evolves to fill a complete course or filter down to the undergraduate level. Again the few very enthusiastic students would probably acquire this information independently, but what about the majority of students?
Programming exercises could be improved to provide an introduction to the new ideas and technology of computer science.

Four needs related to computer science education have been presented: motivation, presentation of essential but non-academic information, introduction to various areas of computer science and introduction to new computer science technology. One method of meeting these needs is through improved programming exercises. The next sections describe an example of how programming exercises can be used to meet these four needs.

BACKGROUND

Programming exercises which were designed to achieve the goals outlined in the introduction were incorporated into a course whose primary purpose was to teach a programming language. Although this experience was with one programming language, the method is relevant to essentially any course in which programming is a major part. The course used in the study is a one credit hour, elective, sophomore-level course on the SPITBOL (SNOBOL) programming language. This course is one of a set of one credit hour courses offered at North Carolina State University with the purpose of teaching a programming language other than the primary one used in the computer science curriculum (currently PL-I). The courses are non-graded pass-fail in which the primary evaluation of the student is his ability to program. Courses exist for APL, COBOL, FORTRAN, LISP and PASCAL in addition to the one in SPITBOL.

In the SPITBOL course, the students are given a set of ten programming assignments of varying difficulty with a total value of approximately 170 points. They must satisfactorily complete 100 points to obtain credit for the course. The assignments are distributed throughout the semester.

EXAMPLE TASKS

Example programming exercises used in the SPITBOL course are sketched to give the reader an idea of the type of exercises being suggested. A reader's lack of familiarity with the SPITBOL programming language should not hinder him from understanding the discussion; it is intended to provide examples rather than the concepts of a specific programming language.

a. Programming goal - programmer defined input/output
Exercise - Read information from a master tape, manipulate the input, write the information to your tape, rewind your tape and then print its contents.
Evaluation -
With this exercise the students gain exposure to tapes in addition to learning about input/output in SPITBOL. To increase the student's interest in the assignment (and since the specific data chosen is independent of the programming goal), data was selected which after manipulation became a poster popular among computer science students, such as a portrait of Einstein or a sketch of a gorilla with the inscription "THINK".
Since the final product was something of interest to the students and since it was immediately obvious if the program ran correctly, it provided motivation to most of the class. A secondary advantage of such an exercise is that the students then have a reduced urge to illicitly print such posters and waste scarce computer funds.

b. Programming goal - built-in functions for string processing
Exercise - Design a data entry form for use with a CRT. Program the CRT to display this form by sending the appropriate control characters to the CRT. Then use the form for entering data and print the contents of the form in a readable format.
Evaluation - The students gained experience using a CRT for something other than a "glass teletype". The students were exposed to an aspect of computer science which in and of itself would not be worthy of academic credit. And the students were challenged to see how simple they could make the form. Again they were able to determine for themselves when the program worked properly.

c. Programming goal - use of external functions
Exercise - Load several external functions which provide the capability to draw on a graphics terminal - Draw, move, print, ... Then construct an object of your choice on a graphics terminal and after obtaining the object, output it to a plotter. (A utility program was provided to obtain a hardcopy of the graphic).
Evaluation - Students drew objects ranging from abstract art to notes on a musical scale. Figure 2 shows typical results of the output.

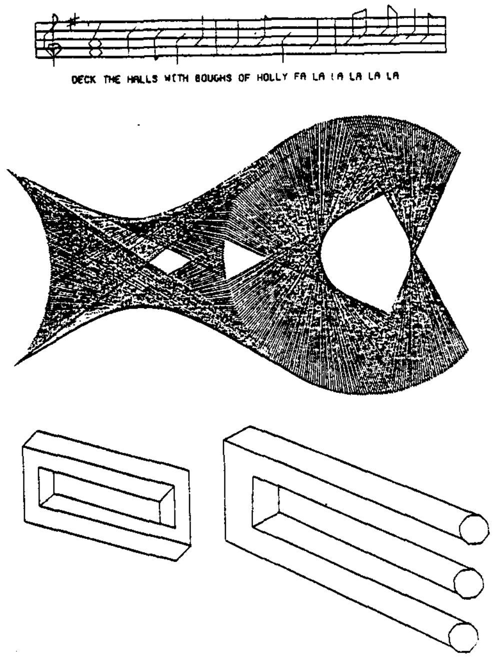

Figure 2
Representative Graphics

The students were again motivated because the result was something of interest to them and they knew when it was correct. They were introduced to the popular subarea of graphics and they were able to use the relatively new technology of graphics terminals. (As an alternative, color graphics could have been used to make the task more interesting and challenging).

d. Programming goal - programmer-defined functions
Exercise - Using the graphics procedures of a previous assignment, write a programmer-defined function which will draw a polygon given its angle and the length of its sides as parameters. Call your polygon function with three sets of parameters and obtain plots of the results.

Evaluation - Plotting can be a valuable aid to displaying information from a computer, but learning the details often does not appear in an undergraduate curriculum or is not deemed worthy of credit. The procedure for using a plotter may appear difficult and foreign to many people primarily because of ignorance. Upon completing this programming assignment, in addition to having developed an understanding of programmer-defined functions, the students had gained familiarity with using a plotter and had learned how simple it is when sufficient information is provided. If looping were also a concept to be illustrated, the students could have been asked to plot a numeric function over a specified range of values. (See Figure 3)

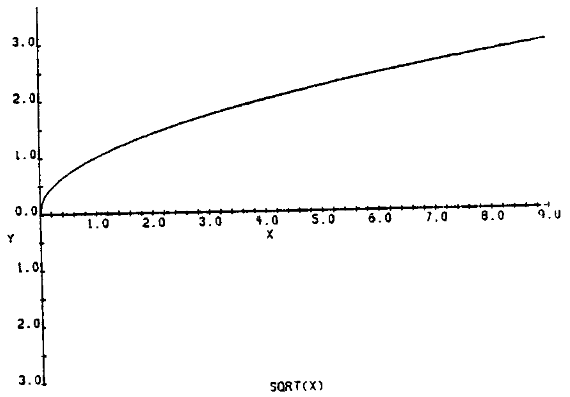

Figure 3
Graphics Example Illustrating Looping

e. Programming goal - programmer-defined datatypes
Exercise - The ASCII codes necessary to drive a VOTRAX speech synthesizer for a few common function words such as "a", "in" and "the" and a few content words such as "computer science" are stored on a disk data set. (A complete list of such words is given. Instructions are also given for invoking a text-to-speech translator for obtaining additional content words such as the student's name). Construct a programmer-defined datatype for a linked list where each node structure contains three fields - one for the word, one for an ASCII code and one for a link, i.e.

Word	ASCII code	Link

Link together the ASCII codes in ascending order of the associated words for both the stored and supplemental vocabulary. Then output the resulting list via the synthesizer. Also form a linked list of words for two sentences of your choosing and then have these lists output through the voice synthesizer.
Evaluation - Using a speech synthesizer is merely one example of introducing students to new technology which may not otherwise be part of an undergraduate curriculum. The synthesizer was used because of its availability, but other equipment could be used for similar tasks. Since computer synthesized speech was new to most students, the exercise provided a high degree of motivation for the students. As with many of the other exercises, the students were able to determine for themselves when the exercise was successfully completed. They also noted a few of the deficiencies of current computer synthesized speech and were thus introduced to the area of computational linguistics.

In a first programming course or in the beginning of a subsequent programming course for a different language, it is probably unwise to introduce too much material to the students immediately. For that reason, it may be preferable to limit these initial programming exercises to ones which are motivating but do not necessarily include practical knowledge, new technology or applications. Examples of such assignments in the Spitbol course are to

. remove offensive words from a passage (e.g. those referring to the athletic archrival).
. generate a greeting card (varies with the time of year, e.g. Halloween, Easter).
. construct a certificate stating that someone is the [scope of recognition] [level] [category], e.g. World's Fastest Programmer (this assignment also introduces the use of special output forms).
. associate students with one another based on their yes-no answers (input on op-scan forms) to a questionnaire. (The assignment also illustrates another way to enter data into a computer.)
. write a function to determine if a string is a palindrome (e.g. a man a plan a canal panama) and add to the test data if you wish.
. write a program in as few statements as possible (a dubious virtue but in this case encourages the students to consider most of the constructs in the language and offers a challenge).

The programming tasks described in this paper are representative exercises to help students achieve more than a mastery of programming syntax or basic concepts. Many other tasks exist. When interactive computing was new, it was introduced to students with this type of programming exercise. As it became more established it was made an integral part of the curriculum. Other possibilities for such programming exercises include a game playing program to introduce artificial intelligence, or use of a word processing terminal, intelligent terminal, videodisk or typewriter printer to introduce new technologies. In certain respects, the new and sophisticated hardware is like a toy to the students and so provides significant positive motivation. It exposes the students to many areas of computer science and new technologies and it provides them with much practical but non-academic knowledge.

IMPLEMENTATION

Introducing such exercises into a programming language course is not without effort; this reality may explain why few textbooks include them. A few of the example tasks described previously use special equipment which may not be readily available everywhere, but a text could include representative exercises from which an instructor could choose depending on the peripherals available to him. In addition, software beyond the basic programming language facilities must be provided for many of the example programming tasks. With the graphics program, a set of external functions to allow the graphics manipulation and a utility program to allow the resulting structure to be output on a plotting device had to be written. With the screen I/O program, an external function had to be written to interface ASCII terminals with an EBCDIC computer. Even the tape program required that the input be placed in a format appropriate for subsequent manipulation. With faculty as occupied as they already are, where can this time be found? Much of the necessary additional software was written by upperclass students as part of an independent project. The students writing the software obtained valuable experience, and the resulting product provided a useful function, rather than being placed on a shelf to collect dust. Knowing their software would be used made the importance of debugging and testing apparent to the students.

A few of the exercises require hardware that may not be available to all departments, but most departments have some type of special equipment used in research projects or for other purposes. Quite often this equipment is used infrequently by faculty and graduate students, and undergraduates are rarely exposed to it. Since the equipment would be used in most cases for only a single exercise over a period of several weeks, it would often be possible to provide access to the equipment for all students in the class without affecting the primary purpose of the equipment.

Such programming exercises clearly require more programming effort than the conventional exercises listed in Figure 1. What effect does this extra effort have on the students? In the SPITBOL course, the evaluation is that the course does take more effort than similar courses but that the effort is justified by the extra knowledge and experience gained. Since the student does not have to complete every programming exercise, having a choice provides compensation for the additional effort.

CONCLUSIONS

Although the resources available and effort required may be such that not every or even most programming exercises can meet the goals presented in this paper, it would be useful for at least a few exercises to aim toward these goals.

Programming exercises such as those suggested could begin to be sprinkled throughout programming textbooks. To reduce the effort of each individual instructor (why have each person reinvent?), additional software could be provided to the instructors much as instructor's manuals are now. With widely used programming languages such as FORTRAN, the additional software could be written in that language for easy portability. [The special interfaces used in the SPITBOL course, which were written in SPITBOL, FORTRAN and IBM 370 assembler, are available from the author upon request].

The results of this study suggest that programming exercises do not need to be as dry and uninteresting as many currently are. With more interesting and challenging exercises, the student gains beneficial skills. Just as programming exercises were modified to teach more than programming language syntax, they can now be extended even further.

REFERENCES

Csikszentmihalyi, M., Beyond Boredom and Anxiety, Jossey-Bass, 1975.

Malone, Thomas W., "What Make Things Fun to Learn?", Xerox PARC Technical Report, SSL-80-11, 1980.

Papert, Seymour, "On Making a Theorem for a Child", Proceedings of the ACM National Conference, 1972.

Ulloa, M., "Teaching and Learning Computer Programming", SIGCSE Bulletin, July 1980, pp. 48-64.