

1995-12-01

## Writing an Authoring CALL Program

Phan Banpho

Follow this and additional works at: <https://digital.car.chula.ac.th/pasaa>



Part of the [Reading and Language Commons](#)

---

### Recommended Citation

Banpho, Phan (1995) "Writing an Authoring CALL Program," *PASAA*: Vol. 25, Article 2.

DOI: 10.58837/CHULA.PASAA.25.1.2

Available at: <https://digital.car.chula.ac.th/pasaa/vol25/iss1/2>

This Original Article is brought to you for free and open access by the Chulalongkorn Journal Online (CUJO) at Chula Digital Collections. It has been accepted for inclusion in PASAA by an authorized editor of Chula Digital Collections. For more information, please contact [ChulaDC@car.chula.ac.th](mailto:ChulaDC@car.chula.ac.th).

---

## Writing an Authoring CALL Program

---

Phan Banpho  
Chulalongkorn University Language Institute

### ABSTRACT

Authoring programs are very useful because they offer language teachers and those involved in language teaching a convenient means of constructing exercises and tests. This article encourages language teachers who have some experience in programming to write an authoring program. It discusses important components of the authoring module, particularly the input, editing, data and information saving. This article also covers practical steps and techniques, including how to retrieve the data input by the teacher and how to mark students' answers.

### INTRODUCTION

The first half of the current decade has seen a marked increase in the use of instructional software for language teaching and learning, especially the use of computer-assisted language learning (CALL) programs. These programs are either dedicated or authorable. Authoring programs are more widely used and often more useful than the dedicated or templated programs. One of the most obvious reasons is that they provide teachers with an easy and non-technical way of writing new data (Jones and Fortescue 1987: 42). In other words, they enable teachers with no

programming experience to construct exercises or tests that suit their students. Despite this fact, there are few authoring CALL programs written by language teachers who teach a second or foreign language. In addition, after half a decade's experience in teaching and conducting workshops on using microcomputers in language teaching, we can affirm that many authoring programs are not always user-friendly. Some are very demanding while others ask you to do a number of unnecessary tasks. The aim of this article, therefore, is to encourage language teachers with some program-

ming experience to write an authoring CALL program. It focuses on how to get started, the main components, essential steps and techniques, the writing of the delivery module, and some considerations on writing an authoring program. The sample programs are written in QuickBASIC.

## HOW TO GET STARTED

One of the most practical ways to begin writing an authoring CALL program is to think about the student program, rather than the authoring program, because it will determine what the authoring program will look like and how it will run.

### *The student's module*

You need to visualize what the student's program, or module, will look like and how it will work, because when you know this, you will know what to include in the teacher's module, or the authoring module.

You may begin by thinking about the general purpose of the student's program, which may derive from your need to help students and other language teachers to correct students' language problems, or to enhance their ability to communicate more efficiently in the target language. This general purpose will act as a guideline that can help you define specific objectives, content, screen design, students' interaction with the program, and a report on the students' performance after they have used the program.

The next step is to write a prototype, which is a working model of the actual program. This prototype should contain a sample of everything in the program (Kearsley 1986: 28). To do this, you need to search for relevant language content and write the student's module, using the computer language with which you are most familiar. You should spend time writing and revising this prototype until you are satisfied with it. If possible, you should try it out with the target students to observe how they learn from the program and how they react to the program. You may have to sit with the students and observe any difficulties that might arise, including both errors and unexpected problems. It may also be a good idea to try out the program on some of your colleagues who will give sincere comments and suggestions on the

program. After that, revise the prototype of the student's program. If possible, subject it to another trial run.

When designing the student's program, you may also design a report on students' performance. The report should be displayed on the screen after each exercise has been done, and it would be a good idea to have the report printed out. The teacher can then give further advice to the students. See Appendix One for an example of such a report.

### *The teacher's module*

When you are satisfied with the student's module, you will be ready to consider the teacher's module, or the authoring module. You should use the information obtained from the writing of the student's module and from the trial run to help you visualize the teacher's program. The visualization should include the content, screen design, input/output of the teacher's responses, and how the teacher can edit the data or information he or she has input into the authoring program.

## MAIN COMPONENTS OF AN AUTHORIZING MODULE

To write an authoring program, you need to learn some of its indispensable components. The following are some of them.

### *1. Input component*

The first is an input component, which is a component that allows the teachers to input their data or information, often through the keyboard. The data and information may consist of a text to read, answers, explanations, suggestions, or hints. This component should be carefully designed so that language teachers with no programming experience can type in their data or information without difficulty and frustration. In this component, you have to write an input statement, e.g. LINE INPUT (Norton and Holzner 1991: 5) in BASIC, or open an output file to allow for the input of data. In this portion of the program, you have to deal with the most basic types of variables, string and numeric variables, which are a crucial concept in programming (Higgins and Johns 1984: 103). You need to assign a set of characters to represent both

a string variable and a numeric variable. A string variable is used to store the text or passage and a numeric variable to store a numeric value. Study the following sample program in QuickBASIC, which demonstrates how the INPUT statement is used:

```
CLS
COLOR 15, 0
LOCATE 2, 5 : PRINT "Type a sentence in the space provided."
COLOR 14, 0
LOCATE 3, 5: LINE INPUT "", sent1$
```

The command in the last line will wait until the teachers have input their sentence and the Enter key has been pressed.

## 2. Editing component

An authoring program also requires an editing component, which allows teachers to correct or edit the words, sentences, or passages they have already input. This component is very important due to typing errors or your decision to change a word, phrase, or sentence. In this component, the original text will be displayed and the teachers will be allowed to enter new data or corrections. This component, in fact, is like an editor in a word processing program. That is, you may design this component so that teachers can edit their text inside the authoring program they are using. However, this also depends on your programming ability. If you are a real novice programmer, you may construct a line editor, which can edit only one specific line at a time. In a line editor, you cannot use an up- or down-arrow key to move the cursor to the upper or lower line. However, since this type of editor is also complicated, it will not be discussed in this brief article. You can use a ready-made line editor that comes with books about programming techniques and utilities.

If you design an input component that opens an output file and allows teachers to type in their text using a word processing program, then the editing component will be easy for you because you will not have to write an editor for inputting and editing text. However, it may be difficult for the teachers because they will have to leave the authoring program and use a word processing program. This requirement may discourage some

language teachers who are not familiar with any word processing program.

One important drawback of inputting data or information from a word processing program is that the text will be a text file, which can often be read by using the *Type* statement of the Disk Operating system (DOS). Thus, those students who learn this command can look at the answers before they do the exercise or the test. However, to prevent students from accessing the answers, we can encrypt, or transform each character in the answers in a systematic way and write a program to decrypt these transformed data into the original data (Kenning and Kenning 1983: 126). For example, the character *a* can be encrypted into */* in the data file, then decrypted into the character *a* again.

## 3. Data and information saving component

This is another important component because data and information input by teachers must be saved so that it can be retrieved when needed. No matter what editor you provide for the teachers to input and edit their data, you need to open one or more files to record it. For example, you need to open a file to store the reading text, a file to keep the questions and answers, a file in which to keep a record of students' scores, and a file in which to keep the teachers' personal data, such as their names. Study the following sample program, which can save a text input by the teachers. This sample program can save a text of 15 lines.

```
OPEN "text1" FOR OUTPUT AS # 10
FOR a = 1 to 15
WRITE # 10, Line1$(a)
NEXT a
CLOSE # 10
```

This component is often filled with a number of variables, both string and numeric variables. Thus, you have to name each variable carefully so that you can remember them all. If you are afraid that you will not be able to remember them, write comments or remarks. (In QuickBASIC, the comment or remark follows an apostrophe, e.g. 'ans1\$ = answer1\$'). Another technique is to name a variable according to its function, e.g. hint1.1\$ for a hint provided for the first answer. When dealing

with this component, make sure that you spell each variable correctly and consistently, otherwise you will not be able to retrieve the data stored in each file. You should also be consistent in using upper-case and lower-case letters because some computer languages are case-sensitive. For example, the ASCII code for the capital "A" is 65 while that for the small "a" is 97. This results in the difference between Answer\$ and answer\$.

#### 4. Help component

Since the main aim of a user-friendly authoring program is to encourage language teachers to construct their own lesson by using a ready-made program, the help component is crucial. There should be an adequate supply of relevant help messages. In fact, help should be provided without being asked. For example, when teachers enter a wrong type of variable, there should be a message displayed on the screen telling teachers that it is a wrong type of variable and suggesting what the correct one should be. The help message should be precise and concise.

#### STEPS AND TECHNIQUES

After you have visualized the authoring module, you may construct a flowchart to demonstrate a step-by-step progression of the program. Then, follow these essential programming steps and techniques.

1. Open an output data file to let the teachers input (or type in) personal information, such as their names, and to write (or store) that data to the disk. This information will be displayed in the student's module so that students will know the author of that particular exercise or test. Since there are several types of files, and each has a different way by which it can be opened and read, you may begin with the easiest type — a sequential file, which is a text or ASCII file — because you do not have to deal with records or fields to any great extent. In this type of file, data is stored as a single line of text and terminated by a carriage-return and a line-feed (Microsoft 1988: 100). The input command is used in this file. Study the following example:

```
CLS
OPEN "teacher" FOR OUTPUT AS # 1
LOCATE 8, 15 : PRINT "Please type your full name."
LOCATE 9, 15 : LINE INPUT "", tname$
CLOSE # 1
```

In this sample program, the teacher will be asked to input his or her name. Then, the name may be stored in a string variable called tname\$, after which this variable will be saved to a file e.g. "teacher1". How to save the input data has already been discussed in the Data and Information Saving component.

2. Open an output file for teachers to input their text, which may consist of a single sentence or an entire passage. You may assign a string variable named Line1\$(j) to store each line of the teachers' text. Study the following example:

```
1.1
OPEN "text1" FOR OUTPUT AS # 2
LOCATE 1, 15 : PRINT "Please type your text."
lin = 2
FOR j = 1 TO 12
1.2
LOCATE lin, 2 : LINE INPUT "", Line1$(j)
lin = lin + 1
j = j + 1
IF j < 13 THEN
GOTO 1.2
ELSE
END IF
NEXT j
CLOSE # 2
```

In this sample program, you can input a passage containing 12 lines.

3. Open an output file to allow teachers to input the questions, the best answer, and some possible answers (e.g. eight answers). In this file, you need to assign string variables to store the question, the best answer, and any possible answers. For example, you may assign quest1\$ for the first question, correct1\$(c) for the best answer and any possible answers. The variable correct1\$(1) can be assigned for the best answer. Study the following example:

```

1.3 OPEN "correct1" FOR OUTPUT AS #3
    COLOR 3, 0
    LOCATE 13, 2 : PRINT "What is the paragraph generally
    about?"
    COLO 13, 0
    LOCATE 14, 2 : PRINT "Please type 8 possible answers,"
    LOCATE 14, 33 : PRINT "beginning from the best answer."
    row = 15
    FOR c = 1 TO 8
1.4 LOCATE row, 2 : PRINT "Ans"; c; : PRINT ":"; : LINE IN-
    PUT ""; Correct1$(c)
    row = row + 1
    c = c + 1
    IF c < 9 THEN
    GOTO 1.4
    ELSE
    END IF
    NEXT c
    CLOSE #3

```

The output in the authoring module may look like this:

Passage 1

Eat a biscuit or a slice of bread and make note of how you go about it. But this is only the start of a long and complicated process of digestion. There are a number of stages in the process of digestion. First, the food is broken up by the teeth, and the tongue then helps to mix the food with the saliva. In the saliva there is an enzyme called ptyalin, which changes sugar into starch. The salivary glands are continuously secreting saliva into the mouth. Have you noticed how active they are when you smell something good? Your mouth waters.

1. What is the paragraph generally about?

Please type 8 possible answers, beginning from the best answer

Ans 1: digestion

Ans 2: food digestion

Ans 3: process of digestion

Ans 4: complicated process of digestion

Ans 5: long and complicated process of digestion

Ans 6: a long and complicated process of digestion

Ans 7: stages in the process of digestion

Ans 8: —

4. Open an output file for the teachers to input answers that are very close to the correct ones, but are still unacceptable. In this case, the teachers need to give explanations, suggestions, or hints so that students will not repeat the same mistakes. In this file, you may assign string variables to store

the answers and the explanations. You may, for example, assign wrong1\$(w) for all answers specified as wrong, and explain1\$(e) for explanations, suggestions, and hints. Study the sample program:

```

1.5 OPEN "wrong1" FOR OUTPUT AS #4
    LOCATE 16, 2 : PRINT "Please type 5 wrong answers that
    need explanations."
    FOR w = 1 TO 5
1.6 COLOR 7, 0
    FOR row = 17 TO 18
    LOCATE row, 2 : PRINT ""
    NEXT row
    LOCATE 17, 1 : PRINT "Wrong"; w; : LINE INPUT "";
    wrong1$(w)
    LOCATE 18, 1 : PRINT "Expl. "; w; : LINE INPUT "";
    explain1$(w)
    w = w + 1
    IF w < 6 THEN
    GOTO 1.6
    ELSE
    END IF
    NEXT w
    CLOSE #4

```

The output in the authoring module may look like this:

Passage 1

Eat a biscuit or a slice of bread and make note of how you go about it. But this is only the start of a long and complicated process of digestion. There are a number of stages in the process of digestion. First, the food is broken up by the teeth, and the tongue then helps to mix the food with the saliva. In the saliva there is an enzyme called ptyalin, which changes sugar into starch. The salivary glands are continuously secreting saliva into the mouth. Have you noticed how active they are when you smell something good? Your mouth waters.

1. Please type five wrong answers that need explanations

Wrong 1: buscuit

Expl. 1: SORRY, PLEASE READ LINES 2 AND 3

## WRITING A DELIVERY MODULE

An authoring program is specially designed for teachers. Therefore, to run or present the exercise or test constructed by teachers, you also need to write a delivery module. In fact, this module was written as a prototype program before you began writing the authoring program. However, that program is not a full program and was not designed to accept data and variables assigned in the

authoring program.

### *How to retrieve the data input by the teachers*

In order to retrieve the data input by the teachers, you need to open several input files to read the data. If you do not open input files, you will not be able to use the data prepared by the teachers. These files are often opened at the beginning of the delivery program because once they are opened, all data will be available and ready for use. However, they can also be opened whenever they are needed.

To open such files, you may adapt the following sample programs to suit your own individual needs:

1. To read the name of the teacher who constructed the exercise

```
OPEN "teacher1" FOR INPUT AS #1
  INPUT # 1, tname$
CLOSE # 1
```

2. To read the text or passage

```
OPEN "text1" FOR INPUT AS #2
DO UNTIL EOF (2)
  FOR a = 1 to 15
    INPUT # 2, Line1$(a)
  NEXT a
LOOP
CLOSE # 2
```

3. To read the correct answers

```
OPEN "correct1" FOR INPUT AS #3
DO UNTIL EOF (3)
  FOR c = 1 to 8
    INPUT # 3, correct1$(c)
  NEXT c
LOOP
CLOSE # 3
```

4. To read the wrong answers and explanations

```
OPEN "wrong1" FOR INPUT AS #4
DO UNTIL EOF (4)
  FOR w = 1 to 5
    INPUT # 4, wrong1$(w)
  NEXT w
  FOR e = 1 to 15
```

```
    INPUT # 4, explain1$(e)
  NEXT e
LOOP
CLOSE # 4
```

### *How the student's answers are marked*

This seems complicated, but once you know the technique it is not too difficult. First, you should know that computers do not actually mark or correct the student's answers but rather they compare each answer with those input by the teacher. If they match, then the student's answer is accepted. If they do not match, then the answer is rejected. Thus, before allowing students to enter their answers in the delivery program, you need to open data files to read the answers set by the teachers, both the correct answers and those that are incorrect. These answers will then be compared with those given by the students.

As regards the decision structures, especially the classic IF ... THEN structure, you need to refer to the correct variables storing the correct answers and the wrong ones. Then, suggest that the computer compare the existing variables with those storing the student's answers. If they match, then display congratulatory messages and record the scores. If they do not match, then display hints, suggestions, or explanations. Study the following sample programs.

1. To evaluate the correct answers

```
FOR c = 1 TO 8
  1.1
  IF ans1$ = correct1$(c) THEN
    LOCATE 18, 20 : PRINT "VERY GOOD."
    SLEEP 2
    GOTO 1.2
  NEXT c
  ELSE
    GOTO 1.1
  END IF
```

2. To evaluate the wrong answers

```
FOR e = 1 TO 5
  1.1
  IF ans1$ = wrong1$(e) THEN
    LOCATE 18, 5 : PRINT explain1$(e)
    SLEEP 2
```

```

GOTO 1.2
NEXT e
ELSE
  GOTO 1.1
END IF

```

As for the rest, it remains the same as in an ordinary dedicated CALL program. This means that you have to write other portions of the delivery program.

### SOME CONSIDERATIONS FOR WRITING AN AUTHORIZING PROGRAM

To make your authoring program run smoothly and look professional, you may find the following considerations useful.

1. Since computers are a highly visual medium, the screen display should be carefully designed because stimulating displays will often contribute to holding a student's attention and enhance his or her interest (Kearsley 1988: 6). They can also facilitate learning. You should try to use your knowledge of art, especially composition and balance, to help you design different portions of the screen.

2. For colors, remember that colors that are too bright or too dim may have adverse effects on students' eyes. Thus, avoid displaying a long text in these colors. To attract students' attention, you may make a word, phrase, or sentence blink (Cox and Sullivan 1986: 435) or you may use different size characters. Also, it is a good idea to let the users choose both the foreground and background colors. This can easily be done. You simply assign numeric variables to the number of colors, and vary them according to the colors chosen by the users. In addition, it is also necessary to consider consistency of screen design and the use of color (Clarke 1989: 31) because there is often great temptation to overuse color and different screen formats.

3. When asking the teacher to choose an option or response from the list given, you should decide whether it is necessary to ask them to press the Enter key after they have typed such a response. If it is unnecessary, then use a program function that continues your program without

waiting for the Enter key to be pressed. For example, in QuickBASIC you can use the `inkey$` function. However, if a decision is very important, or if it may branch off to a different portion of the program, then the use of this type of function is not appropriate. For example, if you ask teachers to choose between different types of questions, then it may be better to ask them to type their responses and press the Enter key.

4. Reading from a computer screen is sometimes more difficult than reading from a page in a book due to the brightness of the screen. We also have to concentrate more when reading from the screen than from a page in a book. Thus, avoid crowding the screen. Instead, you should split a long text into several portions and display each portion sequentially.

5. Because different people read at different speeds, try to let the teachers and the students set their own pace. In this regard, therefore, it is common to use the *Press any key to continue* procedure. This means that the user can read a message or a text as long as he or she wishes.

6. Sound effects and background music can make the program more interesting, but they may also distract the user's attention. Thus, it would be better to let students choose whether they need sound effects or not. To do this you need to assign a variable, often a numeric variable, for storing the user's response.

Writing an authoring program is interesting, but you need to have adequate knowledge and programming experience. This article has suggested some ideas that may help you learn how to begin and how to deal with different components or portions of an authoring program. It also provides sample programs, which can be modified to suit your own particular needs. In addition, it has pointed out that one of the most important tasks in writing an authoring program is to open data files for teachers to input and save their data. However, this article is far from comprehensive. Teachers wishing to write authoring CALL programs should study more about file management and some of the other techniques that can make their program more interesting and more useful.



## REFERENCES

- Clarke, D.** 1989. Design considerations in writing CALL software, with particular reference to extended materials. In Cameron, K. (Ed.), *Computer Assisted Language Learning*. Oxford: Intellect.
- Cox, M. J. and Sullivan, K. B.** 1986 *Structuring Programs in Microsoft BASIC*. Boston, Massachusetts: Boyd and Fraser.
- Higgins, J. and Johns, T.** 1984. *Computers in Language Learning*. London: Collins Educational.
- Jones, C. and Fortescue, S.** 1987. *Using Computers in the Language Classroom*. London: Longman.
- Kearsley, G.** 1986. *Authoring: A Guide to the Design of Instructional Software*. Reading, Massachusetts: Addison-Wesley.
- Kenning, M. J. and Kenning, M-M.** 1983. *An Introduction to Computer Assisted Language Teaching*. Oxford: Oxford University Press.
- Microsoft.** 1988. *Programming in BASIC: QuickBASIC 4.5*. New York: Microsoft Corporation.
- Norton, P. and Holzner, S.** 1991. *Advanced BASIC*. New York: Brady.

APPENDIX ONE — STUDENT'S REPORT

REPORT AND EVALUATION

NO.

ID. NO:

NO:

COURSE:

GROUP:

DATE:

TIME:

1. PASSAGE NO:

2. SCORE:

Total score	— 30 Marks (5 questions)
Your score	— 00
Percent	— 00.00%

3. PROGRAM'S COMMENT:

4. QUESTIONS NOT ANSWERED CORRECTLY:

5. STUDENT'S SIGNATURE: \_\_\_\_\_  
( \_\_\_\_\_ )

\*\*\*\*\* FOR TEACHER ONLY \*\*\*\*\*

6. TEACHER'S COMMENTS / SUGGESTIONS / ASSIGNMENTS:

- ☐ 1. See me on \_\_\_\_ / \_\_\_\_ / 199\_\_ at \_\_\_\_ Room
- ☐ 2. Study from computer program(s) : \_\_\_\_\_
- ☐ 3. Study from videotape : \_\_\_\_\_
- ☐ 4. Do exercise \_\_\_\_\_ on page \_\_\_\_\_
- ☐ 5. Read \_\_\_\_\_
- 6. \_\_\_\_\_

7. TEACHER'S SIGNATURE: \_\_\_\_\_  
\_\_\_\_\_ / \_\_\_\_\_ / 199\_\_