WRITEACOURSE:
An educational programming language*

by EARL HUNT and MARY ZOSEL
University of Washington
Seattle, Washington

The problem

Computer applications in education are becoming more and more prevalent. Perhaps the most talked about use of computers in the schools is to control the educational material presented to students...the Computer Assisted Instruction (CAI) application. CAI requires that two problems be solved. Someone has to decide what material should be sent to a student, and when, and someone has to arrange that the computer actually do what is desired. The first problem, what should be done, is a topic for educators and psychologists. Our concern is with the second. How can we make CAI a convenient tool for the educator?

We will assume that the educator has access to an interactive system, but that system was not specifically designed for computer assisted instruction. Once the educator has determined the form of a lesson he would like to give to the typewriter, type in the directions, and then leave the typewriter knowing when he returns with a student, the computer will be prepared to conduct the lesson. The problem is that the computer “understands” instructions only in a very restricted set of languages. The form of these languages has, for the most part, been dictated either by the internal design of the machine or by the requirements of mathematicians and statisticians who are, after all, the largest group of users of general purpose computers.

The language problem can be solved on several ways. The educator could, himself, become proficient in computer programming. This diverts his time from the problem he wishes to pursue. He could acquire a specially designed computing system which had languages and equipment suitable for his use. This alternative is extremely expensive (the equipment alone would rent for $100,000 a year or better) and is feasible only for large research projects. He could hire a computer programmer and tell him what the computer was supposed to do. This introduces another specialist into the research team, and has the disadvantage that the computer will then act as the programmer thought the educator wanted it to act. The educator may not discover a misunderstanding until after it has been built into the programming system, at which point it is hard to fix.

We advocate another alternative, placing in the general purpose computing system a language which is easy for the educator to use. This is the solution which was taken over ten years ago by mathematicians, when they were faced with the prospect of writing mathematics in a language which was designed for machine execution, rather than for problem statement. The great success of languages such as FORTRAN and ALGOL testifies to the feasibility of the approach. In the next ten years an educator’s language may also be needed.

What should the characteristics of such a language be? By far the most important requirement is that the language should be natural for the teacher. Its syntax and semantics should conform to his writing habits. Insofar as possible, and there are limits on this, the form of the language should not be determined by the physical characteristics of the computer on which it will be used.

Readability is a second requirement. It will often be necessary for a person to understand a program he did not write. The structure of the programming language should be such that the basic plan of a program can be communicated without forcing the reader to master the intricacies of each line of code.

A judicious choice of a language can also ensure the

*This research was supported by the Air Force Office of Scientific Research, Office of Aerospace Research, United States Air Force, under AFOSR Grant No. AF-AFOSR-1311-67. Distribution of this document is unlimited.
availability of a computer. Any language which is not tied to the physical characteristics of a computer requires a translator. Pragmatically speaking, then, the language is defined by the translator program. Thus the educational language can be "inherited" by any machine for which its base language translator exists.

We are by no means the first to recognize the need for an educator's language. Several others have already been developed. The best known are probably IBM's COURSEWRITER\textsuperscript{2} and System Development Corporation's PLANIT.\textsuperscript{3} These languages are admirably suited for the particular computer configurations for which they were developed. For a variety of reasons, however, we believe that they fail to meet the criteria we have listed. Our principal criticism is that they either are too much influenced by the way a computer wishes to receive commands, instead of the way a person wishes to give them, or that they contain features which, although quite useful in themselves, would not be available except in specially designed computing systems.

The WRITEACOURSE Language.

We have developed an educational language, called WRITEACOURSE, which is consciously modeled after the ALGOL arithmetic programming language,\textsuperscript{7} which it resembles in its syntactic programming structure. The basic unit of discourse is the statement, corresponding roughly to an English sentence. Statements are grouped into larger units called lessons, and lessons into courses, similar to the way a group of subroutines make up a program. Statements are composed of instructions. In WRITEACOURSE there are only ten instructions. Physically, they are English words, such as ADD and PRINT, which have been chosen to have a meaning as close as possible to their meaning in the natural language.

Limiting the commands of the language restricts us. There are actions which can be executed by a computer, but which are difficult to express in a restricted idiom. The initial users of WRITEACOURSE have not found this to be a great problem. They appear to be able to say almost everything they want to say with ease of reference in explaining the lesson. We can test this now by presenting a fragment of a WRITEACOURSE lesson. Hopefully it will be readable with only a minimal explanation.

We assume that the particular configuration has an interactive computing capability, in which the user can exchange messages with a program from a remote station equipped with a typewriter or other keyboard device. By 1970 this sort of capability should be common in universities, at a price well within the reach of a modest research budget.

An earlier version of WRITEACOURSE\textsuperscript{4} was defined for the Burroughs B5500 computer only, using the extended ALGOL provided for that machine.\textsuperscript{1} Thus the early version was not machine independent in the sense that our present program is, although it would be a fairly straightforward task to adopt it to some other computer which had an ALGOL compiler.

Our approach should produce an easily maintained system. This is a very important point. Undoubtedly there will be errors in any system as complicated as a programming language. Also, different users will want to extend the language to suit their own purpose. Since the translation program is written in a commonly available, user oriented language, the educator will find that there are many people who can understand and alter it. This will be particularly true in universities, where Computer Science departments and computer centers will regularly offer undergraduate courses in PL/I programming.

A user's view of the language.

The purpose of developing WRITEACOURSE was to have a language which could be easily understood by educators. We can test this now by presenting a fragment of a WRITEACOURSE lesson. Hopefully it will be readable with only a minimal explanation.

The following statements are taken from a fragment of a WRITEACOURSE lesson. They appear exactly as they would be typed by an instructor, with the exception of the numbers in parentheses at the beginning of each line. These have been introduced for ease of reference in explaining the lesson.

\begin{enumerate}
\item \hspace{1em} 20 \textbf{PRINT \textit{"THE ANGLE OF INCIDENCE IS EQUAL TO THE ANGLE OF ..."}}
\item \hspace{1em} \textbf{ACCEPT CHECK \textit{"REFLECTION" "REFRACTION" IF 1 CHECKS THEN GO TO 5}}
\item \hspace{1em} \textbf{IF 2 CHECKS THEN PRINT \textit{"NO, THE ANGLE OF REFRACITION DEPENDS ON THE TYPE OF LENS"}}
\item \hspace{1em} \textbf{PRINT \textit{"TRY AGAIN" ACCEPT CHECK REFLECTION" IF 0 CHECKS THEN}}
\item \hspace{1em} \textbf{PRINT \textit{"THE CORRECT ANSWER IS REFLECTION"}}
\item \hspace{1em} \textbf{5 PRINT \textit{"HERE IS THE NEXT QUESTION"}}
\end{enumerate}

What would happen when a student executed this lesson? The first statement (statement 20) to be executed is the statement beginning on line (1) and extending to the end of statement marker(\textit{";"}) on line (2). Statements always begin on a new line; otherwise, they may be typed in any way convenient. Line (1) would print the question \textbf{THE ANGLE OF INCIDENCE IS EQUAL TO THE ANGLE OF ...} on the computer-controlled typewriter. At line (2) the ACCEPT instruction would print an underscore (\textit{"_"}) on the next line. This would be a signal to the student indicating...
that an answer was expected. At this point the paper in front of the student would look like this

THE ANGLE OF INCIDENCE IS EQUAL TO THE ANGLE OF ...

The computer would then wait for the student, who would type whatever he thought was an appropriate reply, then strike the carriage return key of the typewriter, indicating that he was through with his answer. The program would ACCEPT this answer, and CHECK it against indicated possible answers. Suppose the student had typed

REFRACTION

The CHECK command on line (2) would match this answer against the quoted statements “REFLECTION” and “REFRACTION.” The quoted statements are called check strings. In this case the answer would be identical to the second check string, so we say that “2 CHECKS.” At line (2), however, the question asked is, “DOES 1 CHECK?” This would only be true if the student had replied REFLECTION (the correct answer), in which case control would have been transferred to the statement named 5, at line (7) of the lesson, which continues with a new question.

However, 1 did not check, so the next commands to be executed are those on line (3), which begins a new, unnamed statement. Lines (3) and (4) are straightforward. The computer asks if 2 CHECKS, which it does, since the student’s reply was identical to the second check string. Upon determining this, the computer types out the correcting response given on lines (3) and (4). Next the statement beginning on line (5) is executed. This prints another line, urging the student to try again, and an underscore (the ACCEPT of line (5)) telling him an answer is expected. The student will now have in front of him

THE ANGLE OF INCIDENCE IS EQUAL TO THE ANGLE OF ... REFRACTION
NO, THE ANGLE OF REFRACTION DEPENDS ON THE TYPE OF LENS

Assume that he replies correctly, printing REFLECTION. This will be read by the ACCEPT statement in line (5) and the immediately following CHECK statement will determine that 1 CHECKS is true. IF 0 CHECKS tests to see if nothing checked, i.e., 0 CHECKS is true if the student’s answer does not match any of the check strings. In this case, the condition 0 CHECKS would be true for any answer other than REFLECTION. Looking at the final three lines of the conversation, we have

TRY AGAIN
REFLECTION
HERE IS THE NEXT QUESTION

But suppose that the student had not been so bright. The final lines could have read

TRY AGAIN
WHO KNOWS?
THE CORRECT ANSWER IS REFLECTION
HERE IS THE NEXT QUESTION

More sophisticated programming

The example just given was very simple. Using the computer’s capabilities more fully, WRITEACOURSE makes possible the specification of a much more complex branching sequence. There is also a limited arithmetical capability. A set of counters (temporary variables) are provided to keep track of intermediate results. Counters can be used either to do arithmetic or to record the number of times a student takes a particular path through a course. This turns out to be a powerful device. We will give a few examples.

Counters are named by preceding a number with the symbol “@.” Thus @10 means “counter 10.” Three commands are defined for counters, SET (counter number) TO (value), ADD (value) TO (counter number, and SUBTRACT (value) FROM (counter number). They have the obvious meaning.

SET @10 to 0 establishes 0 as the value of counter 10, while

ADD 5 TO @10 sets the value of counter 10 to 5 plus its original value.

It takes little imagination to see that the counters can be used to keep scores on a student’s responses, through the device exemplified by

IF 1 CHECKS THEN ADD 1 TO @7.

The value of a counter may also be printed. To do this the name of the counter is included in a PRINT command. When the command is executed, its current value will be printed. The statement

SET @8 TO 5 PRINT “THE VALUE OF 8 IS@8”

will print

THE VALUE OF 8 IS 5.

The content of a counter is a value, so arithmetic can be done on counters. ADD @2 TO @3 would set the value of counter 3 to the original value of counter 2 plus the value of counter 3.

There are actually three groups of counters. Counters
50–99 are lesson counters, their values are carried over from one use of a WRITEACOURSE lesson to another. There are several reasons for doing this. For instance, a counter can be used to keep track of the number of students executing a lesson, or the number of students who miss a particular question. Counters 1 to 49 are the temporary counters. They are set to zero when a student first signs in for a session with the computer. They are retained for that student, however, for the duration of the session even if he switches WRITEACOURSE lessons. Finally, Counter 0 is a special counter set by the computer's internal clock. It can be used to time a student's responses.

A set of Boolean IF statements are provided to check the value of a counter against another counter, or some constant value. The command IF @4 = 7 THEN GO TO 6 will cause a transfer to statement 6 only if counter 4 contains 7. The normal arithmetical relations of equality and ordered inequality are permitted.

Counter numbers may also be used for a computed GO TO. GO TO @2 is an instruction to go to the statement whose number is contained in counter 2. Of course, the instructor who writes this command must ensure that counter 2 will contain the name of a statement whenever this command is executed.

Let us look at an example which uses some of these more complex commands.

(1) SET @54, @41 TO 0 PRINT "WHAT DISCOVERY"
(2) LEAD TO LASERS?"
(3) ACCEPT "MASER" "QUASER" "CANDLES" IF 1 CHECKS THEN GO TO 6
(4) ADD 1 TO @41 IF 0 CHECKS THEN GO TO 40
(5) IF 2 CHECKS THEN PRINT "THAT IS IN ASTRONOMY."
(6) GO TO 40
(7) IF 3 CHECKS THEN PRINT "DO NOT BE SILLY.""
(8) 40 IF @41 < 3 THEN PRINT "TRY AGAIN"
(9) GO TO 3
(10) ADD 1 TO @54 PRINT "THE ANSWER IS MASER."
(11) PRINT "HERE IS THE NEXT QUESTION"

The first statement sets counters 54 and 41 to zero, then prints the basic question. Statement number 3 through statement number 40 establish a loop, which checks the student's answer for the correct answer or two anticipated wrong answers, prints an appropriate message for a wrong answer, then gives the student another chance. If the correct answer is detected (if 1 CHECKS in line (4)), the loop is broken by a transfer to statement 6. If a wrong answer is detected, the question is reasked. Counter 41 is used to keep track of the number of wrong answers. If three wrong answers are given, the correct answer is printed, and the program continues on. If this alternative occurs, however, the value of counter 54 is incremented by 1. Recall that counter 54 is one of the lesson counters, i.e., its value carries over from one user of the lesson to another. At some later time, then, an instructor could interrogate the lesson to see how many students had failed to answer this question in three or fewer tries.

Lessons and courses

Statements are grouped into lessons, and lessons into courses. Roughly, a lesson can be thought of as the number of WRITEACOURSE statements needed to carry on the computer's part of a computer-student interaction lasting about half an hour. Another important functional distinction is that a lesson is the WRITEACOURSE unit to which counters are attached. Thus if @54 appears in two different statements in the same lesson, it refers to the same counter. If the two statements are in different lessons, they refer to different counters. Note that this is not true for temporary counters, since they remain attached to a student for the duration of a student-computer conversation. Thus if it is anticipated that a student will use more than one lesson during a single session, the results accumulated while the first lesson is active may be communicated to the second lesson via the temporary counters.

Lessons themselves are grouped into courses. Functionally, the chief distinction of a course is that it is possible to activate one lesson from within another, providing that the two lessons are in the same course. Suppose a student signs in, with the intention of taking a course in Romance Literature. He would begin by indicating that he wanted to work on the first lesson of this course. He would do this by replying, in response to a computer question, that he wished to work on LESSON1/LIT 47. LIT47 is assumed to be a course name, and LESSON1 a lesson of the course. Let us suppose that this lesson is going to discuss the novel Don Quijote. The instructor might want to check to make sure the student knew enough Spanish to understand some of phrases. This can be accomplished by the following statement.
(1) 1 PRINT "DO YOU WISH TO REVIEW SPANISH?"

(2) ACCEPT CHECK "YES" IF 1 CHECKS THEN CALL SPREVUE/LIT47 |
If the last command on the second line is activated, it will suspend the current lesson now active (LESSON1/LIT47), and load the lesson SPREVUE/LIT47. Both lessons must be in the same course. Upon completion of SPREVUE/LIT47, control would be returned to the statement in LESSON1/LIT47 immediately after line (2).

The command LINK (lesson name) / (course name) will also change a student from one lesson to another within the same course. In this case, however, there is no automatic return to the calling lesson after the called lesson is completed. The normal use of LINK is to string together several lessons which the instructor wishes to have executed in sequence.

**Using WRITEACOURSE**

The steps in using WRITEACOURSE will now be described. The steps a student must go through to initiate a lesson have been kept to a minimum. He types XEQ and then supplies the lesson name and course name when requested. After a lesson is over, he may type XEQ and go through another lesson, or type STOP to terminate the session.

When an instructor constructs a lesson, the process is necessarily more involved. After calling the system the instructor sends the message /// COMPILe indicating a course is to be established or modified. (In general, the symbols "///" precede compiler commands.) If a new course is to be written, the order is sent.

///PROGRAM NEW lesson/course

The translator will then be ready to accept the lesson. Each statement is checked for syntax errors as it is received. If there is no error, the next statement is requested. Whenever an error is detected, a message is printed indicating where it occurred. After determining the corrected form, the instructor re-enters the statement, from the point of the error to the end. When the instructor wishes to stop working on the lesson, he types ///END. The lesson will be automatically stored in the computing system's files. If the instructor desires, he may order a check for undefined statement numbers referenced by GO TO instructions before the lesson is recorded.

The instructor may modify existing lessons or obtain a listing of lessons, using the commands ///ADD, ///DELETE and ///LIST.

**System implementation.**

WRITEACOURSE has been tested on an IBM 360/50 with a remote 2741 terminal. The translator was written in the RUSH4 subset of PL/I, provided by Allen-Babcock Computing (8). The only non-standard PL/I used is the timer function. WRITEACOURSE lessons are incrementally compiled into a decimal integer code, which is stored in a data file. The storage file for each course consists of 64 tracks of fixed format data with a block size of 252 bytes. The internal code is edited whenever a teacher makes a modification. The execution program interprets this code to produce the sequence of events planned by the instructor. The first block of code in a course contains the names and locations of the lessons in it. Each lesson occupies 38 blocks of the file, and is divided into five parts.

1. The instruction table, which contains the compiled decimal code with approximately one code word for each instruction in the lesson.
2. The statement number table, which contains the statement numbers with a pointer to the corresponding instructions.
3. The counters attached to the lesson.
4. The print tables, which contain all of the strings to be printed.
5. The print table index, which contains a pointer to the location of each string.

Since the source code is not saved, the compiled code must be used whenever the lesson is changed. To obtain a listing of the lesson, the code is interpreted, as if it were to be executed, and the source code is reconstructed. When a section of a lesson is deleted, the instruction table and the statement number table are closed up to eliminate the desired portion. The strings in the print tables are marked inactive, for later garbage collection. Code is added to a lesson by opening a hole in the instruction table and statement number table of the proper length, and then inserting the compiled code. New print strings are added to the end of the print tables.

A pointer is kept in each table to indicate the last entry in the table, so that new code can be added to the end of a lesson. Source code is input to the compiler one statement at a time. The compiler analyzes the statement instruction by instruction. If it detects any errors it requests that the user re-input the statement from the instruction containing the error to the end.

WRITEACOURSE is broken into several programs in order to fit within the limited computer space available in a time-shared system. The programs operate as overlay segments, with PL/I external variables used to communicate between them. The modular structure of
WRITEACOURSE should facilitate system additions or modifications. Figure 1 shows the basic overlay structure. The functions of each program are indicated in the figure.

**FIGURE 1**

- **COMMON**
  - Handle file operations and lesson modifications

- **STORAGE**
  - Contains code for lessons

- **EXECUT**
  - Interprets the internal code and handles student responses

- **CONFIL**
  - Accepts lesson statements from user, checks, and produces internal code

- **EXCH**
  - (Execution Monitor)
  - Set up lessons for execution and handle file operations

**Status**

The earlier ALGOL version of WRITEACOURSE has been successfully used by people with little programming experience. Although the current version, at the time of this writing, has not yet been put into general use, the programming is completed. A limited number of manuals describing the language details and use of the system are available from the Department of Psychology (Cognitive Capabilities Project), the University of Washington. Listings of the translator and manuals will be provided upon request and at cost.

**REFERENCES**

1. Burroughs B5500 information processing systems extended Algol language manual
   Burroughs Corporation 1966

2. S. L. Feingold, C. H. Frye
   User's guide to PLANIT
   System Development Corporation 1966

3. J. Feldman, D. Gries
   Translator writing systems
   Comm ACM Vol 11 Feb 1968

4. S. Hendrickson, E. Hunt
   The WRITEACOURSE language programming manual
   Department of Psychology University of Washington 1967

5. IBM 1500 operating system computer-assisted instruction course writer II
   Form CAI-4036-1 IBM

6. IBM operating system/360 PL/I: language specifications
   Form C28-6571-2 IBM 1966

7. P. Naur (Editor)
   Revised report on the algorithmic language Algol 60
   Comm ACM Vol 6 pp 1-17 Jan 1963

8. RUSH terminal user's manual
   Allen-Babcock Computing Inc 1966

**FOOTNOTES**

1. This research was supported by the Air Force Office of Scientific Research, Office of Aerospace Research, United States Air Force, under AFOSR Grant No. AF-AFOSR-1311-67. Distribution of this document is unlimited.

2. We wish to express our thanks to Sidney Hendrickson for his comments and work on an earlier version of the language.

3. There is an unfortunate ambiguity in the word “program”, since it is used by educators to mean a sequence of interchanges between student and teacher, and by computer scientists to mean the sequence of commands issued to a computer. We shall use “lesson” when we mean “sequence of educational steps” and “program” when we mean “sequence of commands to be executed by a digital computer.”

4. At this point the mind of people not familiar with modern computer technology tends to swim. It is possible to carry this process even further (3).

5. The statement had to end at line (2) because of the IF-THEN command. The general rule is that when a question of the form IF condition THEN is asked, the commands between the word THEN and the next | are executed only if the condition is true. If it is false, as it is in this case, the command immediately following the |, i.e., the first command of the next statement, is executed.

6. More complicated matches are possible, which do not require exact identity. For instance, it is possible to ask if a check string is included anywhere in an answer, so that, in this case 2 would check if the answer had been IT IS REFRACTION.

7. A manual describing the language in detail is available.