Sabri A. Mahmoud and Mohammed M. Mandurah, “CATIB: A Bilingual Authoring Language with Hypertext and Multimedia Capabilities.” Saudi Computer Journal, Volume 1, Number 1, Nov. 1994, pp. 36-51.

Keywords: CATIB, Arabic Programming Languages, Course Ware Development, Hypertext.

CATIB: A BILINGUAL AUTHORING LANGUAGE WITH HYPERTEXT AND MULTIMEDIA CAPABILITIES[1]

Sabri A. Mahmoud and Mohammad M. Mandurah
College of Computer and Information Sciences
King Saud University
P.O.Box 51178
Riyadh 11543, Saudi Arabia

المستخلص: "كاتب" لغة برمجة ثنائية اللغة (عربية/ إنجليزية) صممت خصيصا لتسهيل تطوير الدروس التعليمية بمساعدة الحاسوب. وتمتاز هذه اللغة ببساطة بنائها الذي يجعلها سهلة الإستخدام لغير المبرمجين، كالمدرسين مثلا، بغرض تعلمها واستخدامها بكفاية لكتابة وتنقيح الدروس التعليمية بمساعدة الحاسوب. وقد تم في عام ١٩٨٤ تطوير نسخة من "كاتب" تعمل تحت نظام تشغيل سي بي إم(CPM). ويناقش هذا البحث تطوير نسخة جديدة من لغة "كاتب" تعمل على نظام تشغيل دوس(DOS)، وهذه النسخة عبارة عن بيئة متكاملة تتضمن محررا شاشيا وبرامج فرعية للتعريب مع كثير من الملامح المحسنة. ويعالج هذا البحث بشيء من التفصيل معالجة النصوص الفوقية والوسائط المتعددة من داخل اللغة "كاتب" ومثل هذه الخصائص تسمح بتطوير فعال للدروس التعليمية بمساعدة الحاسوب.

Abstract: CATIB is a bilingual (Arabic/English) programming language that was especially designed to facilitate the development of computer-assisted instruction (CAI) courseware. The language has a simple syntax that makes it easy for non -programmers, such as teachers, to learn the language and effectively use it in writing and editing CAI courseware. An older version of CATIB was developed in 1984 to run under the then-popular CP/M operating system. This paper discusses the development of a new version of CATIB that runs under MS-DOS. This version is actually an integrated environment with a built-in screen editor, built-in Arabization routines, and with many enhanced features. The paper examines in some detail the implementation of hypertext and multimedia into CATIB for such features allow the development of effective CAI courseware.

1. INTRODUCTION

The efforts to produce Arabic programming languages or to Arabize existing ones started early in the 1970's. Early attempts concentrated on the development of Arabic interfaces to languages in order that they could handle I/O operations in Arabic [1]. Later attempts aimed at producing programming languages with Arabic vocabulary. Examples of these attempts can be found in [2], [3], [4], [5], [6], [7], [8], [9] [10],) The objective of these efforts was to produce general-purpose programming languages, similar to BASIC or PASCAL, where the programmer can use Arabic instructions and commands to code his programs.

Most of the Arabic programming languages that were developed in the past, however, were not well received among programmers for several reasons. Firstly, programmers are professional

[1] This work is supported by KACST, Project #AR-11-42.

people who are usually very fluent in English. Thus, natural language is not a barrier that would hinder their usage of an un-Arabized programming language. Secondly, programming languages are witnessing remarkable improvements in their power and capabilities for they receiving continuing support from the companies that produce them. Unfortunately, the Arabic counterparts could not keep up with these advancements. The reason being, most of these Arabic programming languages were developed as isolated research work that did not materialize into commercial products receiving proper support and further development [11].

Nonetheless, one sector that aspires to the existence of an Arabic programming language is the educational sector. What is needed here is a programming language that facilitates the development of educational courseware for computer-assisted instruction (CAI) applications. It is essential in such programming languages to have an Arabic lexicon because not many of the intended users (teachers and students) are fluent in English. In addition, since the intended users of such a programming language are non-professional programmers, these languages must be user-friendly, easy to learn and use, and must have features that support the development of effective educational courseware.

CATIB is one Arabic programming language specially developed for use in educational applications. It is a bilingual authoring language with many features facilitating the development of CAI courseware [12]. Without compromising the pedagogical objectives of the language, CATIB has a simple syntax making it easy for non-programmers to learn the language, and use it in developing useful applications. The work on the development of CATIB started in 1984. The original version of the language was based on the then-popular CP/M operating system. In 1991, the project received a grant from King Abdul-Aziz City for Science and Technology (KACST) in Saudi Arabia to port the language to MS-DOS based machines and enhance it with advanced features that reflect the advances in computer technology. KACST in Saudi Arabia is equivalent to the National Science Foundation (NSF) in the U.S.A.

This paper discusses the development of the version of CATIB running under MS-DOS operating system. This version is actually an integrated environment with a built-in screen editor and with many enhanced features. In particular, the paper examines, in some detail, the implementation of hypertext into CATIB.

2. CATIB PROGRAMMING ENVIRONMENT

The original programming language, CATIB, was conceptually designed after the well-known language PILOT, "Programmed Inquiry Learning and Teaching" [13]. PILOT is a specialized language oriented towards the development of educational dialogues, drills, tests, etc., rather than towards computations which are well handled by general purpose languages. Similar to PILOT, CATIB is an interpreted language with a simple syntax. A program written in CATIB is composed of consecutive statements' lines; each line may contain a maximum of one statement. A statement line has the following format:

[<statement>] : <text> | <expression>

The <statement> in CATIB syntax is optional; if it does not appear in the line, the interpreter will repeat the statement in the previous line.

A sample program written in CATIB is given in the Appendix. As can be seen, each line in the program follows the format described above. The example is somewhat lengthy as it was intended

to demonstrate the hypermedia capability of the language. This feature will be examined in more detail later in the paper.

Table 1: Instructions of CATIB for MS-DOS Based Machines

I- Input/Output Instructions:

ACcept	accept an answer.
AcceptHang	accept an answer on the same line.
AcceptSingle	accept a single character.
AcceptLine	accept a line of characters.
CeNter	center the text on the display line.
CenterHang	center the text on the display and keep cursor on same line.
CenterPrinter	center the text on the printer.
FooT	indicate end of a screen display. Halt and prompt.
Hyper	display help message on how to process hypertexts
InputMax	input maximum number of characters.
ReMark	indicate that the whole line is a remark.
RemarkWrite	write a remark.
SaY	a special instruction to output a recorded sound message.
TypE	type a message on the display.
TypeHang	type a message and keep cursor on the same line.
TypePrint	type a message on display and print it.
TypePrintHang	type and print a message, and keep cursor on same line.

II- Control Flow Instructions:

Do .. WHile	loop between Do and While keywords.
EnD	end of subroutine of program.
For	loop a predetermined number of iterations.
JumP	jump to a label.
PAuse	pause waiting for a press on the keyboard.
PRoblem	start a problem.
SuBroutine	call a subroutine.
SubroutineR	make a rondom call to a subroutine from a list.
SWitch	a multiway decision.

III- Screen Handling Instructions:

CancelJustify	cancel a previous justification command.
ClearS	clear screen.
ClearL	clear from cursor position to end of line.
ClearE	clear from cursor position to end of screen.
ClearHome	clear screen and home the cursor.
CloseWindow	close a specified window.
CursorMove	move cursor to indicated coordinates.
JustiFy	set the margins for justification.
SetColor	set the color of the text that follows.
UseWindow	make a specified window the active window.
WiNdow	open a window for display.

Table 1 (continued)

IV- String Manipulation Instructions:

CoMpare	compare two expressions or variables.
LeNgth	calculate the length of a string.
MAtch	match with any of the listed patterns.
MatchJump	match and then jump if no match is found.
StrCat	append one string to another.
StrcOpy	copy one string into another.
StrcMp	compare one string to another.
VAl	convert a string to integer.

V- Mathematical Functions:

Abs	compute the absolute value of the argument.
ComPute	compute an expression (string or integer expressions).
Cosine	compute the cosine of the given angle.
Exp	compute the exponential e to the specified power.
Log	calculate the logarithm of the argument with base 10.
NewVariable	clear the content of a variable.
Sine	compute the sine of the given angle.
Sqr	calculate the square of the argument.
SqrT	calculate the square root of the argument.
Tan	compute the tangent of the given angle.

VI- Graphics Instructions:

ARc	draw a circular arc.
BAr	draw a two-dimensional bar.
CiRcle	draw a circle with the given center coordi
ELlipse	draw an ellipse.
GetImage	save a bit image of the specified region of
GetPixel	get the color of the specified point on the
LiNe	draw a line between to the two given poin
PolY	draw the outline of a polygon.
PutImage	output a previously saved bit image onto t
PutPixel	output a pixel at a specified point on the s
ReCtangle	draw a rectangle.

VII- File Control Instructions:

CLose	close the indicated file.
OPen	open the indicated file.

VIII- Environment Control Instructions:

FreeMemory	free a previously allocated memory.
GetDate	display the current date.
GetMemory	allocate an area of memory.
GetTime	display the current time.

The first line in the program starts with the statement 'remark' to indicate that the text in this line is a remark and should receive no further processing. The second line has the statement 'clears', it tells the interpreter to clear the screen. Then, we see several lines that contain the statement 'type'. Each one of these lines tells the interpreter to send the text on the line to be displayed on the screen. More explanations about writing and running CATIB programs will be given in later sections of the paper.

The vocabulary of the original version of CATIB that was developed for CP/M-based machines contained about 33 statements. The MS-DOS version of CATIB is a much improved version with many enhancements. The simple syntax has been preserved, but the list of statements and instruction in CATIB vocabulary now contains many powerful instructions that allow the development of a variety of applications. Table 1 lists the instructions supported by the language [14]. CATIB now has 72 instructions distributed into eight groups: I-Input/Output Instructions, II-Control Flow Instructions, III-Screen Handling Instructions, IV-String Manipulation Instructions, V-Mathematical Instructions and Functions, VI-Graphics Instructions, VII-File Control Instructions, and VIII-Environment Control Instructions. Table 2 lists the number of instructions in each group.

Table 2: The number of CATIB instructions in each group

	Instruction Group	Number of instructions
1	Input/Output Instructions	17
2	II-Control Flow Instructions	9
3	Screen Handling Instructions	11
4	String Manipulation Instructions	8
5	Mathematical Instructions and Functions	10
6	Graphics Instructions	11
7	File Control Instructions	2
8	Environment Control Instructions	4

The current version of CATIB is a complete bilingual programming environment with a built-in screen editor. When CATIB is loaded and runs, it displays the opening screen simulated in Figure 1. Here the user can choose the language of the environment, the language of the text, to go to the editor, to go to CATIB, or to quit the application. The user can move between these selections in the main menu using the cursor keys. As he moves between these selections, a help message is displayed at the bottom of the screen. Figure 1 shows the English environment of CATIB in which all menus, messages, commands and instructions are in English, and writing is from left to right.

By continuously pressing the 'Enter' key while the selection from the main menu is on 'Environment', the user can toggle between the Arabic and English environments. Figure 2 shows the opening screen of CATIB in the Arabic mode in which all menus, messages, commands and instructions are in Arabic, and writing is from right to left. CATIB has its own transparent Arabization shell that is loaded on top of DOS to enable input and output of bilingual texts. Pull-

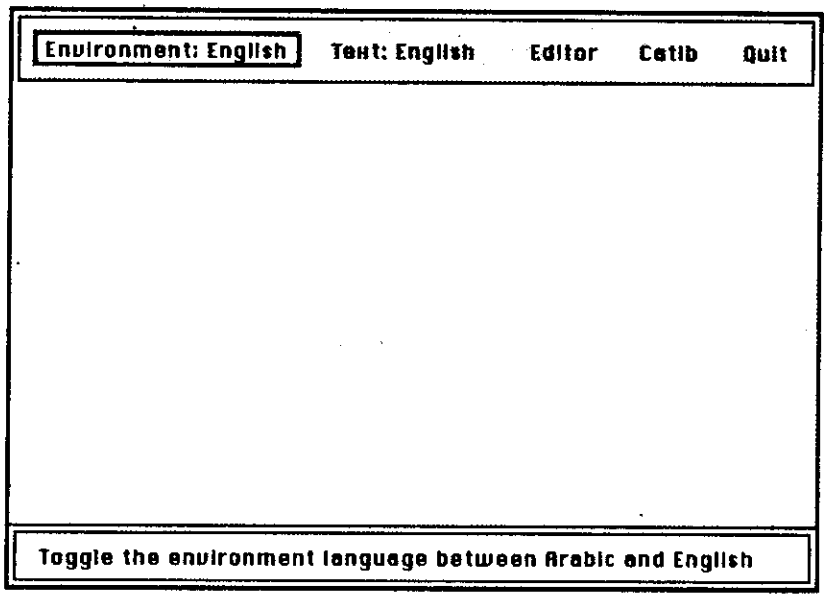

Figure 1: Opening screen of CATIB (English mode)

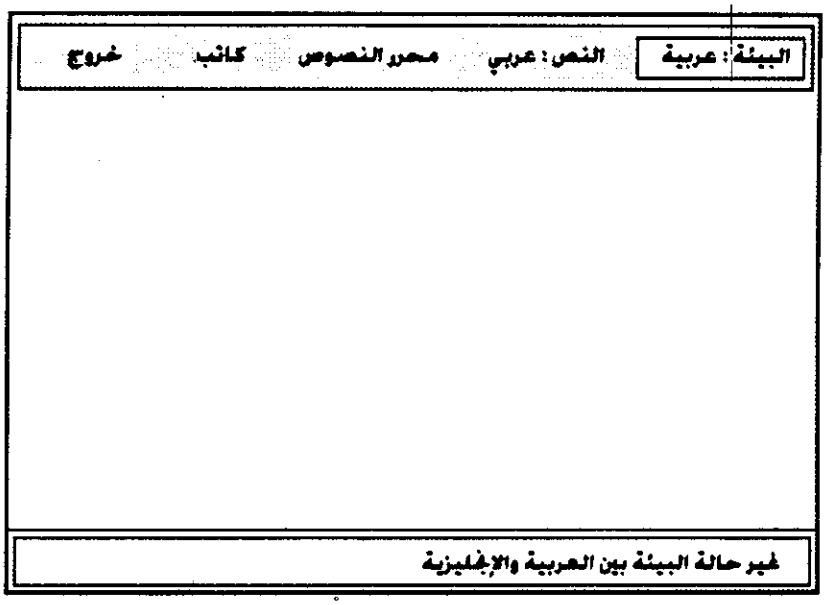

Figure 2: Opening screen of CATIB (Arabic mode)

Figure 3 shows the opening screen of the 'Screen Editor' that is part of the authoring system. The figure displays an example of the menu system in CATIB. This particular example shows the options available under the selection 'System'. As can be seen, the user can toggle between Arabic and English from any place within the application. It is beyond the scope of this paper to describe the detailed operation of CATIB. These details can be found in CATIB User Manual [15]. We would like in the remaining part of this paper to examine some of the novel features in CATIB.

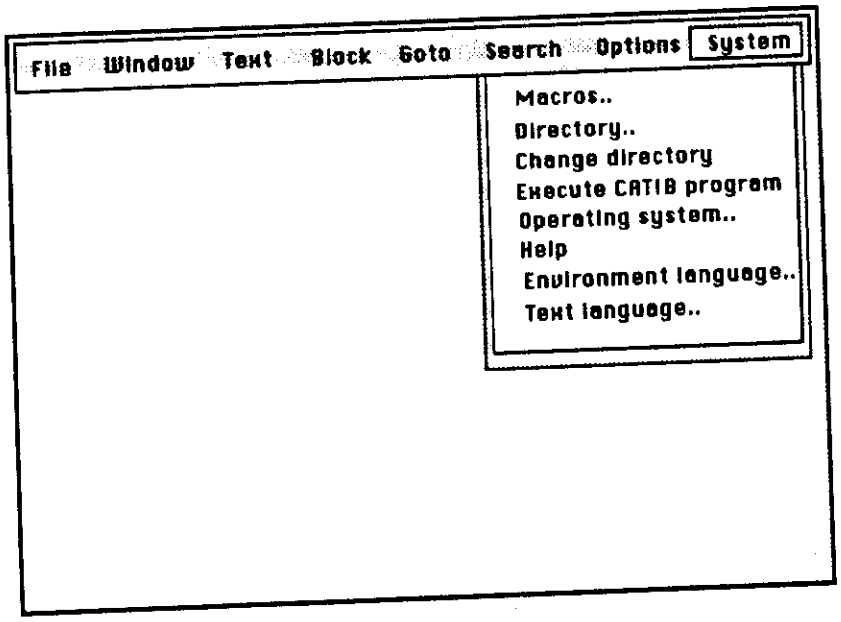

Figure 3: Built-in screen editor of CATIB

Two particular features in CATIB that deserve special discussion is the support of hypermedia. These are major enhancements in CATIB as they facilitate the development of effective educational courseware. The implementation of these features in CATIB is discussed in some detail later in this paper. For the sake of readers who do not speak Arabic, all the examples discussed here were developed under the English evironment of CATIB.

3. HYPERTEXT AND CAI

Hypertext is an approach to information management in which data is stored in a network of nodes connected by links. Nodes can contain text, graphics, audio, video, as well as source code or other forms of data. The nodes, and in some systems the network itself, are meant to be viewed through an interactive browser and manipulated through a structure editor [16]. During the past few years, interest in hypertext has accelerated sharply. To understand why hypertext is attracting such attention, one must understand how a hypertext "document" differs from a conventional paper document.

In most conventional paper documents - such as journal articles, technical manuals, or novels - physical structure and logical structure are closely related. Physically, the document is a long linear sequence of words that has been divided into lines and pages for convenience. Logically, the document is also linear: words are combined to form sentences, sentences to form paragraphs, paragraphs to form sections, etc.

Many other paper documents - such as encyclopedias, dictionaries, and other reference works - separate logical structure from physical structure. Physically, these documents are linear sequences of independent units, such as articles on specific topics or entries for individual words. Logically, they are more complex. The reader seldom reads such documents from beginning to end, but rather searches them to locate the article or entry of interest (a form of random access), then reads that portion sequentially. However, the reader is likely to encounter various cross references to other entries. To follow those pointers, the reader must locate the appropriate volume, find the appropriate entry, and then the relevant portion. The logical structure of reference and other similar documents is, thus, more complex. They have a sequential structure that aids search, but the logical path for the reader is a network that can crisscross the entire document of set of documents, from one item to another, to another, etc. Such documents are more flexible but they are also cumbersome, particularly when they appear in large, multi-volume formats.

Hypertext documents, in which information is stored in nodes connected by links, provide most of the flexibility of reference works as well as add a number of new features. Each node can be thought of as analogous to a short section of an encyclopedic article or perhaps a graphics image with a brief explanation. The links join these sections to one another to form the article as a whole and the articles to form the encyclopedia. These links are usually shown for each node as a "from" link pointing to the node just read and a set of "to" links that indicate the multiple nodes which one may select to read next [17], [18]).

Furthermore, while conventional publications are limited to text and graphics, hypertext nodes offer sound, video sequences, animation, even computer programs that begin running when the nodes in which they are stored are selected. Also, while the organizational and cross-reference structures of conventional documents are fixed at the time of printing, hypertext links and nodes can be changed dynamically. Information in individual nodes can be updated, new nodes can be linked into the overall hypertext structure, and new links added to show new relationships.

4. IMPLEMENTATION OF HYPERTEXT IN CATIB:

In the design of CATIB, we tried to borrow many of the better features found in different programming languages. Following the example set by assembly language, in CATIB any line of the code can have a label through which it can be called as a subroutine, or to which control flow can be transferred . A label-name in CATIB is identified by a star at the beginning of the line immediately preceding the label itself. In the sample program in the Appendix we can see many labels: *Edit, *Watch, *Format, *Evaluate, *Zoom, and *Output at lines 20,53,82, 101, 120, and 135 respectively. Line numbers are added to facilitate referring to specific commands. We found that this technique is a very flexible way to store and access knowledge in the program. For example, we can have the following label: *Test3_Lesson5, to indicate that this label is the entry of Test3 on Lesson5, after which is added the code of the test. This test can then be addressed from anywhere in the program using its label.

Utilising this labelling feature in CATIB, enabled simple and flexible implementation of hypertext. A hypertext keyword is treated as a label of a subroutine that should be called and executed. When the user selects a hypertext word or phrase, CATIB searches for the line number that has a label matching the hypertexted phrase. CATIB then commences execution of all the instruction lines following that label. Before jumping to the label, CATIB stores the return address. Once an 'end:' statement is encountered, CATIB returns to the original text.

Figure 4 shows the structure of hypertext as is implemented into CATIB. The figure shows that while the main lesson is being executed, the user selected more details about "phrase_b". CATIB searches in the labels table for a label matching "phrase_b", and then transfers control to it after saving the return address. The system presents the requested details of "phrase_b". Once an 'end:' statement is encountered in the subroutine "phrase_b", the user is returned to continue the execution of the main program.

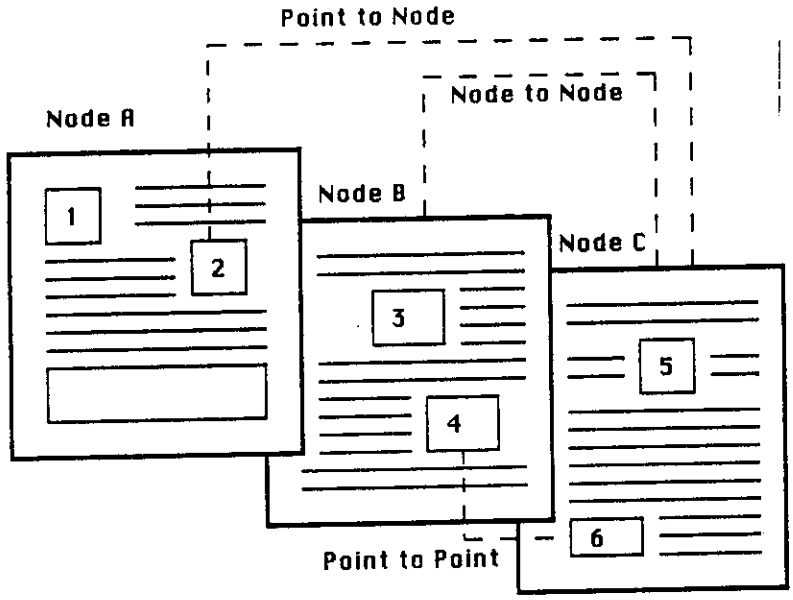

Figure 4: Hypertext links

The process of inserting hypertexted words or phrases in any CATIB program is as follows:

a. The programmer specifies a hypertexted word/phrase while programming by enclosing the hypertexted phrase between stars and square brackets as follows: [*hypertext phrase*].

b. The programmer then writes the details of this hypertexted word/phrase as a CATIB subroutine with a label matching exactly the hypertexted phrase. In this part of code the user may insert other hypertexted phrases.

c. When CATIB code is executed by the user, CATIB displays hypertext phrases in a different color. The user at this stage has several options :

i. Continue with the lesson by pressing the "Space-bar" key.

ii. Display the previous page by pressing "Pg Up" key.

iii. Activate the selected hypertext phrase by pressing "Enter" CATIB then calls the corresponding subroutine and starts executing it.

iv. Select another hypertext phrase. This is done by pressing the "Tab" key for forward movement to other phrases or "Shift-Tab" keys for backward movements. After selecting the desired hypertext phrase, the user can activate the phrase by pressing the "Enter" key.

This process is repeated every time information is displayed on the screen. Note that the hypertext node, here, need not contain only text, but can be a full CATIB code that plays sounds or music, or display images and graphics. This is done by including CATIB instructions for playing sound, displaying images, and generating graphics.

This implementation of hyptertext in CATIB has the following advantages :

1. It is userfriendly, as constructing a hypertext system is as easy as writing a CATIB program.
2. It does not require additional complex constructs to CATIB which is intended as a simple and easy to learn language.
3. It enables the programmer to use all CATIB instructions in the hypertext nodes, and therefore not restricted to text. Interactive instructions may be included in delivering more information about any phrase. Text, graphics, and speech messages may be included in any hypertext node.
4. Older CATIB courseware may be easily modified to include hypertexts.

5. EXAMPLE OF A CATIB PROGRAM WITH HYPERTEXT

The following example demonstrates the use of CATIB in writing an instructional program explaining the operation of a screen editor. The example also shows implementation of hypertext is implemented in CATIB. This example is a section of the CAI-editor help document. The listing of this program is given in the Appendix.

At the beginning of the lesson, the screen shown in Fig. 5 is displayed. This screen simulates the outcome of executing the first 17 lines in the program. The Edit, Watch, Output, and Zoom words are all enclosed between stars and square brackets, and will be displayed in a different color as they are hypertexted words. In Fig. 5, the hypertexted phrases are underlined by a thin line or by a thick line to simulate their different colors of display. A thin line under a word/phrase indicates a hypertext word/phrase. A thick line under a hypertext word/phrase indicates that the word/phrase is selected. From the figure, it can be seen that the "Edit" word is initially selected, line 7 of the program.

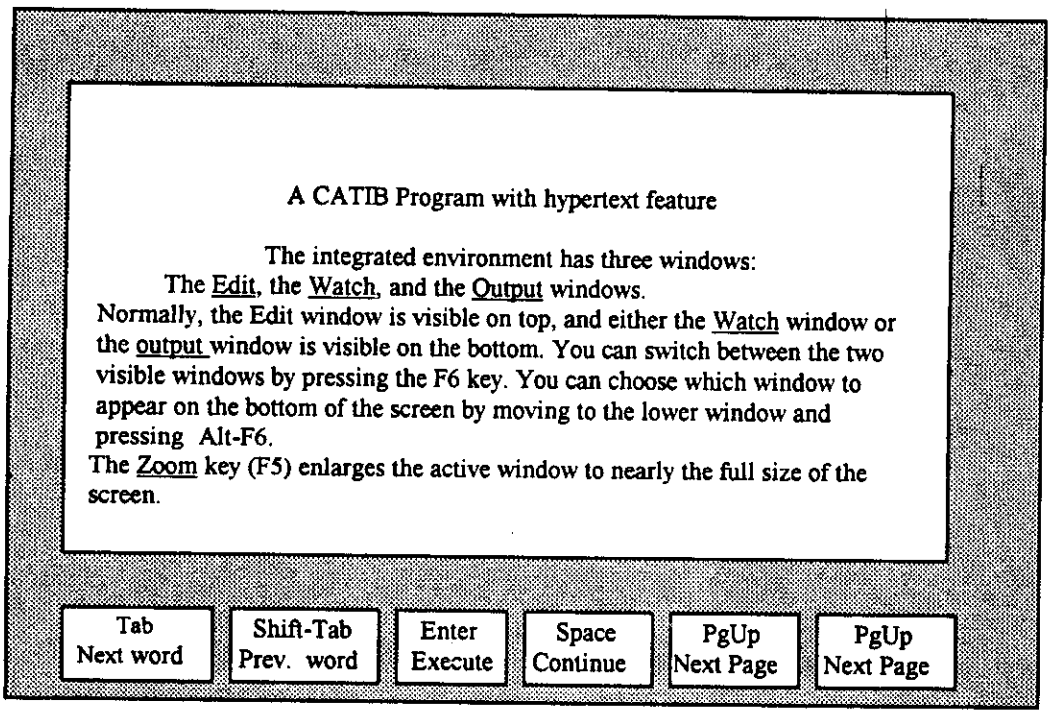

Figure 5: A lesson with hypertext phrases

Line 17 in the program has the special statement 'hyper:'. This statement tells CATIB to display a help-message at the bottom of the screen to guide the user on how to select a hypertext word/phrase. As shown in Fig. 5, the user has several options: select next word, select previous word, execute a hypertext word/phrase, continue, page up, or return.

If, for instance, the user wants more information about the hypertext word "Edit", he presses the "Enter" key while the word "Edit" is being selected. This opens the hypertext screen shown in Fig. 6. From the listing in the Appendix, it can be seen that the hypertext node of "Edit" is actually a subroutine that contains several lines of CATIB code, and that begins with the label "Edit", lines 20-52. During the compilation of the program the compiler searches for all labels in the program and builds a table of them. These labels can be addressed using statements such as, 'Jump' and 'Subroutine', or they can be accessed as hypertext words.

If the hypertext node has more than one page, the screen will indicate the page number as shown in the Fig. 6. After reading the above screen, if the user presses 'Space-bar' key for more information on "Edit", the next page will be displayed. By repeatedly pressing 'PgUp' or 'Space-bar' keys, the user can read the different pages in the node. The user can return to the first screen by pressing the 'ESC' key. From there, he can choose other hypertexted keywords if he desires. Pressing 'Space-bar' or 'ESC' in the first screen executes the rest of the code and terminates the main CATIB program.

As has been mentioned before, the <statement> in a CATIB line is optional; thus, repeating the instruction 'type' in the sample program is optional. In CATIB, a line starting with ':' indicates that the instruction of the current line is the same as the previous line. This is clearly shown in the example in the code which describes the label "Zoom" and "Output", lines 4-17, 22-35, .. etc.

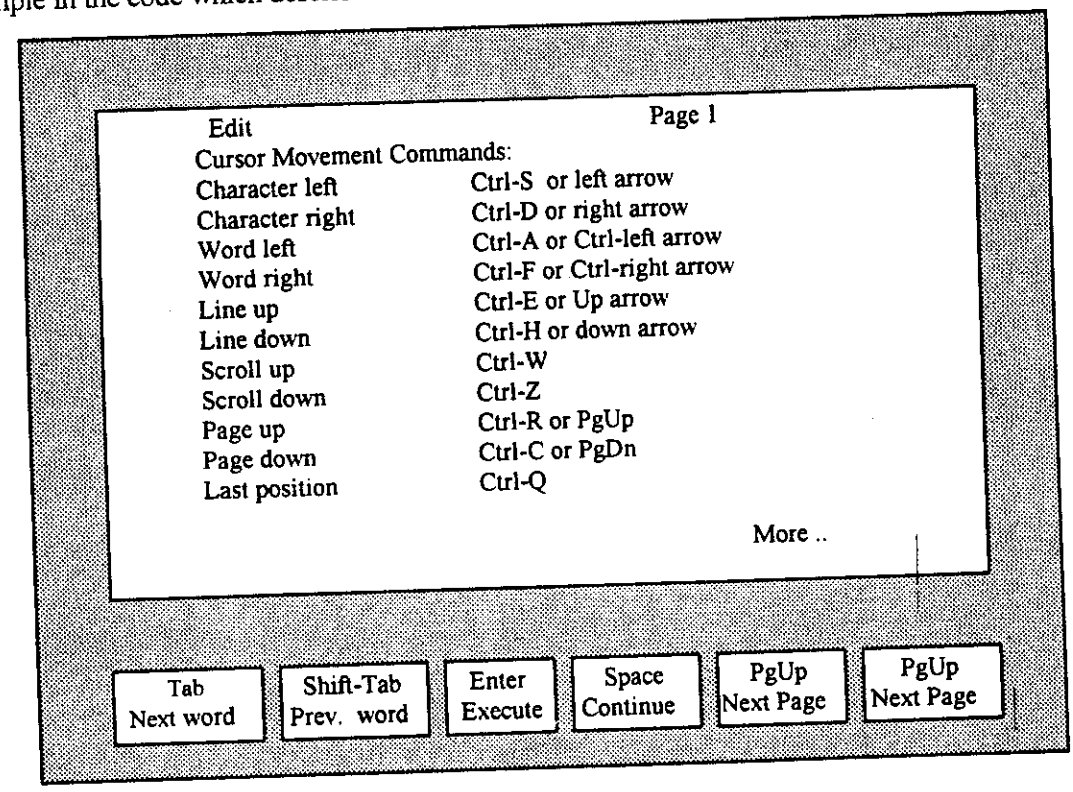

Figure 6: "Hyper" help messages

7. IMPLEMENTATION OF MULTIMEDIA

CATIB has many instructions that facilitate the incorporation of multimedia in lessons. In Group VI of CATIB instructions in Table 1, we find many instructions that can be used to create and manipulate shapes and images. Of particular importance is the instruction 'PutImage'. When using this instruction it should be followed by a file name that contains the image to be displayed. The file should contain a bit image in the 'PCX' format. Hence, images, shapes, and graphics can be easily intermixed with text in the lessons written in CATIB.

Also, CATIB supports sounds and voices. The special instruction 'Say' in Group I in Table 1 reads the data of a digitized sound from a file and sends it to the speaker of the computer. In this version of CATIB, the only sound system that is supported is the speaker in the PC. However, other sound systems are planned to be supported in future versions of CATIB; particularly, the popular Sound Blaster card.

The sample program in the Appendix demonstrates how to use the instruction 'Say'. Line 18 of the program contains the instruction 'Say: WINDEND.SPH'. This instruction tells CATIB to open the file: WINDEND.SPH, reads its contents of digitized speech messages, and sends it to the speaker. The stored data in this file is a voice message to inform the user that there are no more pages of text following this line and that he has reached the end of the program or the subroutine. As can be seen in other locations in the program, the same instruction has been inserted near the end of all subroutines.

8. CONCLUSIONS

CATIB is a bilingual programming language that was developed to facilitate the production of educational courseware. The language runs under MS-DOS, and it has a simple syntax that makes it easy for non-programmers to learn the language and use it effectively to develop a variety of applications. At the same time, the language has many advanced features that allow the development of sophisticated applications. One major feature in CATIB is the support of hypertext and multimedia. These capabilities are becoming essential in modern programming languages, and can aid the development of effective educational courseware.

The work in this project has been supported by a grant from KACST in Saudi Arabia. Therefore, it is hoped that CATIB will receive continuing support for further developments. Future plans for further developments include porting the language to other operating systems, particularly, Windows and Unix.

ACKNOWLEDGMENTS

The authors would like to express their gratitude to the referees for their constructive criticism. The incorporation of their suggestions and comments improved the clarity of the final manuscript.

REFERENCES

[1] Ghandour, Z. "An Arabic Interface to APL," Proc. of the Conf. on Inf ormatics, Paris, 1976, 483-92.

[2] Dehlawi, F.M.A., and Mandurah, M. "LPA: Another Arabic Programming Language," Proc. of the 9th Saudi NCC, 1985,. 20-60.

[3] Hawaf, A.Y. "Design of an Arabic Programming Language ARABW with an Efficient Compiler," Proc. of the 9th Saudi NCC, Vol. 2, 1986.

[4] Kasim, Z.S. "The Development of the Arabic Programming Language LAITH," J. of Electronic Computers, Baghdad, #3, 1978, pp. 28-45.

[5] Khayat, M.G. "The Arabic Programming Language DHAD," Proc. of the Workshop on Comput er Processing of the Arabic Language, Kuwait, 1985.

[6] Khidr, M.Z., and Abdul-Majeed S. "GHAREEB: An Arabic Programming Language," J. of Electronic Computers, Baghdad, # 4, 1978, 480-97.
[7] Mahjoub, A., and Mathkour, H. "An Interactive Compiler for Arabic PASCAL," Proc. of the Regional Conf. on Informatics and Arabization [IRSIT), Tunis, 1988, pp. 167-82.
[8] Nasser, H. "Design and Development of PASCAL-like Arabic Programming Language," Proc. of the 2nd Int. Baghdad Conf. on Computer Technology and Applic ations, Baghdad, March 1986, C. 3-2.
[9] Sadoun, H. "Arabic BASIC: A New Arabic MS-DOS Based Programming Language," Proc. of the 12th Saudi NCC, Riyadh, Oct. 1990,. 449-61.
[10] Sayed, H. "Verbum (al-Qawl): An Arabic Programming Language," Proc. of the 1st Seminar on Bilingual Computing, Cambridge Univ., U.K., Sept. 1989,. 1-13.
[11] Mandurah, M.M. "Software Industry in the Arab Countries: Status, Current Issues, and Future Directions," J. of Information Technology for Development, Vol. 5, No. 2, June 1990,. 157-75.
[12] Mandurah, M.M., and Azhari, I. "CATIB: An Arabic Authoring Language for CAI," Proc. of the 7th Saudi NCC, Riyadh, Jan. 1984, 271-8.
[13] Starkweathers, J. *Nevada PILOT Reference Manual* , ELLIS Computing, 1982.
[14] Mandurah, M.M., and Mahmoud, S.A. *Development of An Advanced Arabic Authoring Language for CAI Applications,* KACST project #AR- 11-42, Progress Report # 2, KACST, Riyadh, April 1992.
[15] Mandurah, M.M., and Mahmoud, S.A. *CATIB User Manual,* to be published.
[16] Smith, J.B., and We iss, S.F.(1988), "Hypertext", Comm. of the ACM, Vol. 31, No. 7, July 1988,. 816-9.
[17] Rada, R. *Hypertext: from Text to Expertext* , McGraw-Hill Co., U.K., 1991.
[18] Shneiderman, B., and Kearsley, G.P. *Hypertext Hands-On!,* Addison Wesley Publishing Co., 1988.
[19] Hashim, S.H. *Exploring Hypertext Programming* , Windcrest Books, 1990.

Appendix
Source listing of a Sample Program in CATIB

```
remark : This is an example CATIB program to implement a hypertext example
clears :
type :
 : A CATIB program with hypertext feature
 :
 : The integrated environment has three windows:
 : The [*Edit*], the [*Watch*], and the [*Output*] window.
 : Normally, the Edit window is visible on top,
 : and either the [*Watch*] window or the [*Output*]
 : window is visible on the bottom. You can
 : switch between the two visible windows by
 : pressing the F 6 key.You can choose which window
 : to appear on the bottom of the screen by
 : moving to the lower window and pressing Alt -F6.
 : The [*Zoom*] key (F 5) enlarges the active window
 : to nearly the full size of the screen.
hyper :
say: windend.sph
end :

*Edit
type :          Edit          page 1
 :
 : Cursor Movement Commands
 : Character left   Ctrl-S or Left arrow
 : Character right  Ctrl-D or Right arrow
 : Word left              Ctrl-A or Ctrl-Left arrow
 : Word right             Ctrl-F or Ctrl-Right arrow
 : Line up                Ctrl-E or Up arrow
 : Line down              Ctrl-X or Down arrow
 : Scroll up              Ctrl-W
 : Scroll down            Ctrl-Z
 : Page up                Ctrl-R or PgUp
 : Page down              Ctrl-C or PgDn
 : Last position          Ctrl-Q
 :                        More ..
hyper :
type :          Edit     page  2
 :
 : Insert & Delete Commands
 :
 : Insert mode on/off    Ctrl -V or Ins
 : Insert line          Ctrl -N
 : Insert options       Ctrl -O O
 : Delete line          Ctrl -Y
 : Delete to end of line Ctrl -Q Y
 : Delete character left  Ctrl -H or Backspace
 : Delete character      Ctrl -G or Del
 : Delete word right     Ctrl -T
 : Delete block         Ctrl -K Y
hyper :
say : windend.sph
end :
```

```
*Watch
type :                    Watch    page  1
 :
 : The Watch window contains your watch expressions
 : whose values are updated each time you execute
 : a step of the program. You can use watches to
 : follow what happens to your data structures as
 : the program executes.
 : Watches can include [*format*] specifiers that
 : specify how they are to be displayed. This
 : allows you to see your data in a convenient
 : format.
 :                    More ..
hyper :
type :          Watch      page  2
 :
 : While the Watch window is a convenient way to
 : check the values of variables and
 : expressions as you trace through your
 : program, there are times when you only need
 : to check the value once and not clutter the
 : screen with another watch variable.
 : That's when you might find it more convenient to
 : use the Debug [*Evaluate*] command, which lets
 : you evaluate and modify the contents of
 : variables and expressions.
hyper :
say : windend.sph
end :

*format
type :        format
 : A format specifier consists of an optional
 : repeat count (an integer), followed by zero
 : or more format characters.
 : C      Character. Shows special display
 :  characters for control characters
 :  (ASCII 0..31); by default, such
 :  characters are shown as ASCII values
 :  using the xx syntax. Affects
 :  characters and strings.
hyper:
type : D      Decimal. All integer values are
 :  displayed in decimal. Affects simple
 :  integer expressions as well as
 :  structures containing integers.
hyper :
say : windend.sph
end :

*Evaluate
type :        Evaluate
 :
 :        Debug/Evaluate  (Ctrl  -F4)
 :
 : Brings up an Evaluate window with three boxes.
 :        [*Evaluate box*]
 :        [*Result box*]
```

```
                                                                    109
:       [*New value box*]                                           110
: The [*Output*] window contains the output                         111
: generated by your program. At startup, it                         112
: will display the last screen from DOS. You                        113
: can pan through this window using the cursor                      114
: keys, as well as the Home, End, PgUp and                          115
: PgDn keys. Use [*Zoom*] (F 5) to enlarge this                     116
: window to almost the full screen.                                 117
hyper :                                                             118
say: windend.sph                                                    119
end :
                                                                    120
*Zoom                                                               121
type :          Zoom                                                122
:       Options/Environment/Zoom windows                            123
: When the Zoom windows toggle is On, the                           124
: [*edit*], [*watch*], or [*output*] window is expanded to          125
: full screen. You can still switch between                         126
: the windows using F 6, but only one window at                     127
: a time will be visible. When this item is                         128
: toggled Off, you're returned to the                               129
: split-screen environment containing the [*edit*]                  130
: window and either the [*watch* ] or [*output*]                    131
: window. F 5 is the hot key for this toggle.                       132
hyper:                                                              133
say : windend.sph                                                   134
end :
                                                                    135
*Output                                                             136
type :          Output                                              137
: The Output window contains the output                             138
: generated by your program. At startup, it                         139
: will display the last screen from DOS. You                        140
: can pan through this window using the cursor                      141
: keys, as well as the Home, End, PgUp and                          142
: PgDn keys. Use [*zoom*] (F 5) to enlarge this                     143
: window to almost the full screen.                                 144
: Output 2/2                                                        145
: The Output window has borders and is always                       146
: shown in character mode. If your last screen                      147
: was a graphics screen, or if you want to see                      148
: ALL of the program's output screen with no                        149
: borders, use the User screen command                              150
: (Alt-F5) to view the last execution screen.                       151
: This only works for text and CGA graphics.                        152
hyper :                                                             153
say : windend.sph                                                   154
end :
```