Published in Bulletin CILA (Commission interuniversitaire suisse de linguistique appliquée
(« Bulletin VALS-ASLA » since 1994) 46, 26-37, 1987, which should be used for any reference to this work

Programming languages, author languages and authoring packages

Computers are useless without programs, but programming them can take an inordinate amount of time – much more time than the average teacher has available. My own experience suggests it takes about one hundred programming hours to write a new program which will keep the student busy for about one hour. Nevertheless, it may be essential for teachers to create their own CALL materials if there is nothing available commercially. The teacher may consider three possible ways of creating CALL software: [1] with a programming language, [2] with a dedicated author language, [3] with an authoring package.

A computer system can be broadly divided into hardware and software. Software can be subdivided into programs and data. Of these three elements data is unquestionably the most important – especially as far as the language teacher is concerned. Unfortunately, however, the people who take the decisions about computing in education often have their priorities upside down. They consider the hardware first, when really they should consider the software, as the effectiveness of any computer depends entirely upon the range and quality of the software that runs on it. If the available software includes a lot of ready-made data then so much the better, but the teacher venturing into CALL should also be given the opportunity to use the computer as a data-creation tool. This is why the programming languages with which it is provided and the authoring possibilities it offers are so important.

Virtually all microcomputers are capable of «understanding» the programming language known as BASIC. BASIC – or rather the BASIC interpreter – is actually a very clever piece of software, the function of which is to translate BASIC instructions into a special kind of machine language. Machine language is not easy for human beings to come to terms with, but BASIC is relatively easy to learn. So BASIC acts as a sort of lingua franca for man and machine. BASIC is an acronym, standing for Beginners' All-purpose Symbolic Instruction Code. It began as an instructional language for training beginners in computer programming, who would then move on to more complex languages, but now it is a fully-fledged programming language in its own right. A set of instructions written in BASIC forms a program, which can be typed at the computer keyboard and eventually saved on disk and called up whenever it is needed.

BASIC is unpopular amongst many professional programmers. It is said to be an «inelegant» language and criticised for its lack of structure. Other languages, such as PASCAL or LOGO, tend to be favoured by professionals. The fact is, however, that BASIC is by far the commonest microcomputer language and the easiest to access. Most commercial programs for microcomputers are written in BASIC because it is a standard feature on virtually all machines. If you wanted to run programs written in LOGO, for example, you would probably have to pay a good deal extra for a special LOGO chip in order to make them work. This is one reason why producers of commercial software avoid writing programs in less common languages; if customers had to buy extra chips in order to use them, sales would plummet.

In an ideal world, programmers and linguists would sit down together and collaborate on programming projects. But programmers are expensive and not many educational institutions can afford enough of them to satisfy the demand for new software. There is no reason, however, why a linguist should not become a competent programmer, given the motivation and time. The band of polymaths bridging the two disciplines is small but growing. Even an elementary knowledge of BASIC will enable the linguist to understand some of the problems faced by programmers and perhaps modify unreasonable demands. Not that one has to learn how to write a new program from scratch. Simply knowing how to amend the BASIC instructions containing text is useful. This is the key to translating simulations and adventure games – which consist mainly of text – into different languages. It takes time to achieve real proficiency, however, and one should beware of glib salesmen who claim BASIC can be learned in «just a few hours». This is no more true of BASIC than it is of French or German.

BASIC has many similarities to natural language, one of them being most infuriating: there is no standard version but a multitude of dialects. Microsoft BASIC is the commonest dialect on microcomputers. It is important that the user is aware that there are differences between these dialects and that not all versions of BASIC are suitable for CALL programs. Of all the current versions of BASIC, the version running on the BBC microcomputer is probably the best as far as the linguist is concerned.

Learning BASIC requires an introductory course of some 20–30 hours, followed by hundreds of hours of practice. Courses designed for people from the non-mathematical disciplines are still relatively rare. Anyone with a background in language teaching embarking upon a course designed mainly for mathematicians or scientists will probably find it impossible to follow, not because BASIC itself is difficult but because the programming tasks to which it is applied will be incomprehensible. This is why a course tailored to the needs of a linguist is essential. Although there are still relatively few courses appropriate to linguists, there are a number of

textbooks now available which are aimed at the non-mathematician. A selection is listed at the end of this article.

The following program gives an indication of what BASIC looks like. It is a simple gap-filling exercise, the logic of which is more or less self-explanatory. A question is presented to the user (lines 10–30); the user enters a response at the keyboard (line 40); the response is matched with the correct answer (line 50); if it is incorrect then the program branches to line 80; if not, the program continues with line 60. Finally the program stops at line 90.

```
10 PRINT «Type the correct form of 'spielen' in»
20 PRINT «the Present Tense:»
30 PRINT «Mein Bruder ------ Tennis.»
40 INPUT A$
50 IF A$ = «spielt» THEN 80
60 PRINT «Sorry, wrong, the answer is 'spielt'.»
70 GOTO 90
80 PRINT «Well done!»
90 END
```

Although BASIC is the commonest microcomputer language, it suffers from the weakness that it attempts to be all things to all men. It is a general-purpose programming language. If you aim to produce CALL programs then a dedicated author language is much more suitable. One well-established author language is TUTOR, which has been used for many years to create material for the mainframe PLATO system. A version of TUTOR is also used on PLATO's stand-alone microcomputer systems.

Two popular author languages are PILOT and MICROTEXT, both of which are suitable for CALL. It must be borne in mind, however, that the uninitiated computer user will need a lot of time to achieve complete fluency in an author language and will require a good deal of determination in order to get the best out of it. Users must have a good sense of logic and understand concepts of variables, matching and conditional/unconditional branching. It is nevertheless true that the conventions of author languages are much easier to learn than those of BASIC.

PILOT is available for most popular microcomputers. It is probably the easiest author language to learn, its set of instructions comprising mainly single letters – the initial letters of the words they represent. The following is an example of a PILOT program which achieves the same result as the BASIC program above:


```
T:    Type the correct form of «spielen»
T:    in the Present Tense:
T:    Mein Bruder ------ Tennis.
A:
M.:   spielt
.TY:  Well done!
TN:   Sorry, wrong, the answer is «spielt».
E:
```

The logic of the program is reasonably transparent even to someone who has had no experience with an author language. The letters followed by a colon on the left are the instructions which cause the computer to behave in a particular way. They are interpreted as follows:

T:	display *Text* on this line on screen
A:	*Accept* input from keyboard
M:	*Match* item on this line against input
TY:	display *Text* on this line on screen if match succeeds *(Yes)*
TN:	display *Text* on this line on screen if match fails *(No)*

The program gives no indication, however, of the full power of PILOT's matching facilities. PILOT allows the user to anticipate specified wrong answers, accept answers in upper or lower case letters, accept alternative answers, ignore minor spelling errors, and so on. Branching and scoring are other features, and in some versions elaborate graphics can be created. It is an ideal author language for the beginner.

MICROTEXT was developed at the National Physical Laboratory, Teddington, Middlesex. The most recent and best version runs on the BBC microcomputer[1]. MICROTEXT is designed to simplify the production of person-computer dialogues. It is quite suitable for creating simple tutorial or interactive material for CALL.

MICROTEXT is based on a series of frames of information, each of which is numbered. This facilitates branching and remedial runs conditional upon the learner's performance. The following MICROTEXT program shows how the dialogue is set up:

```
*1
What is the capital of the
United States?
?
NEW YORK – 2, WASHINGTON – 3, – 4
```

1 Published 1985 by Acornsoft Ltd, Betjeman House, 104 Hills Road, Cambridge CB2 1LQ.

```
*2
No, although New York is the largest
city, it is not the capital.
— 1

*3
Yes, well done.
— 5

*4
No, the capital is Washington.
— 5

*5
That is the end of this lesson.
$END
```

The logic of the program is fairly transparent. The numbers preceded by asterisks represent the frame numbers, and the numbers preceded by arrows indicate the frames to which the program must branch. Where, for example, NEW YORK precedes the arrow, the program must branch to frame 2 if the learner gives the answer NEW YORK. Unconditional branching is indicated by an arrow followed by a frame number. A question mark at the beginning of a line causes the program to wait for an input from the keyboard.

Using MICROTEXT, the program creator has considerable control over the appearance of each frame. Colour changes, underlining, fixed messages and scrolling are possible. A wide range of matching facilities, like those for PILOT described above, are also provided. Graphs and charts can also be created. Scoring is easily handled.

MICROTEXT is more complex than PILOT, but it offers greater flexibility. Versions for creating interactive video material have also been produced[2]. The newcomer to computing should be able to write simple CALL routines without too much difficulty, but a degree of perseverance is needed to create really interesting dialogues.

The problem with CALL software is that one needs so much of it in order to make it effective. As already indicated, creating new software with a general-purpose programming language is very time-consuming, and although a dedicated author language is easier to learn it does not make

2 A cassette-based interactive video system using MICROTEXT has been developed by David Little, Trinity College, Dublin. A system using videodiscs has been developed by Paul Bangs, Information Technology Unit, Buckinghamshire College of Higher Education, High Wycombe.

the job of software creation that much quicker. So what is the solution? A view I have expressed on numerous occasions[3] is that the most efficient way in which an educational institution can build up a reasonable software library is by using a range of authoring packages (sometimes known as authoring systems). An authoring package designed specifically for creating CALL software enables the non-programmer to create usable material exceptionally quickly, reducing the time taken to create one hour of learning material to about three to five hours.

An authoring package opens the door to do-it-yourself software, shielding the user from the complexities of the logic of programming, and offering a simple framework into which the CALL material can be slotted. The disadvantage of authoring packages is that they can be restrictive and result in rather unimaginative courseware. The user is saddled with the framework set up by the creator of the authoring package, and although the content can be infinitely varied, the form of presentation tends to become monotonous. Nevertheless, authoring packages do enable pedagogically sound courseware to be produced at an impressive rate, and if a variety of packages is used the problem of monotony can be overcome.

TEACHER'S TOOLKIT was the first authoring package I wrote. It was simple in concept and enabled the teacher to set up a series of questions with a range of possible answers. It was later rechristened QUESTIONMASTER[4]. In the light of recent developments, it is not a package which I regard with much pride. However, it has served its purpose. Apart from forming the basis of APFELDEUTSCH[5], it has also been used (by a linguist with no knowledge of programming) to produce two revision packages for learners of French, LOGIFRENCH 1 and LOGIFRENCH 2[6].

There is now a good selection of authoring packages available for the BBC microcomputer, which is the most widely used computer in education in the United Kingdom. Using a range of authoring packages, it is quite feasible to build up a decent CALL library in a relatively short space of time. I say «relatively», because creating any kind of software is a time-consuming process, even with an authoring package; APFELDEUTSCH took six months of solid work to produce with QUESTIONMASTER.

One of the easiest authoring packages to use is GAPKIT[7]. A good number of reinforcement exercises used by language teachers rely on the tech-

3 See DAVIES (1982a, 1982b, 1985a, 1986).
4 Published 1982 by Hutchinson Software, 17–21 Conway St, London W1P 6JD. See HOLMES (1984), LAST (1984).
5 Published 1981 by Wida Software, 2 Nicholas Gardens, London W5 5HY.
6 Published in 1983 and 1984 by Wida Software (see Note 5).
7 Published 1983 by Camsoft, 10 Wheatfield Close, Maidenhead, Berks SL6 3PS.

nique of presenting the student with a series of sentences or a continuous piece of text in which specific categories of words or parts of words, have been deleted: for example, prepositions, articles, verb endings, suffixes and prefixes. This technique of selective deletion is easily implemented with GAPKIT. The teacher uses one program in the package to create a new exercise or revise an old exercise. Creating a new exercise is easy. All the teacher has to do is type the text, indicating where the gaps are to appear by placing the «slash» character (/) at the beginning and end of each gap, thus:

D/er/ Rhein ist e/in**/ europäisch/er/ Fluss. Er entspringt /in/ /der/ Schweiz und ...

Explanatory notes to precede the exercise are also created by the teacher. These can be summoned up by the student at any time during the exercise. Discrete clues can also be built into the data: for example, indicating the case or gender of a noun. The gapped text, together with the explanatory notes and clues is stored on disk and can immediately be used by the student by means of another program.

The above extract would be presented to the student like this:

D-- Rhein ist e---- europäisch-- Fluss. Er entspringt -- --- Schweiz und ...

The computer works out where the gaps are positioned on the screen, and locates the cursor in turn at the point where each gap begins. The student can then attempt to fill in the gap, ask for a clue, or make the computer fill in the gap itself. Errors are reviewed over and over again, until the student has produced a perfectly correct «page».

The current version of GAPKIT allows the teacher to «disguise» the length of the anticipated response by padding out the answer with any number of asterisks. The above extract contains an example of this: e/in**/. When the student completes this gap, the RETURN key has to be pressed after entering the «-in» of «ein», in order to indicate that no ending is required. This feature makes it possible to produce exercises which would be pointless if the student could work out the answers just by counting the dashes.

Allied closely to the selective gap-filling exercise is Cloze procedure. Having been neglected by foreign-language teachers in the United Kingdom for so many years, Cloze is now coming into fashion – although teachers of English as a Foreign Language realised its potential a long time ago. Two authoring packages, my own CLOZEWRITE[8] and Chris Jones's CLOZEMASTER[9], enable the teacher to create and store texts which are

8 Published 1985 by Camsoft (see Note 7).
9 Published 1982 by Wida Software (see Note 5).

then used by the student to generate computerised Cloze exercises. In Cloze exercises words are removed at regular intervals and replaced by numbered blanks.

In CLOZEWRITE, the student can choose any regular deletion interval from 2 to 9, and also specify the deletion starting point, so that a variety of different exercises can emerge from one text. The length of each deleted word is not revealed to the student, until at least one attempt at the word has been made. A partial matching routine reveals minor typing or spelling errors. At any stage, the student can ask for individual letters or whole words to make the task easier. CLOZEWRITE can be used interactively or to produce printed handouts for use in class.

CLOZEMASTER works in a similar way, but permits the creation of longer texts – up to 50 lines, compared with the 18 lines allowed by CLOZEWRITE – and has less elaborate error diagnostic routines. The longer texts are handled by means of a «scrolling screen» facility, which is built into the program.

It can be argued that CLOZEWRITE and CLOZEMASTER do not produce «true» Cloze exercises, because the computer accepts as answers only those words which appear in the original texts. Ideally, the student should be able to enter any semantically and syntactically correct words in the numbered blanks. The main problem is that the computer is unable to recognise what words make sense in a particular context. In a «true» Cloze exercise every possible alternative would have to be anticipated and built into the data. Field-testing indicated, however, that although teachers felt it would be wonderful if the computer could accept a range of alternative responses, scarcely anyone was prepared to invest the necessary time in setting up the data accordingly. A CLOZEWRITE/CLOZEMASTER text can be typed in about twenty minutes, but it would take hours to prepare a text that allowed for every possible alternative rendering of every word which might be deleted. Above all, it must be borne in mind that the ease with which the teacher can create material means that new and topical texts can constantly be produced, so that the data never goes out of date.

It is debatable to what extent students would benefit from more elaborate preparation. Most words which are removed in a Cloze exercise of this type are function words, and there is usually little argument over what fits. The few content words that disappear can be guessed by most students by the third attempt. Teachers often talk about the frustration students experience when they are told they are wrong after producing an acceptable answer. In the author's experience it is teachers who tend to get frustrated rather than students. Most students willingly make several successive attempts

at a word without exploding with indignation if acceptable answers are rejected. The mind-searching which goes on at each attempt is in itself a valuable activity. In any case, CLOZEWRITE and CLOZEMASTER do not display messages indicating that the student is «wrong»; the computer merely signals that the student's answer does not match the original word.

CLOZEWRITE is part of a planned package of exercises centred on the mutilation, transformation and reconstruction of texts. It was conceived as a companion to my earlier COPYWRITE package[10], which I describe below, and uses the same teacher's authoring/editing program. Texts created for COPYWRITE are thus completely compatible with CLOZEWRITE. The other similarly compatible programs in the package are provisionally known as ENIGMA, SCRAMBLER, PREDICTION and TEXTSALAD. ENIGMA is a sort of decoding game in which the learner tries to decipher a text in which all the letters are replaced by other letters. SCRAMBLER jumbles each word in the text and invites the user to unscramble them. PREDICTION aims to exercise the learner's ability to anticipate what is likely to come next in the text. Starting with a blank screen the learner is offered a selection of words, four of which are taken at random from the text and one of which is the original first word. The idea is to rebuild the original text word by word by making accurate predictions. TEXTSALAD shuffles the lines of the original text and asks the learner to re-arrange them.

COPYWRITE was derived from John Higgins's STORYBOARD[11]. Like CLOZEWRITE and CLOZEMASTER, it enables the teacher to create and store texts which are then mutilated by the computer and presented to the student for reconstruction. COPYWRITE takes the Cloze idea to its extreme. The text is displayed on the computer screen, and after a time reduced to punctuation marks and dashes indicating the length of the missing words. The student then attempts to reconstruct the text. This seemingly impossible task is easier than it looks, because any word anywhere in the text can be chosen. Each time a correct word is entered, every occurrence of it appears on the screen. Different strategies may be adopted. Students may begin with low-frequency content words they remember from the first reading, or high-frequency words such as articles, common verbs, prepositions, conjunctions and pronouns. A partially completed screen might look like this:

10 Published 1983 by ESM, Duke St, Wisbech, Cambs PE13 2AE. The new package is provisionally known as FUN WITH TEXTS and due to be published by ESM in 1986.

11 See HIGGINS & JOHNS (1984), p. 107.

---- a ------- ---- a ---------, he
------- ------ the ------ ---- he -----
to ----. He -----, ---- ------- ----
------ is -- is --- the -----. I
--------- -- ------- I --- -- a -----
----- the ------- --- -------- ---, I
--------- ----- the -----, Are you a
----?, Are you a --------? --- --
--. He ----- --- ----- --- Are you
----? The --- ------ ------- --- ----
--, I --- -----. -----! the -------
--------. You ---- --- ---, I --.

A scoring facility is provided, the maximum number of points being the number of words in the text multiplied by 10. If the student gets stuck, it is possible to call up the first letter of a word as a clue, to request the computer to fill in a whole word or to read the text again. This, however, causes points to be deducted: 5 for a letter, 10 for a word and 50 for reading the text again. The original STORYBOARD had no scoring facility, but field-testing by the publishers indicated that some kind of points system provided a powerful incentive.

COPYWRITE encourages intensive reading and gives the student valuable insight into language redundancy and the way words tend to combine and suggest what is coming next. If several students work together, it is interesting to note how much conversation the exercise generates about both the content and the language of the text.

Creating new COPYWRITE texts is simplicity itself. The teacher chooses a passage of suitable length, calls up the teacher's authoring/editing program and just types the text line by line. The completed text is stored on disk, and can immediately be made available to students. It is therefore possible to produce a constant supply of up-to-date material; a text from the morning newspaper can be ready by lunchtime.

Teachers have also discovered that students can benefit by using the authoring/editing program. The student uses the program to create a short essay, which the teacher corrects on the computer screen. The essay is thus marked as it is actually written, and can also be used by other students as a reading exercise.

SPEEDREAD[12] is a package designed for producing material to improve reading skills. The principle of authoring and editing the text for storage on disk is much the same as described for COPYWRITE. But in addition SPEEDREAD enables the teacher to insert sets of multiple-choice ques-

12 Published 1984 by Wida Software (see Note 5).

tions at any point in the text in order to test the student's comprehension. The text can be presented to the student at different specified speeds, thus providing a controlled and flexible environment for the development of rapid reading skills.

There are a number of packages for authoring multiple-choice exercises and tests. One easy-to-use package is Chris Jones's CHOICEMASTER[13]. The teacher simply types in the questions, answers and distractors, and leaves the computers to organise the storage of the data on disk and present it to the student. CHOICEMASTER offers two modes to the student: tutorial and test. In tutorial mode, the student is given immediate feedback as each question is attempted, and offered clues or explanations when wrong answers are selected. In test mode, the student attempts the questions but is offered no feedback until the whole test has been completed. Some teachers are prone to disparage multiple-choice exercises, but my own experience indicates that students perceive them as beneficial, particularly if they have to tackle examinations in which multiple-choice tests play a significant part. One advantage of making computerised multiple-choice tests available to students as examinations approach is the free time which the teacher thereby gains to concentrate on other activities which cannot be handled by the computer.

A completely different type of authoring package, entitled VOCAB and also written by Chris Jones[14], attempts to move away from the tutorial approach to CALL by offering the teacher the facility to create files of words in suitable contexts, which are used in a variety of linguistic games. The games include SKULLMAN, a variation of HANGMAN; ANAGRAMS; MINDWORD, a word game based on the Word Mastermind principle; ALPHAGAME, a game in which the student has to guess what word the computer is thinking of; WHICH WORD?, a multiple-choice exercise; WORD ORDER, a game in which the student has to sort a set of jumbled words into order to form a meaningful sentence. All the teacher has to do is to enter a series of words together with sentences containing them. The words and sentences are then used as bases for all the above games.

So authoring packages are by the far the easiest route to do-it-yourself software and it is likely that this is the area of CALL which will to expand most. It is easy to see why. A program that offers just a few words and phrases is of very limited value, however cleverly executed, and most off-the-peg CALL programs do not make it easy for the teacher to amend the data to suit his/her curriculum and teaching style. Authoring packages

13 Published 1983 by Wida Software (see Note 5).
14 Published 1984 by Wida Software (see Note 5).

may not produce imaginative software but they produce it quickly. Perhaps the ideal solution is a judicious mix of software produced by means of a programming or author language in combination with a large quantity of material produced with authoring packages. At present this seems to be the only way in which CALL software can be produced quickly enough to satisfy the demand – or before the hardware goes out of date.

National Centre for Computer-Assisted Language Learning
Ealing College of Higher Education, London

GRAHAM DAVIES
Project Leader

Bibliography

DAVIES, G. D. (1982a): *Authoring Techniques and Computer-Assisted Language Learning,* INTUS NEWS 6, 2, pp. 46–57.

DAVIES, G. D. (1982b): *Doing It Yourself,* EFL Gazette 37, pp. 6–7.

DAVIES, G. D. (1985a): *Computers in Modern Language Learning and Teaching.* In: WELLINGTON, J. J. (ed.) Children, Computers and the Curriculum, Harper and Row, pp. 134–151.

DAVIES, G. D. (1986): *Authoring CALL Courseware.* In: LEECH, G. & CANDLIN, C. (eds.) *Computers and the English Language,* Longman.

HIGGINS, J. & JOHNS, T. (1984): *Computers in Language Learning,* Collins.

HOLMES, G. (1984): *Creating CAI Courseware: Some Possibilities.* In: WYATT, D. H. (ed.) *Computer-Assisted Language Instruction, Pergamon,* pp. 21–32. (First published as a special issue of System 11, 1, 1983.)

LAST, R. W. (1984): *Language Teaching and the Micro,* Blackwell.

Books on BASIC:

DAVIES, G. D. (1985): *Talking BASIC: an introduction to programming for users of language,* Cassell. This is a comprehensive introduction to BASIC for the non-scientist. All the listed programs were written in a common version of Microsoft BASIC and tested on the Commodore 64 and BBC microcomputer.

HIGGINS, J. & JOHNS, T. (1984): *Computers in language learning,* Collins. A useful book on CALL in general, which includes numerous program listings for the Sinclair Spectrum. The Spectrum's dialect of BASIC is very different from most others, so the reader may need to work quite hard to convert the programs to run on other computers.

KENNING, M. J. & KENNING, M.-M. (1983): *An introduction to computer assisted language teaching,* Oxford University Press. This is geared mainly to CALL and guides the reader step by step through the process of creating CALL programs in BASIC. The version of BASIC presented to the reader is one found on mainframe computers, but the listed programs can be converted to run on microcomputers without too much difficulty.

LAST, R. W. & JOHNSTON, I. (1986): *BBC micro programming for the language teacher,* Blackwell. This book was still in preparation at the time of writing. It aims to present BBC microcomputer BASIC to the non-scientist, specifically the CALL user.