Computer Science and Systems Analysis

Computer Science and Systems Analysis

Technical Reports

Miami University *Year* 1995

Development of the Advanced Bi-Lingual Reading Assistant (ABRA) And Authorizing System (ABRA-AS)

Ray Marcelo
Miami University, commons-admin@lib.muohio.edu

MIAMI UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE & SYSTEMS ANALYSIS

TECHNICAL REPORT: MU-SEAS-CSA-1995-007

Development of the Advanced Bi-Lingual Reading Assistance (ABRA) And Authorizing System (ABRA-AS)

Ray Marcelo

School of Engineering & Applied Science | Oxford, Ohio 45056 | 513-529-5928

DEVELOPMENT OF THE ADVANCED BI-LINGUAL READING ASSISTANT (ABRA) AND AUTHORING SYSTEM (ABRA-AS)

by

Ray Marcelo
Systems Analysis Department
Miami University
Oxford, Ohio 45056

Working Paper #95-007 Dec. 1995

DEVELOPMENT OF THE
ADVANCED BI-LINGUAL READING ASSISTANT (ABRA)
AND AUTHORING SYSTEM (ABRA-AS)

Raymundo Marcelo

Committee Members:

Systems Analysis Department

Advisor: Alton Sanders, Ph.D.

Doug Troy, Ph.D.

German, Russian, and East Asian Languages Department

Ruth Sanders, Ph.D.

Audrone Willeke, Ph.D.

Table of Contents

Abstract

Two programs were created to fulfill a contract with the German, Russian, and East Asian Language Department at Miami University. These programs, Advanced Bi-lingual Reading Assistant (ABRA) and Advanced Bi-lingual Reading Assistant - Authoring System (ABRA-AS), create a teaching system designed to implement a "reading strategies" approach to teaching German. ABRA is a reading program where a student reads a given text several times, each time in greater detail. At each level highlighted words or phrases to which the student can obtain translation help. Each level also has comprehension questions. ABRA-AS is the authoring system which creates the levels, the highlights and definitions, and question and answers for the articles read by the ABRA program.

Although this program has been written specifically for this project, it may have uses outside the foreign language classroom.

1 Introduction

ABRA and ABRA-AS are Computer Assisted Language Learning (CALL) programs intended primarily to assist upper level language students with reading comprehension. The programs constituted a research project completed as partial fulfillment of the requirements for the Masters of Systems Analysis degree. The specifications for the project were compiled with the clients, Ruth Sanders, Ph.D. and Audrone Willeke, Ph.D., professors in the Department of German, Russian, and East Asian Languages at Miami University.

With the increase in the number of CALL programs, some teachers are coordinating their course syllabi with such programs as a means of introducing additional practical exercises. Unfortunately, those CALL programs currently available frequently do not coincide with a given instructor's preferences and are often difficult or even impossible to tailor to the instructor's wishes.

Additionally, most programs within this area aid introductory level students rather than advanced students. The programs are not flexible in the amount and kind of information that is displayed to the student. This inflexibility makes classroom use difficult since some information should be suppressed under certain circumstances and augmented in others. In addition, only articles that have been specifically prepared on electronic media may be used with certain programs. This greatly reduces the number of available reading selections and often adds significantly to the cost of using CALL programs since there is likely to be an additional fee for the specially prepared reading material.

Another drawback of current CALL programs is the lack of flexibility in the type of help available to the student user of the program. In particular, my clients wanted to have the flexibility to provide various kinds of assistance to the student and to provide

that assistance in stages or levels. For example, a student might be initially provided with language help such as alternate wordings, grammatical clues, or English translations of selected words or phrases. Once the basic language problems were past, perhaps the assistance would relate to the substantive content of the article itself. Such help would most generally, but not necessarily, be posed as a sequence of questions about the content of the article.

It was consequently my task to work with my clients to define, design, and implement the system according to their specifications.

To implement this system, I used a software process model based upon an initial prototyping stage. "The process model ... assumes that the prototype is developed from an outline system specification, delivered for experiment and modified until the client is satisfied with its functionality. At this stage, a conventional software process is entered, a specification is derived from the prototype and the system re-implemented in a final production version." (Sommerville, 1992)

The paper follows my design process for this project (definition phase, initial prototype, feasibility survey, study phase, and final project). The next section will examine the definition phase of the project. The definition phase contains the program specifications. After the definition phase, I introduce my initial prototype of the program. The feasibility section contains the reactions of the clients toward my initial prototype. The study phase contains the modifications and problems which I encountered when attempting to implement the final system. The final project consists of the variables and the functions within the present program.

2 Definition Phase -- Program Specifications

Initial meetings with my clients defined the project as a set of two programs: a reading program for student reading exercises, and an authoring system for the instructors to create the different levels of assistance for the student. General specifications for both programs and the functional specifications for the reading program were established. The functional specifications for the authoring system were postponed until completion of the reading program.

2.1 General Specifications

2.1.1 User-friendliness

My clients emphasize the need for simplicity and user-friendliness. Many language instructors as well as most language students are computer novices who would be intolerant of a program which is difficult to use.

User-friendliness is a measure of the program's understandability and may aid in the continued use of the program. To create a user-friendly product, the product flows through a user-centered design process. In the user-centered design process, the interface for the system is prototyped by the programmer to create an exchange of ideas (likes and dislikes) between the customer and the programmer. This allows the customers to 'touch and feel' the system and drive the user interface design.

Final user-friendliness of this system will be measured by the student and the teacher. If a student learns the material without a major loss of time learning how to use the program, ABRA is user-friendly. If the instructor can easily create the highlights within a text, giving the correct response and questions and answers, ABRA-AS is user-friendly.

2.1.2 Familiarity

My clients emphasized that the system constructed must be one which did not require any additional hardware that most language instructors did not have at their

disposal. Many language instructors do have access to personal computers and word processing programs.

Since my clients and I were most familiar with DOS-based systems, the programs were developed for that environment. Their preference of word processors was WordPerfect. Since my clients were quite familiar with WordPerfect, the keystroke functions of both programs match those of WordPerfect where possible.

2.1.3 Keyboard Oriented

Due to the emphasis that no additional hardware be required to run this program, both programs were designed with a keyboard interface. An additional mouse interface would be acceptable only if the program could be used without the presence of a mouse. My clients were excellent typists and personally preferred a keyboard to a mouse. Additionally, many computers in their department were older machines that were not equipped with mice. They strongly desired the programs to work on all the computers available to them.

2.1.4. Flexibility

My customers wanted the flexibility to use any article or text that they desired. These programs were designed to present any text supplied by the end-user. The end-user will also have the authority to provide whatever information might be required to understand the general assistance given within a level. No additional text need be purchased to run this program.

2.2 Functional Specification of the Reading Program

2.2.1 Article Length

The reading program must be capable of handling articles of substantial length. In particular, it was to handle articles of up to 50 pages of text.

2.2.2 Screen Format

The screen format will be fully described in the initial prototype section. The screen was divided into two sections: a text section displays the text of the article and the help section displays the assistance created by the teacher. In addition, on-line assistance is available to help the student progress through the various levels.

Color coding would be used to aid students and teachers through the different levels.

2.2.3 Assistance, not Examination

The reading program was strictly a program for assisting the student rather than for testing purposes. The student's performance was not recorded and no attempt was made to provide the ability to record responses. A later version of this program could include this enhancement.

2.2.4 Levels of Assistance

The assistance provided to the student was presented in levels. Initially, the student reads the article with minimal help. Once the article had been read (but almost certainly not entirely comprehended), the student would be presented with some very general questions regarding the content of the article. The student would then re-read the article from the beginning. This time, additional help would be available (the student would be reading at a higher level of detail and comprehension). The process would potentially be repeated for each level of help provided for the given article. The program should be capable of displaying information for at least five levels.

Advanced students of a language must be able to understand and interpret their reading material much as they would if they were reading their native language. They, however, have the additional difficulty of dealing with the language as well as the content. The rationale of the layered design was to give the student assistance in dealing

with both problems, but in an ordered structured way so that only a limited amount of complexity is confronted at any one time.

2.2.5 Types of Assistance

Four types of assistance can be provided to the student:

(1) Word/Phrase Help. Within each help level, the student reads the article from beginning to end. As each page of the article is displayed, the text portion of the screen would contain the text of the article with certain words and phrases marked or highlighted. Those marked or highlighted phrases are referred to as hot phrases. Typically, one of the hot phrases would be marked or highlighted in some additional way (e.g., displayed in another color) so that it would stand out among the hot phrases. This additionally marked or highlighted phrase is referred to as the current hot phrase.

Help related to the current hot phrase is displayed when the current hot phrase is highlighted. Displayed in the help section of the screen will be text that provides assistance in comprehending the phrase. The form of the assistance is defined by the instructor (via the authoring system) and most commonly consists of the appropriate English translation of the phrase given the context which the phrase appears. Of course, the assistance could be any other form of help provided by the instructor.

(2) Question and Answers. In addition to the highlights within each level, questions and answers can be displayed to the student upon completion of examination of the text. The answers to the questions can be multiple choice, explanatory answers, or answers directly from the text. To fulfill the "Assistance, not Examination" decree, answers will be passively displayed by pressing the expected key. This allows the student some time in which to examine the text for answers or ponder the answer before having the answer appear.

(3) Introductory Screen and Status Line. The clients were not interested in emulating a Windows[TM]-like display with a menuing system. In addition, having the

information displayed in the help area was not a wholly feasible situation, so I emphasized that a status line can be useful in aiding the student within the program. An introductory screen would be helpful to give the students some direction prior to reading the article.

(4) Help System. They also wanted a help system for both programs to aid the development and reading process.

2.3 Program Descriptions

Based upon these specifications, the following descriptions of the two programs were developed.

2.3.1 Description of ABRA

ABRA is a reading assistant program consisting of a specific number of levels created by the teacher for a specific article. On each level, the student will be presented with pages of the article with a highlighted phrase and highlightable phrases interspersed on the page. A definition of the highlighted phrase as defined by the teacher is displayed at the bottom of the screen. When the last page of the article is encountered, a series of questions relative to the highlighted phrases and definitions on the level are displayed. Three different types of answers can be given: multiple choice, question/answer, and text answer. As per specifications, the student is shown the definitions of the highlighted phrases as well as the answers to the questions.

2.3.2 Description of ABRA-AS

As per specifications, the teacher uses the authoring system to create the highlights, definitions, and questions and answers.

ABRA-AS is the authoring system that which creates and inputs the information needed by the ABRA program into the required files for each article. The teacher is the primary user of this program and controls what information will be shown to the student.

3 Initial Prototype

The first step in the user-centered design process is the creation of a prototype implementing the specifications created with the customers. This prototype should be quickly built and should emphasize the interface of the system. The customers can make modification suggestions upon this prototype. After approval of the prototype by the customers, this prototype is discarded and the functionality underlying the interface is built.

To expedite the creation of the prototype, I examined Borland's Turbo Vision demo program and discovered that I could adapt this program to partially emulate the final interface to be used by the programs. Being unrealistically optimistic, my futuristic plan was to modify the demo program code to create the final version of the program.

My initial prototype was an object-oriented program where the text windows were text objects and the button windows were button objects. A text object contained the text, the background around the text, and the colors and other functions available for that text. A button window contained the text of the button function and the link to activate the function. Since the demo program was attempting to emulate a Windows[TM]-like system, mouse controls and a menu system were included. The menu system included such functions as getting a new file, saving changes to a file, and exiting from the program.

The screen format of my prototype, shown in Figure 1, consisted of two text windows and four buttons. The two text windows held the text of the article and the text of the definitions, question and answers, and program help. Two thirds of the screen encompassed the text section and the other third encompassed the definition section. The text section was placed above the definition section. Along the right side of the screen, four buttons corresponded to the functions of increasing or decreasing a level and going to the next or to the previous hot word.

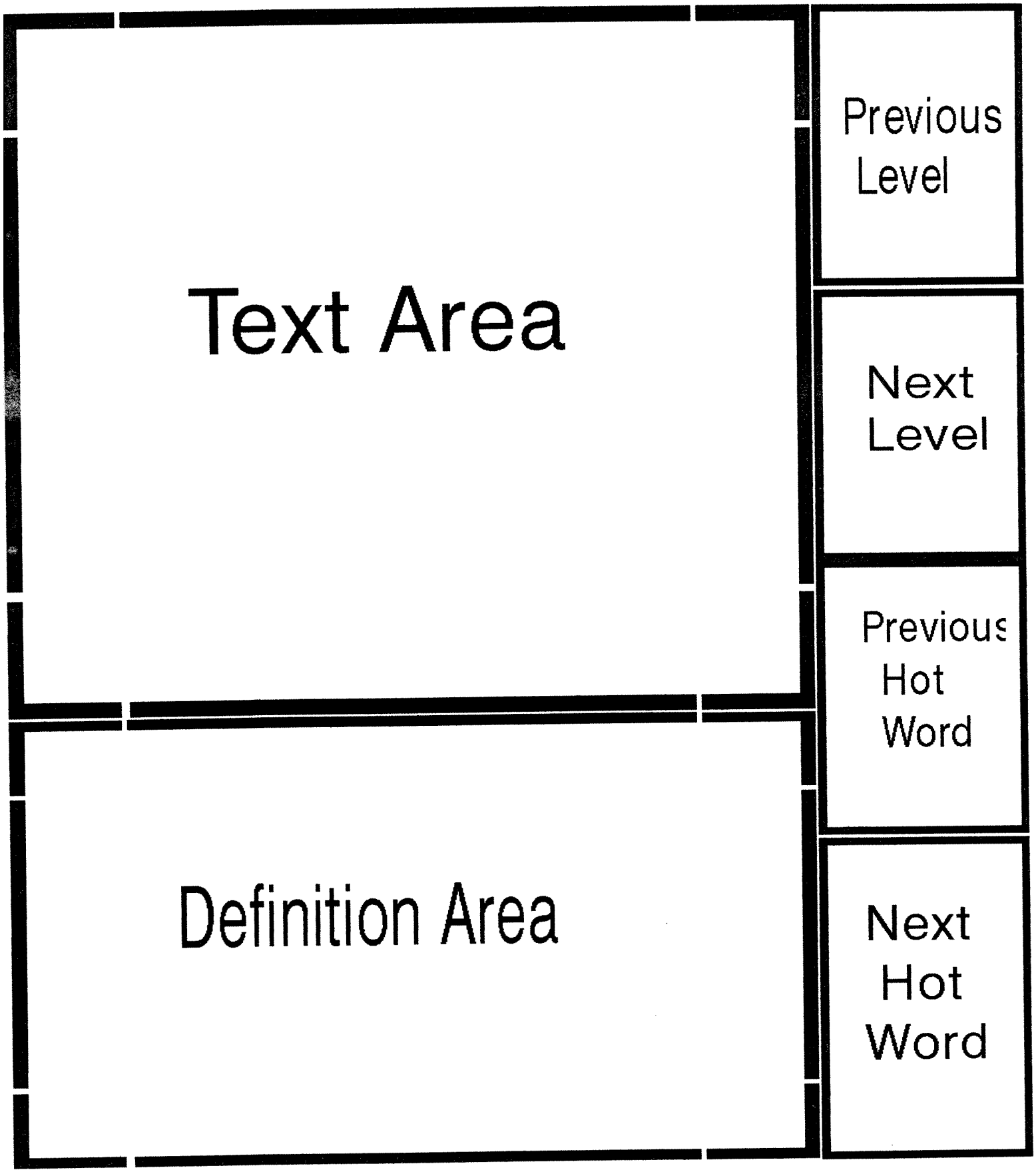

Text Area
Previous Level
Next Level
Previous Hot Word
Definition Area
Next Hot Word

Figure 1

Function keys would be useful for the help system and would be more prevalent within the authoring system.

4 Feasibility Survey

After the initial prototype, a feasibility survey examined the responses of the customers. This feasibility survey allowed the customers to determine if their expectations on the project matched the initial prototype. In addition, this survey allowed me to tailor the project to match my customers' expectations. The following are the results of the survey.

4.1 Interface Change

Although my customers liked the text objects of the project, they disliked the menu system and my button placement.

The menu system allowed the teacher to easily change from reading one article to reading another and was a Windows[TM]-like interface. Other than these functions, the menuing system did not contain enough functions to justify its existence for the customers. Thus, the menu system was discarded. If any text switching is required, restarting the program will be used.

The buttons were placed along the right side of the screen. I was originally uncertain of the placement of the buttons so I placed them along the right hand side to elicit a reaction from the customers. The customers' response was dislike since they felt that the buttons were taking up too much space. Surprisingly, they wanted to change the buttons from visible buttons on the screen to function keys. Given this response, I changed the prototype and loaded most of the functions onto the function keys.

4.2 Color Use

In the initial meetings with the customers, color was a key component of the project interface. Colors delineated the hot words, highlighted or non-highlighted, from the other words in the text. In addition, different color sets could be used for each level and was to be a variable which could be changed by the prospective users of the system.

Although different color schemes would be useful for my customers' situation, different color schemes would not be practical for future customers. The time required to determine the colors for each level would be of more use toward increasing the level of comprehension for the student. If more than one level would be used by the prospective customer, determining the color scheme for each level would detract from the user friendliness of the system.

Since we felt that color was still important within this system, we decided that a global color change would be more cost effective. This allows those customers who do not like the color scheme to choose their own and the time required to make the change would be minimal. The default color scheme is described in a later section.

4.3 Placement of on-line help

An important component which was changed due to the creation of the prototype was the placement of on-line help. Generally, my customers wanted a relatively clean work area. They wanted the text to be displayed in the bulk of the screen and also wanted the system to be intuitive. Unfortunately, they had a little difficulty with how the system progresses from one area to another.

I suggested using a status line to display some on-line help to the students. The customers disliked this idea. We compromised by having the definition section of the screen double as the help section while the definitions were not displayed. This compromise could be done since they had originally decided that the student would have the option to "see" the definition of the highlighted phrase or not to see it.

5 Study Phase

After including the three previous modifications, the prototype was approved. Next the study phase of the project examines the whole project as well as the individual parts. This study phase determines if the project is continuing on schedule and toward the goal of the customers. Enhancements and changes to the project may still be made as

the customers begin seeing the final product or programming difficulties occur. During this phase, the quality of the product and the completion schedule are examined.

This project was to be my first in Turbo Vision and in using object-oriented methods. Given this scenario, I am not surprised that I began having programming difficulty shortly after completing the prototype. My problem stemmed from little practical experience using this programming methodology or the object-oriented programming language within TurboPascal.

The following highlights some of the milestones which were encountered during this phase.

1) Changing from Borland's TurboPascal 6.0 to TurboPascal 7.0. Since I lacked skills within this area, I relied heavily on the demo code and the users' manual. Although these references helped on the short term, lack of adequate description required buying another Turbo Vision guide.

This guide was more helpful in understanding Turbo Vision, but emphasized the workings of TurboPascal 7.0's Turbo Vision. Since more features are within the later version and a memory problem was detected within the TurboPascal 6.0's Turbo Vision, I chose to upgrade my programming language to TurboPascal 7.0.

2) Non-support by Borland. Although I was having some difficulty in programming, I felt that Borland would support their product and get me through the rough spots if required.

Unfortunately, I was wrong in my assumptions. The program was experiencing text blinking at undesirable times caused by a bit being incorrectly set. I called Borland for support and found out that Borland no longer supported Turbo Vision. Two versions of TurboPascal 7.0, one for DOS and one for Windows[TM], are available. Since my customers did not want to use a Windows[TM] product and Object Vision emphasized Windows[TM] development, I chose TurboPascal 7.0 for DOS. Unfortunately, Borland

decided to only support the Windows[TM] version (Object Vision) instead of the DOS version (Turbo Vision).

3) The Addition of the Status Line and Introduction. One of the first programming changes involved the changing mindset of the customers regarding the definition display. As indicated earlier, the placement of on-line help was put into the definition text area of the screen, since the definition would only be displayed when the student wanted to view the definition.

As my customers began seeing the progress of the final product and the extra keystrokes required for displaying the definitions, they decided that the definitions should be always displayed while there are highlighted words on the screen. Since the on-line help was sharing space with the definition, they forgot the commands to traverse through the system when the definition was being displayed.

Although they disliked a status line area, we had no other choice for displaying on-line help. With the addition of a status area, the number of keystrokes made by the student will be decreased and the on-line help would always be available.

4) Switching Programming Technique. The final milestone for this study phase was the inevitable switch from an object oriented methodology to a structured methodology. As indicated earlier, this project seemed upon first review as an ideal candidate as an object-oriented system. This system would have the text itself as a primary object and the highlights and definitions as sub-objects. Other objects could include the question and the various types of answers.

Since I was able to get the prototype quickly coded, I felt that the rest of the project should proceed at the same pace. Unfortunately, I forgot the second rule of user-centered design: dump the prototype after approval. Since I did not code the prototype, I relied too heavily on trying to modify someone else's code and comprehending the process. At the intellectual level, I had no problem understanding what needed to be done and which functions would be involved. My problem stemmed from being able to

call one object with another object since this was not as straightforward as calling a function in a structured method. I ran into the classic black box problem in which I was sending data into the box hoping to get the desired result and unsure why I got the particular response that I did get.

I got stuck within this paradigm and my only recourse to get the project completed was to begin programming using the structural methodology with which I was most familiar. Although I did switch the methodology that I was using, I was still able to incorporate some of the code generated in the object-oriented methodology. The basic functions within many of the original objects were reincorporated within the final project.

6 Project Description

This section describes the functionality of the final project based upon the specifications. The project itself is a multilevel palate in which different pieces of information are stored on different levels. The base level is a screen template which displays a generic background determined by the default colors. On the next level, a stable textual base is required. This textual base includes the text of the article as well as the introduction which will be displayed to the student prior to working on a particular article. Upon this textual base, the hot words, their definitions, and the questions which may be asked about that level of information are erected. I will describe the pieces of this project using the same multilevel design as the final project.

Since this project was designed as freeware, I attempted to make the program easily maintainable. This will allow other programmers to comprehend and modify the program to suit their particular needs. Technical descriptions of the variables, structures, functions, and procedures will also be dispersed through this general description. ABRA-AS, of course, requires more functions than those for ABRA and all functions are described in this section.

The general overview of the functionality and descriptions of the major functions in both programs are given in the User's Manual, Appendix A. Since this section only gives a general overview, any specific steps required to perform any of the major functions are referred to the User's Manual. The technical descriptions and minor modifications are given in the Technical Manual, Appendix B. The two manuals differ based upon the level of detail and text from both manuals can be found within this section. In addition, I added a section of screen examples to clarify many of the functions described in the following sections.

NOTE: An *italicized* word in this section is the actual variable name used within the programs.

6.1 Background Screen Template

The screen template, shown in Figure 2, is divided into three general areas -- the text area, the definition area, and the status line. The **text area** is where the text of the article and the introduction will be displayed. The **definition area** is the area where the definition of the highlighted words and the questions will be displayed. The sizes of the text area and the definition area are dependent upon whether the level uses the *NormTextLines* variable or the *InterlinearLines* variable. *NormTextLines* indicate that around three-fourths of the screen will be text and the other fourth will be the definition. *InterlinearLines* indicate that about two-thirds of the screen will be the text and the rest the definition. The different text and definition area sizes allow for sparse or paragraphical definitions. When the question and answers are in use, the area sizes defaults to the *InterlinearLines* variable. For both the article text and the questions, the program has defined constant values for the left margin and the maximum length of the line.

Screen layout modifications can be made through changing the constant variable values in ABRA and ABRA-AS. These constant variable values are program specific, so changes do not automatically transfer to both programs. Color variable values are chosen from the eight foreground and sixteen background colors available within the DOS environment. The left margin and the line length variable values may also be changed. Upon changing constant variable values, the program(s) must be re-compiled

The following list contains the constant names and default values which may affect the text and definition areas.

Constant Name	Constant Value
MaxTextLines	25
MaxLineLen	80
LtMargin	4
ScrLineLen	72

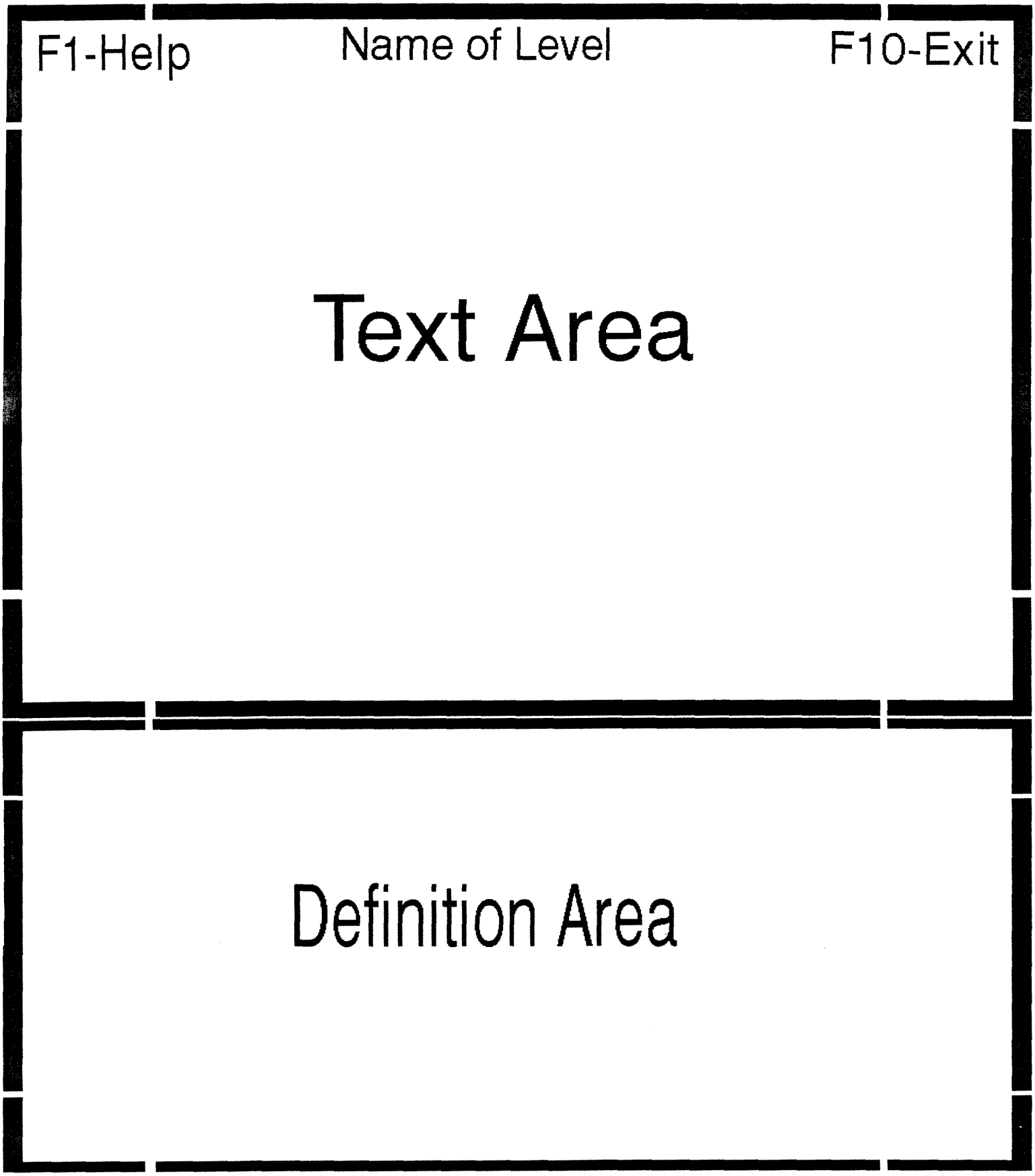


Figure 2

NormTextLines	18
InterlinearLines	15

Constant Name	Color
NormalBackground	Blue
NormalText	Yellow
HelpBackground	Red
HelpText	White

The **status areas** are lines at the top and bottom of the screen which display on-line help for the user. The status line at the top of the screen constantly displays the name of the highlighted word and question level and the keypress choices to exit from the program and to get help. The status line at the bottom of the screen displays the available keystrokes based upon the status of the program.

The status areas also have variable color constants which may be changed.

Constant Name	Color
StatusBackground	Magenta
StatusText	LightCyan
SelectedStatusText	Yellow

6.2 Textual Object

This textual object consists of the text of the article being read. A summary of the purpose, variables, and functions is given in Table 1. Functions and variables used by the textual object are also used by the introduction object. The introduction object will be described in a later section.

6.2.1 The Article Text

The article is the heart of the ABRA and ABRA-AS programs. The article text is a flat file containing the text of the article in extended-ASCII. The text should be extremely stable since the program does not attempt to analyze different words within the text, but uses the position of those words based upon the beginning of the flat file. This

Textual Object

Purpose: To maintain displaying the text of the article.
To perform some of the basic scrolling processes.

Variables: TextBuffer -- Array which contains one "page" of characters
TextPage -- Array which contains the position of the first character of each line within the article
TotalChar -- Long Integer of the total number of characters within the article
TotalLines -- Integer of the total number of lines within the article
Page -- The number of swaps performed to get to the present character
MaxTextSize -- The largest number of characters held within the TextBuffer at any one time
MaxPageSize -- Value used to determine if a new "page" of characters needs to be swapped into the TextBuffer

Functions: InitText -- Initializes text variables
GetText -- Gets the File to be Examined
FindPage -- Finds a particular page within the text
PageUp -- Scrolls the text up one page
PageDown -- Scrolls the text down one page
ScrollUp -- Scrolls the text up one line
ScrollDown -- Scrolls the text down one line
LastChar -- Scrolls back one character
NextChar -- Scrolls to next character
FirstPage -- Goes to the first page of the article
LastPage -- Goes to the last page of the article

Table 1. The Text Object

increases the speed of displaying the text since the inference engine will not become overloaded determining highlighted words.

All article files have an extension of {article name}.TXT, since this is a naming convention used by both programs. I recommend the article be scanned or retyped using WordPerfect or Wordstar with pica or elite type. This is recommended since little reformatting of the article will be required when converting to ASCII using these word processors. Since an extended-ASCII is allowed, diacritical marks over letters will be displayed but bolds and underlines will not. The display of the article should fit based upon the left margin and maximum line length constant variable values.

Since there is no editing of the article text allowed and a stable article base is a requirement, I recommend that the text be pulled into the program and examined for any visual problems prior to adding any layers. Although many articles will successfully port into this program, any article over 75 pages may create a memory problem due to a constraint on the number of characters allowable for the article. In addition, files that are too large may significantly increase the time required to display the text and highlights.

6.2.2 Article Functions

When examining the textual functions, we decided that the functions to be performed on the article were the same as the functions in any word processing program without the ability to edit information shown on the screen. The word processing functions would include the ability to go to the beginning and the end of the file with a keypress (HOME/END) and the ability to scroll up and down in the article. In the ABRA-AS program, the left and right arrow keys are used to help position the cursor to the specific location on the screen. Since cursor position is not a factor in the ABRA program, the left and right arrow keys move between highlighted words on the screen.

6.2.3 Article Variables

The variable structure of the article is a character array (*TextBuffer*) with a maximum of 3000 characters in the array at any time. If the article has more than 3000 characters of text, pages are swapped from the article file. Swapping occurs when the first character's array value on the page is less than 1000 when traveling toward the beginning of the article or when the last character's array value on the page is greater than 2000 when traveling toward the end of the file. Since a number of swappings may occur while reading the article, the programs use a double buffering system to decrease the time required to swap in the appropriate page.

Presently, the article is allowed to have 3000 lines of text. To keep track of the total layout of the article, an array (*TextPage*) contains the array value of the first character in each line. This array would have recorded the actual array value in the character array as if the complete article were saved in resident memory. In addition, a *Page* variable indicates which section of the 3000 characters the article is presently displaying. The array and *Page* are referenced during paging and swapping pages to determine which characters will be displayed.

6.2.4 Multilevel Variables

Five extremely important variables tie the all the other objects together. These five variables are *H*, *TextStat*, *DataStat*, *Stats*, and *ScreenHelp*.

To maintain contact with the lists of highlighted words and question and answers, an *H* record contains pointers to the heads of the list of highlighted words, the display list of highlighted words, and the list of questions with their corresponding answers.

The *TextStat* and *DataStat* variables, control what can be occurring on the screen at a particular time. These variables replaced many Boolean variables which were originally scattered throughout the program. By looking at the combination of the *TextStat* and *DataStat* variable, the programmer can determine the state or status of the program at a particular instance. Table 2 describes the state of the program based upon these variables. As can be seen by the variable values, I attempted to group either the function type or the area type together. In addition, the variable structure is such that additional functions may be later added and some which I can envision being added have also been indicated.

The information on the screen is tracked with a *StatusType* record which contains most of the information required. This record stores the array locations of the first and last character on the screen as well as the present character being pointed at by the cursor. It stores the first and last pointer to the display list of highlighted words and

TextStat	*DataStat*	Description
Intro		
30	0	Intro w/o highlighted words
30	1	Intro w/ highlighted words (Not implemented)
30	6	Creating an intro file
30	7	Modifying an intro file
Highlighted Words		
Normal		
10	-1	Text w/o highlighted words
10	0	Text w/ highlighted words
Highlights		
11	3	Creating a new highlight - MarkON
11	4	Creating a new highlight - MarkOFF
Definitions		
12	5	Editing a new definition
Question and Answers		
21	0	Displaying question (no highlights)
21	5	Editing a new question
22	0	Displaying Text Answer
22	3	Creating Text Answer - MarkON
22	4	Creating Text Answer - MarkOFF
22	5	Creating Text Answer - Starting Condition
23	0	Displaying Multiple Choice Answer
23	3	Creating Multiple Choice Answers - MarkON
23	4	Creating Multiple Choice Answers - MarkOFF
23	5	Creating Multiple Choice Answers - Starting Condition
24	0	Displaying Text Response
24	5	Editing Text Response

Table 2. System State Given *TextStat* and *DataStat*.

stores the current highlighted word on the page and the first and last highlighted words on the page. When the questions and answers are being displayed, this variable will hold a pointer to the current question.

The final required variable structure is a screen matrix array. To create the final template, a character and its corresponding attributes are assigned to a specific location on the screen and stored within a temporary screen matrix. When all screen locations have their values stored in them, the temporary screen matrix is switched with the page presently being displayed and becomes the present page. This decreases flickering of text which may occur with constant writing of information on the screen.

6.3 Hot Word Object

With the article text and the screen background creating a stable base, the hot word objects are overlaid upon that base. Each Hot Word Object contains a highlighted area within the text and a corresponding definition. A summary of the purpose, variables, and functions is shown in Table 3. The degree of complexity between the highlighted area and the definition can be simple, such as definitions or translations, or complex, such as theories behind using one word versus another. It may include vocabulary, explanation, or even sentence structure.

The hot word object has a look and feel similar to a Hypertext object. The difference between the hot word objects of this program and a Hypertext object is the multi-leveled structure in this program. A hot word object may be highlighted in one level and not highlighted in the next. If the same word is highlighted on different levels, different definitions may be associated with that word.

6.3.1 Highlighting Variables

Three variables (*HelpNodeType*, *StringBuffer*, and *TextNodeType*) are required to create and display the highlights.

A *HelpNodeType* is created for each sequential set of characters within a hot word object. The *HelpNodeType* contains the first and last position in the article text array of the highlighted word and the first position in the *StringBuffer* array of the definition. If multiple *HelpNodeTypes* are required for one hot word object, the different

Hot Word Object	
Purpose:	To highlight words or phrases to increase the student's level of comprehension when reading a text. To display a definition for the highlighted area. To maintain the highlights and their definitions.
Variables:	H.HelpNodeList -- Pointer to the beginning of the list of Hot Word Objects HelpNodePtr -- Pointer to a record which contains the beginning and ending position of the highlighted word, pointers to the other Hot Word Objects, and the beginning position of the definition TextNodePtr -- Pointer to a record used when importing a file of highlightable words or phrases and its corresponding definition
Functions:	Previous Highlight -- Go to the previous highlighted word or phrase Next Highlight -- Go to the next highlighted word or phrase GetHelpRange -- Determines the positions of the highlighted words and phrases within the text BeginNewHotWord -- Begins marking a highlighted area EndNewHotWord -- Finishes marking a highlighted area DisplayDefinition -- Display the definition of the highlighted word EditHWDefn -- Edit the definition of the highlighted word NewHotWordFile -- Import a Highlighted Word file into ABRA-AS DeleteHotWord -- Delete a highlighted area and its definition

Table 3. Hot Word Object

parts of the hot word are doubly linked together. For traveling from one *HelpNodeType* to another, each *HelpNodeType* is doubly linked to a list of all the *HelpNodeTypes* for that level. So that all possible highlighted areas are displayed on the screen, each *HelpNodeType* is doubly linked sequentially to other *HelpNodeTypes* based upon the first position of the highlight in the article text.

A *StringBuffer* is a character buffer which contains the text for all the definitions. Since a character buffer requires less overhead to maintain, the text for all the definitions are stored within this array and the *HelpNodeType* determines which definition to display based upon the array position of the first character of the definition. The *StringBuffer*'s maximum number of characters is 1000. This maximum only effects the definition characters on one level.

The *TextNodeType* is implemented when a series of highlights and definitions are imported into the program. This *TextNodeType*, which contains the highlighted word and definition strings, is singly linked. When the highlighted word and definition are displayed, the *TextNodeType* is deleted.

As with the article text, the current hot word and the highlightable text have default color constants associated with them. The following is a list of these color constant defaults:

Constant Name	Color
SelectableBackground	Blue
SelectableText	LightRed
SelectedBackground	Magenta
SelectedText	White

6.3.2 Highlight and Definition Types.

Two types of highlights, the current hot word and other highlightable words, may be present on the screen. Only one current hot word will be displayed though this highlighted word may contain multiple highlighted areas. The current hot word coincides with the definition shown at the bottom of the screen. Highlighted words presented in ABRA are created in ABRA-AS.

In association with each highlighted word, a definition is displayed in the definition area of the screen. The definitions may include the interlinear or the translation of the highlighted word(s), or may involve an analysis of the sentence structure or other analyses. The types of definitions used within any level are defined by the instructor who highlights the various sections.

NOTE: The text of the definition cannot be scrolled. If the definition for that level requires more lines than given by the *NormTextLines* variable, the *InterlinearLines* variable should define the size of the text area.

6.3.3 Highlighting Functions

As is the case with many of the functions within these programs, more functions can be invoked in ABRA-AS than in ABRA. The ability to highlight word(s) within the text and editing of the definitions are the major functions for ABRA-AS. The displaying of the highlights and definitions are the major functions for ABRA. The following lists the functions which may be invoked in either program.

Previous and Next Highlights. Since the input from the student is limited, a cursor was deemed not necessary. The student's use of the arrow keys will allow him/her to traverse through the article. It was also decided that if an actively highlighted word is no longer on the present page, the next available highlighted word on that page will become actively highlighted.

Creating and Editing a Highlighted Area. When highlighting a hot word, the program allows for multiple highlighted areas to be associated with a definition. To create a hot word, the **F4** key starts and ends the marking of a hot word. When starting a hot word, a highlight will begin when the **F4** key is pressed. This starting position will also be used to determine the sequence within the hot word link list. The forward movement arrow keys will continue to mark the hot word part. If the position is not on the starting position, backward movement keys will take away characters previously marked. To mark the beginning or end of an interior hot word part, the **Insert** key is pressed.

Since an interior hot word part may be started prior to the beginning of the hot word, the program acts as if another hot word is sequentially earlier than this one, even though one of the interior highlight areas is actually earlier than the other hot word. This may be useful when discussing the parsing of a sentence.

When creating a new highlighted area, this area should not overlap an existing marked area. Each highlighted area is specifically linked with a definition. If an already

highlighted area is rehighlighted for another definition, the previous link created between the highlighted area and the definitions will be lost.

The program is limited when changing an existing highlight location. If the location of the highlight is incorrect, changing the highlight location involves deleting the highlight and rehighlighting the hot word. The definition would need to be retyped.

Creating the Definition. After highlighting an area, the definition text is typed within the definition area. Using a word processor-like subprogram with minimal functions, the definition will be typed within the definition area. As indicated earlier, no scrolling will be available for the definition area, so the definition text must be contained within the confines of the definition area. Pressing **F5** completes the creation of the definition.

Editing the Definition. The definition can be edited by pressing the **F5** key. The text of the definition will be displayed in the definition area as originally typed. The definition may be repositioned within the definition area or the wording may be changed. If both the hot word and its definition should be deleted, the deletion function should be used. If a new definition is required, the definition should be reedited. After completing editing changes, **F5** saves the changes into the *StringBuffer* array.

Saving Highlighted Word(s) and Definitions. Although the highlighted word(s) and its definition will be displayed when moving through the highlighted words, this change will not become permanent until all the highlights and their corresponding definitions within this level are saved. After making a few changes, the work should be periodically saved through pressing the **F6** key. Since minimal delay is experienced when saving the information, no indication is given that saving is occurring. This allows the program to save changes to a file and to reclaim some of the lost memory space created through changes in the definitions. If a change is made to a particular level and this information has not been saved, the program will prompt the user to save the level when traveling to another level or exits from the program.

Deleting a Hot Word. If a hot word and its definition no longer fit within a particular level, that hot word and its definition can be deleted from the hot word list using the **Delete** key.

Importing a Series of Definitions. In the previous descriptions, highlighting words and creating their definitions performed in real time. With some initial preparations, the highlighted word(s) and its definition could be written to a file and imported to the program. The program will parse through the file alternately displaying the words to be highlighted and its definition. When displayed, the user is given the opportunity to find the word or phrase and mark it as described earlier. After the phrase is highlighted, the definition is written in the definition area and the program is placed in editing mode for the definition. When the entry is completed, the next entry will be displayed. This process will be continued until the end of the file.

When a highlight file is imported into this program, all other highlighted word(s) and their definitions on this level prior to importing will be lost. When all words have been imported to a level, any additions to those highlights will not be lost.

An example of some entries is shown in Table 4. From these examples, each entry requires up to four pieces in information. Two required pieces are the "|" character and the line between entries. The "|" character separates the highlighted phrase and its definition. Since the definition may use more than one line, a blank line is necessary to separate the individual entries.

When creating the file to be imported, the highlighted phrase and definition pieces are optional. It is recommended to use at least one or the other as a reference guide. If an assistant to the instructor creates the highlights and definitions, it is recommended for both the highlighted phrase and the definition to be displayed. If the instructor knows which word or phrase should be highlighted and would like to use another word processor to create the definitions, the highlighted phrase would not be required.

1. Erinnerung| memory

2. mangelnden Zuspruch| deficiency of encouragement

3. Überlieferung| tradition

| across the country

5. geschichtlicher Bewegung unterworfen bleiben| remain subordinate to historical movement

6. ging es . . . doch nur um| it was in actuality only a question of

Table 4. Import Highlights Entries Examples

6.3.4 Structure of a Highlight File

Although I indicate the structure of the highlighted file in the following section, this information is transparent to the user since the authoring program creates the structure of the file during an interactive session.

An example of the contents of the highlighted words file is as follows:

<<363,373,420,429>[27,3]first line of definition[27,4]second line of definition>

In the example, a few sets of brackets are used. The outer <> are used to delimit the highlighted word entry. The inner <> are used to indicate which characters are to be highlighted. The numbers within the inner <> are number pairs which indicate the beginning and ending character to be highlighted. The number represents the array location of the character relative to the first character in the article.

The [] are used to indicate the location within the definition area where the definition line will begin. The definition line is displayed following the []. When

additional lines are required for the definition, combinations of [] and definition lines are used.

6.4 Question Object

The question object works in conjunction with the hot word object to aid the student in comprehending the article read. A summary of the question object's purpose variables, and functions is given in Table 5. With the question objects, the teacher is able to allow the student to gauge his or her own progress through answering the question within their own minds.

Question Object
Purpose: To display a question and its corresponding answer To create and modify question and answers for a level To maintain the question and answers
Variables: H.Questions -- Pointer to the beginning of a list of questions QuestionBuffer -- Array of characters which contain the text for the questions and the answers QuestionPtr -- Pointer to a record of which contains the position of the first character of the question and the first character of the answer within the StringBuffer array, and pointers to other Question Objects within the question list
Functions: PreviousQuestion -- Go to the previous question NextQuestion -- Go to the next question GetLevelQuestions -- Reads questions from file and populate question list DisplayQuestion -- Display question DisplayAnswer -- Display answer FirstQuestion -- Start at the first question in the list GetNewQuestion -- Create a new question EditQuestion -- Edit the text of a question GetMultipleChoiceAnswer -- Get the multiple choice answer QandAAnswer -- Get a text response for the question BeginNewAnswer -- Begin marking an answer which is within the article EndNewAnswer -- Finish marking an answer which is within the article DeleteQuestion -- Delete question SaveLevelQuestions -- Save all questions for this level to a file

Table 5. Question Object

When the student traverses to the end of the article, the student may proceed into the questions by pressing the **F9** key.

6.4.1 Question Variables

Up to two variables can be used for question and answers. A character array similar to the highlights character array is used to store the question and the answers. All the questions for a level are stored within a doubly linked list composed of *QuestionTypes*. These *QuestionTypes* hold the beginning position of the question and the beginning position of the answer. The program knows to stop entering characters when the "|" character is encountered for the questions. For the answers, the ">" character is used to determine the end of the answer.

The question have default color constants associated with them. These defaults involve those answers which are displayed in the definition area of the screen. Any answer displayed in the text area of the screen uses the defaults of the Current Hot Word. The following is a list of these color constant defaults:

Constant Name	Color
AnswerBackground	LightGray
AnswerText	Red

6.4.2 Question Type

Each level has its own set of questions to test the student's comprehension of the article. When a question is displayed the student has the option to examine the text of the article to find the answer.

6.4.3 Answer Type

Three types of answers, multiple choice, explanation, and text referenced, can answer the question given. Multiple choice allows a choice among the answers from a multiple choice type question. Explanation can be used when a True or False question

with an explanation is given. An explanation can also be used if the question requires a written response. Text Referenced answer will be used if the answer to the question can be answered by highlighting a section of text within the article. The professors decided that the questions will be answered passively. By pressing a key, the student will have the answer displayed.

6.4.4 Question Functions

In the ABRA program, questions and answers will be displayed after the student reaches the bottom of the article. When a question is displayed, a keypress will display the answer. When an answer is displayed, a keypress will display the next question. If the answer is the last question for that level, a keypress will go to the next level.

In ABRA-AS, the question and answers are created. The teacher has the option to create the question and answers anytime he or she is working on a particular level. The teacher should have an idea what question they would like to have asked and the order in which they will ask them. There is no function to reorder questions. New questions are tacked onto the end of the question list.

Previous and Next Question. The student's use of the **TAB** and the **SHIFT-TAB** key will allow him/her to traverse through the questions. It was also decided that within ABRA and after the answer is displayed, the next question is displayed when the **Enter** key is pressed.

Creating a Question. As with the highlighted phrases, the questions may be edited, reedited, or deleted. To add a new question to the end of the question list, the teacher traverses to one question after the end of the list. This will place the teacher into the edit mode for the question.

Creating an Answer. When the question is edited, the program asks the teacher for the answer type. If the answer type is a multiple choice question, the cursor will be placed at the beginning of the multiple choice question. Highlighting the multiple choice

answer(s) is similar to highlighting a phrase. If the answer is an explanation, the cursor will be placed at the end of the question and the answer could then be edited. Finally, if the answer is text referenced, the cursor will be placed in the text window. The teacher can move the cursor to the location which will be highlighted and highlight the answer in a manner similar to highlighting a phrase.

Editing the Question and Answers. After the question and answer is written, the question and answer may be re-edited. Since there is a possibility that the type of answer for the question may be changed, both the question and the answer will be re-edited.

Saving the Questions and Answers. As with the highlighted phrases, the teacher should periodically save his/her work. This allows the program to save changes to a file and to reclaim some of the lost memory space created through changes in the question and answers. As with other saving by the program, there may be a minimal delay resulting from saving the information.

Deleting a Question and Answer. If a question and its answer no longer fit within a particular level, that question and its answer can be deleted from the question list using the **Delete** key.

Importing a Series of Questions. As with the highlighted phrases, the questions and answers are written for a particular level at the time that that level is being created. To speed up the process and allow definite thought to the questions, the user can create a file which can import the questions and possibly the answers to the questions. As with the highlighted phrases, previous changes will be LOST when the program imports the question and answer file. Any additional changes to the questions on a level may be made after importing the questions from the file.

Examples of possible imported question and answers are given in Table 6. For all imported questions, the program asks the teacher which type of answer is required for that question. The teacher should also be aware of the number of lines required for each question. There is no built-in capability to scroll within the questions.

1. What is the theme of the article? a) The importance of history as national memory; b) The centrality of the Nazi period to German history; c) The responsibility of Germany for European stability. True or false? If false, what is true instead? 2. The Germans refuse to accept that they are subject to historical forces.|F; The Germans discovered that even the Federal Republic and the world system of which it is a part are subjugated to historical movement 3. The answer to this question is in the text

Table 6. Possible Imported Question Examples

Question 1 is an example of a multiple choice entry. The teacher will be asked to highlight the answer(s). Question 2 is an example of an explanation The "|" separates the question from the answer. Upon importing the question with an answer, the answer is displayed after the question is displayed and repositioned. Question 3 is an example of a text response.

This process will be continued until the end of the file.

6.4.5 Example of Question File

An example of the contents of a question file is as follows:

<?}1. What is the theme of the article?}} a) The importance of history as national memory;} b) The centrality of the Nazi period to German history;} c) The responsibility of Germany for European stability.|[44,91]> <?2. This is where a question will go.|[10,3]This is the answer> <?3. What would be the answer to question in text?|<150,160>

As with the highlights file, the "|" separates the question from the answer. The outer **<?>** are used to delimit the question entry. For the question, new lines are indicated by a "}". Square brackets can indicate either position within the definition area or characters to be highlighted. Position within the definition area is understood when text follows the "]". If no text after the "]", then this indicates a multiple choice question.

If the answer values are within <> then this indicates the character array location of the article text which should be highlighted. Within the []s or <>s are pairs of numbers which give the beginning and ending array location of the answer which should be highlighted.

6.5 Level Object

The hot word objects and question objects are combined within a level object. The purpose, variables, and functions of a level object are summarized in Table 7. The maximum number of level objects for any article text is limited to 99 LEVELS of information. This restriction is due to the DOS naming convention, since the last two characters of the name extension are used to indicate the level number.

Level Object	
Purpose:	Maintains the present statistics of the screen and variables Maintains the ends of the TextBuffer and QuestionBuffer Maintains the List of HotWord Objects and the List of Question Objects
Variables:	LevelPtr -- Pointer to a record which contains the level name, pointers to the heads of the DisplayList, the HelpNodeList, and the Questions, and the position of the last character in the StringBuffer and the QuestionBuffer Status -- Record of the current, first, and last position on the screen, the current, first, and last line on the screen, the present coordinate values. Also, the location of the current, first, and last Hot Word Object on the screen, and the first and last Hot Word Object displayed on the screen. Finally, it also points at the current question within the question list
Functions:	PrevHelpLevel -- Go to the previous help level NextHelpLevel -- Go to the next help level InitLevel -- Initializes the variables in a level LoadFile -- Gets the file to be read or to be highlighted GetLevel -- Loads the lists, reading information from file SaveLevel -- Save the highlight word information onto disk

Table 7. Level Object

6.5.1 Level Variables

For each level, three structures are used to maintain level and program cohesiveness.

The main level structure is a doubly linked list (*LevelPtr* of *LevelRec*) that contains the filenames of the highlighted words and the questions, holds the level name and the Boolean value used to determine if the highlighted word area will conform to *NormTextLines* or *InterlinearLines*, and points to the previous and the next levels.

When a particular level is chosen, the *LevelType* record holds the name of the present level, as well as pointers to the beginning of the highlighted words and the questions lists.

6.5.2 Level Functions

In the ABRA program, a new level will be displayed after the student finishes with all the questions. When the last question and answer for one level is displayed, a keypress will go to the next level. When the last question and answer of the last level is displayed, the program will exit with a keypress.

In ABRA-AS, a new set of highlighted words and a new set of questions and answers are created. New files are created for the highlighted words and questions.

Previous and Next Levels. The student's use of the **F2** and **F3** keys will allow him/her to traverse through the levels. It was also decided that within ABRA, the next level is displayed when the **Enter** key is pressed after the last answer is displayed.

Creating a New Level. A new level is created when the level pointer is increased or the program has just begun. When a level is created, the name of the file for the highlighted words or questions is attached to a particular level when a word is highlighted or a question is entered. When a particular level is exited the file names for the highlighted words and questions are saved to disk. Each level can be given a unique name which is displayed centered in the top status area. This name can indicate what the

highlighted words and question and answers are attempting to teach. The name of the level is given before entering the highlighted words.

Saving Level Changes. Upon completion of creating all the levels, ABRA-AS automatically constructs the filename file. This makes exiting from the program extremely important. If some method other than the **F10** key is used to exit from the program, the edits to the highlights and questions may be on disk, but the level driver for that article may not be accessible. No indication is given that the this file is being created and the delay experienced by saving the information will be unnoticeable.

6.6 Introduction Object

The introduction allows the teacher to give the student an introduction to the keypresses of the program and may give the student some insight on the nature of the article being read. The introduction is displayed in the text area prior to displaying the article and uses the same structures and functions as the article.

In ABRA, the introduction will be displayed upon entering the name of the article to be examined. By pressing the appropriate key, the introduction will be replaced by the text of the article. Editing and saving the introduction can only be performed in ABRA-AS.

6.7 Files File

For each article, the ABRA-AS program creates a number of files required by the ABRA program for each article. The ".FLS" file contains the names of the files referenced when this article is read. Creating and saving of this file is automatically performed by the ABRA-AS program.

The ".FLS" files is an ASCII file created by the authoring system. The file nomenclature takes the article file name and adds the ".FLS" extension. Thus, each article has one and only one set of referenced files.

If the teacher would like to test different teaching strategies, he/she would need to copy the article into another file name and re-highlight the phrases and re-edit the

questions. If the same highlights and questions are used, a resourceful teacher can copy the article file using another file name and create the ".FLS" file.

6.7.1 Layout of the ".FLS" file

The ".FLS" file contains pairs of lines which layout the names of the article file, the introduction file, and the file(s) for each level. The first line indicates the file type and the second line indicates the name(s) of the file(s).

An example of this file demonstrates all the elements that may possibly be in this file. The numbers in the file are used to denote line numbers and are not part of the file:

```
1.  @I Introductory Level
2.  GER.$00
3.  @T
4.  GER.TXT
5.  @H N Level 1 -- Vocabulary
6.  GER.$01|GER.&01
7.  @H I Level 2 -- Highlighted Phrases
8.  GER.$02|
9.  @H N level 3
10. |GER.&03
11.  @H I level 4
12. |
```

Lines 1 and 2 show an example of an introductory item. "@I" signifies that the file name on the second line is the introduction file. The file extension for the introduction file is ".$00". After the "@I", the rest of the line is used to house the name which will be displayed.

Lines 3 and 4 show an example of an article item. "@T" signifies that the file name on the second line is the article file. No additional information will be shown on that line. The file extension for the article file is ".TXT". The name of this file is the same name as the ".FLS" file.

The rest of the lines show examples of the possible level items. "@H" signifies that the file(s) on the second line houses the highlighted phrases file and the questions

files. Two other pieces of information are present on the first line. The "N" or "I" indicates whether the *Interlinear* variable is false or true. The other piece of information is the name of the level. On the second line, the highlighted phrases file extension is "$"<level number>. The question file extension is "&"<level number>. A "|" separates the two files. If no highlighted phrases are used on that level, the second line begins with an "|". If no questions are used on that level, the second line ends with an "|". If no highlighted phrases or questions are used on that level, the second line begins and ends with "|".

6.7.2 Level Variables

The programs references three structures to store the file names of the article, introduction, and levels. A *FilesRec* record is the structure which contains the names of the files being referenced (the introduction file, the article file, a pointer to the first level file name, and a pointer to the current level file name). For the introduction and the article files, a *FilesPtr* record stores the name of the file. A header is stored in the *FilesPtr* record for an introduction file. The third structure, the *LevelRec* holds the file name(s) and other information required for each level. This structure will be examined in a later section.

6.8 Background

There are many background functions which occur within the programs which are too involved to go into in this document. If the teacher or maintenance programmer is interested in these background functions, the source code should be referenced.

In addition to the background functions I wrote, there were many functions I had written help in creating. Throughout using ABRA-AS, an editor function is used to edit the definitions and the question and answers. Though I would like to take credit for writing the editor, I used code adapted from O'Brien and Nameroff's TurboPascal 7: the

Complete Reference. This book was invaluable for the editor, for cursor control, as well as displaying information upon the screen.

7 Future Additions to the Program

Although this program presently fulfills the needs initially set out by the clients, I see a few additions which may be added to the program to fulfill the needs of other users.

Firstly, a simple addition would enhance the ability to create the definitions and questions for each level. This enhancement is a paging function for those files which would be similar to the paging function already in place with the article functions.

Another addition would change the interface to be closer to the standard Windows[TM] interface including mouse functions, scrolling bars, and icons to represent the different functions. This would give greater flexibility to the program and would appease those users who love working in Windows[TM]. In that same vein, the program may be ported into an object-oriented language (such as, Turbo Vision or a Visual language) in which the article is an object.

Other changes to the program involve the teacher's ability to use the program for different purposes. If the teacher would like to record answers to the questions, the program could be made flexible enough to incorporate some recording function. Statistics could be performed upon the question answers and a grade can be given. Other statistics that may be of interest could possibly monitor the keypresses and the time between keypresses of the students. This ability may lead to insights to the effectiveness of the teacher's teaching style and to the effectiveness of the definitions for the student.

The final change is in an area which we discussed but for which no final decision was made: the ability to remember where a person was if he/she previously exited before finishing all the levels.

Finally, another program may be placed on top of these two programs to change the parameters within the program and recompile those units that need recompiling.

Within this additional program the parameters may be changed and the teacher would be allowed to view these changes prior to saving these changes to disk. In addition, this program may make alternate color scheme data which would allow the teacher to use the program but have different colors for the text depending upon the discussion which is going on in class. An example of this would be if the teacher's class format examined the same order but with different topics (such as psychological illnesses -- symptoms, treatments). In this case, either the symptoms could be in the same color scheme or the illness may be in the same color scheme. This could also be expanded to include screen layout (location of line separation between parts, margins of the text, etc.).

8 Conclusion

A program was required to fill the gap in computer aided instruction for advanced students in foreign language departments. An authoring system was required for the previous program. ABRA and ABRA-AS fills the gap within the CAI arena and is flexible enough to be used in other fields.

Although this program has been written specifically for foreign language teachers and upper level college students, this program may also be used in other settings in which highlighting of words or phrases is required and an explanation given.

References

Sommerville, Ian. (1992). Software Engineering. Workingham: Addison-Wesley Publishing Company. p. 107.

Appendix A

User's Manuals

Appendix A1

User's Manual

Introduction

ABRA and ABRA-AS are Computer Assisted Language Learning (CALL) programs intended primarily to assist upper level language students with reading comprehension. The programs constituted a research project completed as partial fulfillment of the requirements for the Masters of Systems Analysis degree. The specifications for the project were compiled with the customers, Ruth Sanders, Ph.D. and Androne Willeke, Ph.D., professors in the Department of German, Russian, and East Asian Languages at Miami University.

Description of ABRA

ABRA is a reading assistant program consisting of a specific number of levels created by the teacher for a specific article. On each level, the student will be presented with pages of the article with a highlighted phrase and highlightable phrases interspersed on the page. A definition of the highlighted phrase as defined by the teacher is displayed at the bottom of the screen. When the last page of the article is encountered, a series of questions relative to the highlighted phrases and definitions on the level are displayed. Three different types of answers can be given: multiple choice, question/answer, and text answer. As per specifications, the student is shown the definitions of the highlighted phrases as well as the answers to the questions.

Description of ABRA-AS

As per specifications, the teacher uses the authoring system to create the highlights, definitions, and questions and answers.

ABRA-AS is the authoring system that which creates and inputs the information needed by the ABRA program into the required files for each article. The teacher is the primary user of this program and controls what information will be shown to the student.

System Requirements. Any DOS-based machine should work with these programs. A 386 or higher is recommended since the page swapping will be negligible on those machines.

Article Requirements. The article is the heart of the ABRA and ABRA-AS programs. The article text is a flat file containing the text of the article in extended-ASCII. The text should be extremely stable since the program does not attempt to analyze different words within the text, but uses the position of those words based upon the beginning of the flat file. This increases the speed of displaying the text since the inference engine will not become overloaded determining highlighted words.

All article files have an extension of {article name}.TXT, since this is a naming convention used by both programs. I recommend the article be scanned or retyped using WordPerfect[TM] or Wordstar[TM] with pica or elite type. This is recommended since little reformatting of the article will be required when converting to ASCII using these word processors. Since an extended-ASCII is allowed, diacritical marks over letters will be displayed but bolds and underlines will not. NOTE: If the article is written using Word for Windows[TM] and uses the symbol table to create the diacritical marks over the letters, these symbols DO NOT coincide with the extended-ASCII value. The display of the article should fit based upon the left margin and maximum line length constant variable values.

Since there is no editing of the article text allowed and a stable article base is a requirement, I recommend that the text be pulled into the program and examined for any visual problems prior to adding any layers. Although many articles will successfully port into this program, any article over 75 pages may create a memory problem due to a constraint on the number of characters allowable for the article. In addition, files that are too large may significantly increase the time required to display the text and highlights.

Program Functions

Article Functions

When examining the textual functions, we decided that the functions to be performed on the article were the same as the functions in any word processing program without the ability to edit information shown on the screen. The word processing

functions would include the ability to go to the beginning and the end of the file with a keypress (HOME/END) and the ability to scroll up and down in the article. In the ABRA-AS program, the left and right arrow keys are used to help position the cursor to the specific location on the screen. Since cursor position is not a factor in the ABRA program, the left and right arrow keys move between highlighted words on the screen.

Highlighting Functions

As is the case with many of the functions within these programs, more functions can be invoked in ABRA-AS than in ABRA. The ability to highlight word(s) within the text and editing of the definitions are the major functions for ABRA-AS. The displaying of the highlights and definitions are the major functions for ABRA. The following lists the functions which may be invoked in either program.

Previous and Next Highlights. Since the input from the student is limited, a cursor was deemed not necessary. The student's use of the arrow keys will allow him/her to traverse through the article. It was also decided that if an actively highlighted word is no longer on the present page, the next available highlighted word on that page will become actively highlighted.

Creating and Editing a Highlighted Area. When highlighting a hot word, the program allows for multiple highlighted areas to be associated with a definition. To create a hot word, the **F4** key starts and ends the marking of a hot word. When starting a hot word, a highlight will begin when the **F4** key is pressed. This starting position will also be used to determine the sequence within the hot word link list. The forward movement arrow keys will continue to mark the hot word part. If the position is not on the starting position, backward movement keys will take away characters previously marked. To mark the beginning or end of an interior hot word part, the **Insert** key is pressed.

Since an interior hot word part may be started prior to the beginning of the hot word, the program acts as if another hot word is sequentially earlier than this one, even

though one of the interior highlight areas is actually earlier than the other hot word. This may be useful when discussing the parsing of a sentence.

When creating a new highlighted area, this area should not overlap an existing marked area. Each highlighted area is specifically linked with a definition. If an already highlighted area is rehighlighted for another definition, the previous link created between the highlighted area and the definitions will be lost.

The program is limited when changing an existing highlight location. If the location of the highlight is incorrect, changing the highlight location involves deleting the highlight and rehighlighting the hot word. The definition would need to be retyped.

Creating the Definition. After highlighting an area, the definition text is typed within the definition area. Using a word processor-like subprogram with minimal functions, the definition will be typed within the definition area. As indicated earlier, no scrolling will be available for the definition area, so the definition text must be contained within the confines of the definition area. Pressing **F5** completes the creation of the definition.

Editing the Definition. The definition can be edited by pressing the **F5** key. The text of the definition will be displayed in the definition area as originally typed. The definition may be repositioned within the definition area or the wording may be changed. If both the hot word and its definition should be deleted, the deletion function should be used. If a new definition is required, the definition should be reedited. After completing editing changes, **F5** saves the changes into the *StringBuffer* array.

Saving Highlighted Word(s) and Definitions. Although the highlighted word(s) and its definition will be displayed when moving through the highlighted words, this change will not become permanent until all the highlights and their corresponding definitions within this level are saved. After making a few changes, the work should be periodically saved through pressing the **F6** key. Since minimal delay is experienced when saving the information, no indication is given that saving is occurring. This allows

the program to save changes to a file and to reclaim some of the lost memory space created through changes in the definitions. If a change is made to a particular level and this information has not been saved, the program will prompt the user to save the level when traveling to another level or exits from the program.

Deleting a Hot Word. If a hot word and its definition no longer fit within a particular level, that hot word and its definition can be deleted from the hot word list using the **Delete** key.

Importing a Series of Definitions. In the previous descriptions, highlighting words and creating their definitions performed in real time. With some initial preparations, the highlighted word(s) and its definition could be written to a file and imported to the program. The program will parse through the file alternately displaying the words to be highlighted and its definition. When displayed, the user is given the opportunity to find the word or phrase and mark it as described earlier. After the phrase is highlighted, the definition is written in the definition area and the program is placed in editing mode for the definition. When the entry is completed, the next entry will be displayed. This process will be continued until the end of the file.

When a highlight file is imported into this program, all other highlighted word(s) and their definitions on this level prior to importing will be LOST. When all words have been imported to a level, any additions to those highlights will not be lost.

Question Functions

In the ABRA program, questions and answers will be displayed after the student reaches the bottom of the article. When a question is displayed, a keypress will display the answer. When an answer is displayed, a keypress will display the next question. If the answer is the last question for that level, a keypress will go to the next level.

In ABRA-AS, the question and answers are created. The teacher has the option to create the question and answers anytime he or she is working on a particular level. The teacher should have an idea what question they would like to have asked and the order in

which they will ask them. There is no function to reorder questions. New questions are tacked onto the end of the question list.

Previous and Next Question. The student's use of the **TAB** and the **SHIFT-TAB** key will allow him/her to traverse through the questions. It was also decided that within ABRA and after the answer is displayed, the next question is displayed when the **Enter** key is pressed.

Creating a Question. As with the highlighted phrases, the questions may be edited, reedited, or deleted. To add a new question to the end of the question list, the teacher traverses to one question after the end of the list. This will place the teacher into the edit mode for the question.

Creating an Answer. When the question is edited, the program asks the teacher for the answer type. If the answer type is a multiple choice question, the cursor will be placed at the beginning of the multiple choice question. Highlighting the multiple choice answer(s) is similar to highlighting a phrase. If the answer is an explanation, the cursor will be placed at the end of the question and the answer could then be edited. Finally, if the answer is text referenced, the cursor will be placed in the text window. The teacher can move the cursor to the location which will be highlighted and highlight the answer in a manner similar to highlighting a phrase.

Editing the Question and Answers. After the question and answer is written, the question and answer may be re-edited. Since there is a possibility that the type of answer for the question may be changed, both the question and the answer will be re-edited.

Saving the Questions and Answers. As with the highlighted phrases, the teacher should periodically save his/her work. This allows the program to save changes to a file and to reclaim some of the lost memory space created through changes in the question and answers. As with other saving by the program, there may be a minimal delay resulting from saving the information.

Deleting a Question and Answer. If a question and its answer no longer fit within a particular level, that question and its answer can be deleted from the question list using the **Delete** key.

Importing a Series of Questions. As with the highlighted phrases, the questions and answers are written for a particular level at the time that that level is being created. To speed up the process and allow definite thought to the questions, the user can create a file which can import the questions and possibly the answers to the questions. As with the highlighted phrases, previous changes will be LOST when the program imports the question and answer file. Any additional changes to the questions on a level may be made after importing the questions from the file.

Level Functions

In the ABRA program, a new level will be displayed after the student finishes with all the questions. When the last question and answer for one level is displayed, a keypress will go to the next level. When the last question and answer of the last level is displayed, the program will exit with a keypress.

In ABRA-AS, a new set of highlighted words and a new set of questions and answers are created. New files are created for the highlighted words and questions.

Previous and Next Levels. The student's use of the **F2** and **F3** keys will allow him/her to traverse through the levels. It was also decided that within ABRA, the next level is displayed when the **Enter** key is pressed after the last answer is displayed.

Creating a New Level. A new level is created when the level pointer is increased or the program has just begun. When a level is created, the name of the file for the highlighted words or questions is attached to a particular level when a word is highlighted or a question is entered. When a particular level is exited the file names for the highlighted words and questions are saved to disk. Each level can be given a unique name which is displayed centered in the top status area. This name can indicate what the

highlighted words and question and answers are attempting to teach. The name of the level is given before entering the highlighted words.

Saving Level Changes. Upon completion of creating all the levels, ABRA-AS automatically constructs the filename file. This makes exiting from the program extremely important. If some method other than the **F10** key is used to exit from the program, the edits to the highlights and questions may be on disk, but the level driver for that article may not be accessible. No indication is given that the this file is being created and the delay experienced by saving the information will be unnoticeable.

Introduction Function

Only one major function is required for the introduction.

Creating an Introduction. When the user would like to create an introduction which will give the student some insights as to the nature of the article which will be read or any instructions which may be required for use of this program. If the user would not like to create an introduction through the program, an extended-ASCII introduction can written using another word processor and the file named {.TXT file name}.&00 and the introduction line added to the {.TXT file name}.FLS file. Of course, if the user uses this method, he/she should check to make sure that the introduction text fits well within the screen. To edit an introduction, press **F8.** When editing is completed, press **F5**.

Appendix A2

Quick Reference Guide

ABRA Quick Reference Guide

Within Text Movements -- Self-explanatory
LeftArrow, RightArrow, UpArrow, DownArrow, PageUp, PageDn

To Go To the Beginning of the Text <Home>

To Go To the End of the Text <END>

To Get Keyboard Help
<F1>

To Exit From the Program
F10

Level Movement
F2 -- Go to the Previous Level
F3 -- Go to the Next Level

Movement From Highlights to and from Questions
Press <F9>

Movement among Highlights or Questions
TAB -- In Highlights - Go to Next Highlighted Word or Phrase
In Questions - Go to Next Question
SHIFT-TAB -- In Highlights - Go to Previous Highlighted Word or Phrase
In Questions - Go to Previous Question

Creating a Highlight (In Text, In Multiple Choice Answer, or In Text Answer)
1) Position the cursor to the first letter of the text to be highlighted.
2) Press <F4>. (This begins the highlight).
3) Arrow to the end of the highlighted word.
4) If the end completes the highlight, go to Step #10.
5) If multiple highlighted parts, press <INSERT>. (Ends highlight)
6) Position the cursor to the first letter of the next section to be highlighted.
7) Press <INSERT>. (Begins a new highlighted section).
8) Arrow to the end of the highlighted section.
9) If additional highlighted sections required, go to Step #5.
10) Complete highlight by pressing <F4>.

To complete Editing (For Creating Definitions, Creating Questions, Creating Answers, Creating the Intro)

1) Editing will commence when triggered
   a) after highlighting a hot word
   b) when a new question is to be created
   c) when a Question & Answer answer is created
2) Type in your definition, question, answer, or intro.
3) Press <F5> when completed.

To Modify a Definition or Question

Press <F5>

To Create an Intro

Press <F8>

To save Highlights or Questions

Press <F7>

Appendix B

Technical Manual

Technical Manual

Program Description

ABRA and ABRA-AS are multilevel palates in which different pieces of information are stored on different levels. The base level is a screen template which displays a generic background determined by the default colors. On the next level, a stable textual base is required. This textual base includes the text of the article as well as the introduction which will be displayed to the student prior to working on a particular article. Upon this textual base, the hot words, their definitions, and the questions which may be asked about that level of information are erected.

Major Data Structures

Background Screen Template

The screen template is divided into three general areas -- the text area, the definition area, and the status line. The **text area** is where the text of the article and the introduction will be displayed. The **definition area** is the area where the definition of the highlighted words and the questions will be displayed. The sizes of the text area and the definition area are dependent upon whether the level uses the *NormTextLines* variable or the *InterlinearLines* variable. *NormTextLines* indicate that around three-fourths of the screen will be text and the other fourth will be the definition. *InterlinearLines* indicate that about two-thirds of the screen will be the text and the rest the definition. The different text and definition area sizes allow for sparse or paragraphical definitions. When the question and answers are in use, the area sizes defaults to the *InterlinearLines* variable. For both the article text and the questions, the program has defined constant values for the left margin and the maximum length of the line.

Screen layout modifications can be made through changing the constant variable values in ABRA and ABRA-AS. These constant variable values are program specific, so changes do not automatically transfer to both programs. Color variable values are chosen from the eight foreground and sixteen background colors available within the DOS

environment. The left margin and the line length variable values may also be changed. Upon changing constant variable values, the program(s) must be re-compiled.

The following list contains the constant names and default values which may affect the text and definition areas.

Constant Name	Constant Value
MaxTextLines	25
MaxLineLen	80
LtMargin	4
ScrLineLen	72
NormTextLines	18
InterlinearLines	15

Constant Name	Color
NormalBackground	Blue
NormalText	Yellow
HelpBackground	Red
HelpText	White

The **status areas** are lines at the top and bottom of the screen which display on-line help for the user. The status line at the top of the screen constantly displays the name of the highlighted word and question level and the keypress choices to exit from the program and to get help. The status line at the bottom of the screen displays the available keystrokes based upon the status of the program.

The status areas also have variable color constants which may be changed.

Constant Name	Color
StatusBackground	Magenta
StatusText	LightCyan
SelectedStatusText	Yellow

Global Variables

Five variables are extremely important in tying the multiple levels together into a coherent mass.

To maintain contact with the lists of highlighted words and question and answers, an *H* record contains pointers to the heads of the list of highlighted words, the display list of highlighted words, and the list of questions with their corresponding answers.

Two variables, the *TextStat* and *DataStat*, control what can be occurring on the screen at a particular time. These variables replaced many Boolean variables which were originally scattered throughout the program. By looking at the combination of the *TextStat* and *DataStat* variable, the programmer can determine the state or status of the program at a particular instance. The following chart describes the state of the program based upon these variables:

TextStat	DataStat	Description
Intro		
30	0	Intro w/o highlighted words
30	1	Intro w/ highlighted words (Not implemented)
30	6	Creating an intro file
30	7	Modifying an intro file
Highlighted Words		
Normal		
10	0	Text w/o highlighted words
10	1	Text w/ highlighted words
Highlights		
11	3	Creating a new highlight - MarkON
11	4	Creating a new highlight - MarkOFF
Definitions		
12	5	Editing a new definition
Question and Answers		
21	1	Displaying question (no highlights)
21	5	Editing a new question
22	1	Displaying Text Answer
22	3	Creating Text Answer - MarkON
22	4	Creating Text Answer - MarkOFF
22	9	Creating Text Answer - Starting Condition

23	1	Displaying Multiple Choice Answer
23	3	Creating Multiple Choice Answers - MarkON
23	4	Creating Multiple Choice Answers - MarkOFF
23	9	Creating Multiple Choice Answers - Starting Condition
24	1	Displaying Text Response
24	5	Editing Text Response

As can be seen by the variable values, I attempted to group either the function type or the area type together. In addition, the variable structure is such that additional functions may be later added and some which I can envision being added have also been indicated.

The information on the screen is tracked with a *StatusType* record which contains most of the information required. This record stores the array locations of the first and last character on the screen as well as the present character being pointed at by the cursor. It stores the first and last pointer to the display list of highlighted words and stores the current highlighted word on the page and the first and last highlighted words on the page. When the questions and answers are being displayed, this variable will hold a pointer to the current question.

The final required variable structure is a screen matrix array. To create the final template, a character and its corresponding attributes are assigned to a specific location on the screen and stored within a temporary screen matrix. When all screen locations have their values stored in them, the temporary screen matrix is switched with the page presently being displayed and becomes the present page. This decreases flickering of text which may occur with constant writing of information on the screen.

Article Variables

The variable structure of the article is a character array (*TextBuffer*) with a maximum of 3000 characters in the array at any time. If the article has more than 3000 characters of text, pages are swapped from the article file. Swapping occurs when the first character's array value on the page is less than 1000 when traveling toward the beginning of the article or when the last character's array value on the page is greater than

2000 when traveling toward the end of the file. Since a number of swappings may occur while reading the article, the programs use a double buffering system to decrease the time required to swap in the appropriate page.

Presently, the program is allowed to have 300 lines of text. To keep track of the total layout of the article, an array (*TextPage*) contains the array value of the first character in each line. This array would have recorded the actual array value in the character array as if the complete article were saved in resident memory. In addition, a *Page* variable indicates which section of the 3000 characters the article is presently displaying. The array and *Page* are referenced during paging and swapping pages to determine which characters will be displayed.

Highlighting Variables

Three variables (*HelpNodeType*, *StringBuffer*, and *TextNodeType*) are required to create and display the highlights.

A *HelpNodeType* is created for each sequential set of characters within a hot word object. The *HelpNodeType* contains the first and last position in the article text array of the highlighted word and the first position in the *StringBuffer* array of the definition. If multiple *HelpNodeTypes* are required for one hot word object, the different parts of the hot word are doubly linked together. For traveling from one *HelpNodeType* to another, each *HelpNodeType* is doubly linked to a list of all the *HelpNodeTypes* for that level. So that all possible highlighted areas are displayed on the screen, each *HelpNodeType* is doubly linked sequentially to other *HelpNodeTypes* based upon the first position of the highlight in the article text.

A *StringBuffer* is a character buffer which contains the text for all the definitions. Since a character buffer requires less overhead to maintain, the text for all the definitions are stored within this array and the *HelpNodeType* determines which definition to display based upon the array position of the first character of the definition. The *StringBuffer*'s

maximum number of characters is 1000. This maximum only effects the definition characters on one level.

The *TextNodeType* is implemented when a series of highlights and definitions are imported into the program. This *TextNodeType*, which contains the highlighted word and definition strings, is singly linked. When the highlighted word and definition are displayed, the *TextNodeType* is deleted.

As with the article text, the current hot word and the highlightable text have default color constants associated with them. The following is a list of these color constant defaults:

Constant Name	Color
SelectableBackground	Blue
SelectableText	LightRed
SelectedBackground	Magenta
SelectedText	White

Question Variables

Up to two variables can be used for question and answers. A character array similar to the highlights character array is used to store the question and the answers. All the questions for a level are stored within a doubly linked list composed of *QuestionTypes*. These *QuestionTypes* hold the beginning position of the question and the beginning position of the answer. The program knows to stop entering characters when the "|" character is encountered for the questions. For the answers, the ">" character is used to determine the end of the answer.

The question have default color constants associated with them. These defaults involve those answers which are displayed in the definition area of the screen. Any answer displayed in the text area of the screen uses the defaults of the Current Hot Word. The following is a list of these color constant defaults:

Constant Name	Color
AnswerBackground	LightGray
AnswerText	Red

Level Variables

The programs references three structures to store the file names of the article, introduction, and levels. A *FilesRec* record is the structure which contains the names of the files being referenced (the introduction file, the article file, a pointer to the first level file name, and a pointer to the current level file name). For the introduction and the article files, a *FilesPtr* record stores the name of the file. A header is stored in the *FilesPtr* record for an introduction file. The third structure, the *LevelRec* holds the file name(s) and other information required for each level. This structure will be examined in a later section.

Program and Module Listing

ABRA.PAS converts calls the functions in the other modules based upon the keypress. It displays the highlights and questions created by the authoring system. No editing is involved with this program.

ABRA-AS.PAS creates the highlights and questions for display by ABRA. Based upon keypresses, the program calls the required functions.

ABRAVARS.PAS contains all the global variables which will be used by all the modules.

ABRATEXT.PAS contains the text functions. These functions control only the text and not the highlights which are overlayed upon the text. These functions include movement functions of the cursor.

ABRAHW.PAS contains the highlighting functions. These functions control overlaying highlights upon the text. These functions include creating and editing the highlights.

ABRAQUES.PAS contains the question functions. These functions control the question and answer which are displayed in the definition area. These functions include creating and editing the question and answers.

ABRAINTR.PAS contains the introduction functions. These functions control the creation and display of the introduction prior to displaying the article. The main functions include editing and displaying the introduction.

ABRALEV.PAS contains the level functions. These functions control the files for the text, introduction, highlights, and questions. It performs the required functions to traverse between the various levels.

ABRAED.PAS contains the editor functions. These functions are used whenever the introduction, a definition, a question, or an answer is created or edited. Though I would like to take credit for writing the editor, I used code adapted from O'Brien and Nameroff's TurboPascal 7: the Complete Reference.

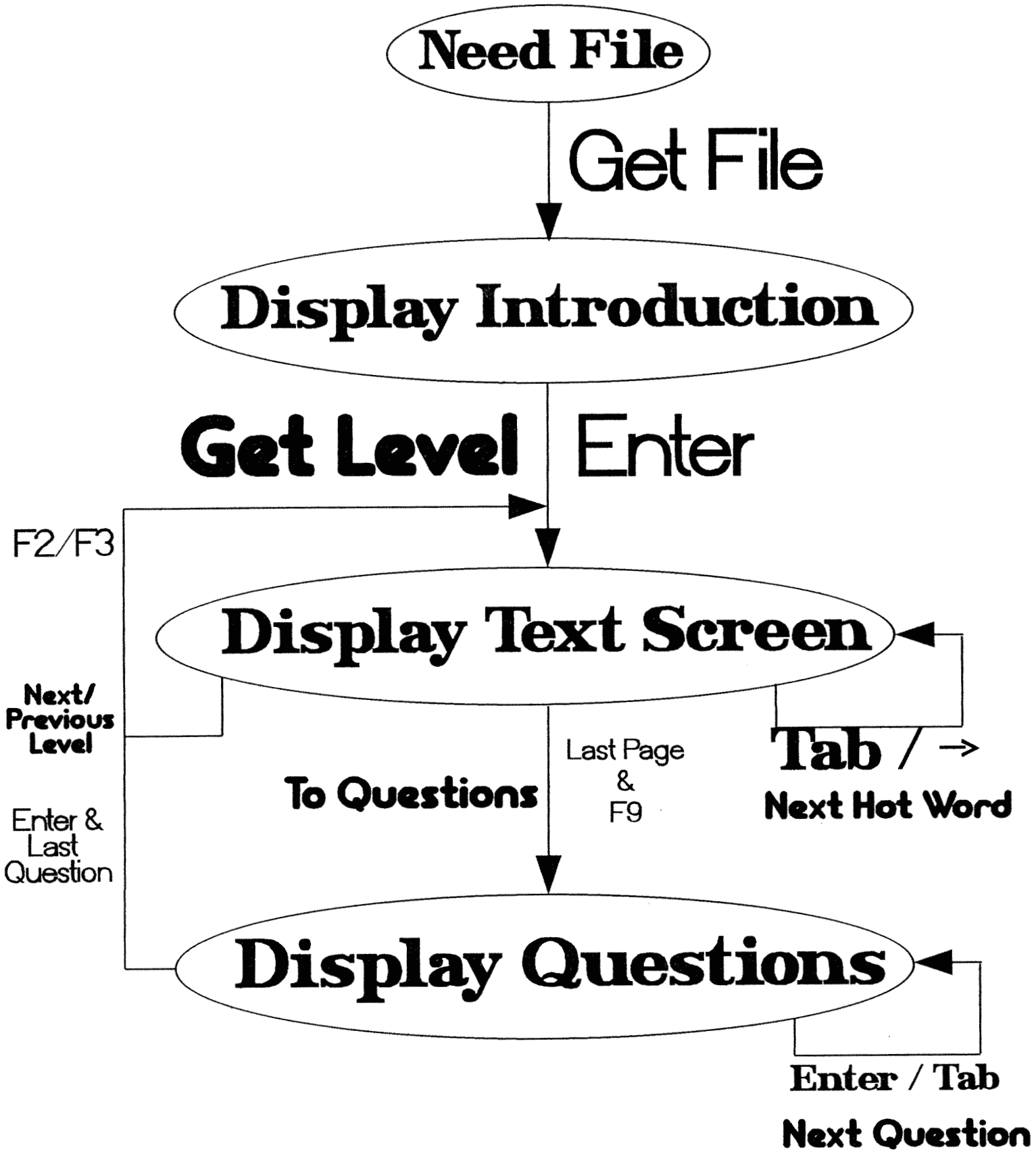
ABRABACK.PAS contains all the functions which are performed in the background. These functions include the displaying of a screen when all pieces are available. Other functions consist of functions which may be used by multiple modules and require and common calling module.

TP_REF.PAS contains the other functions which were copied from O'Brien and Nameroff's TurboPascal 7: the Complete Reference. This book was invaluable for cursor control, as well as displaying information upon the screen.

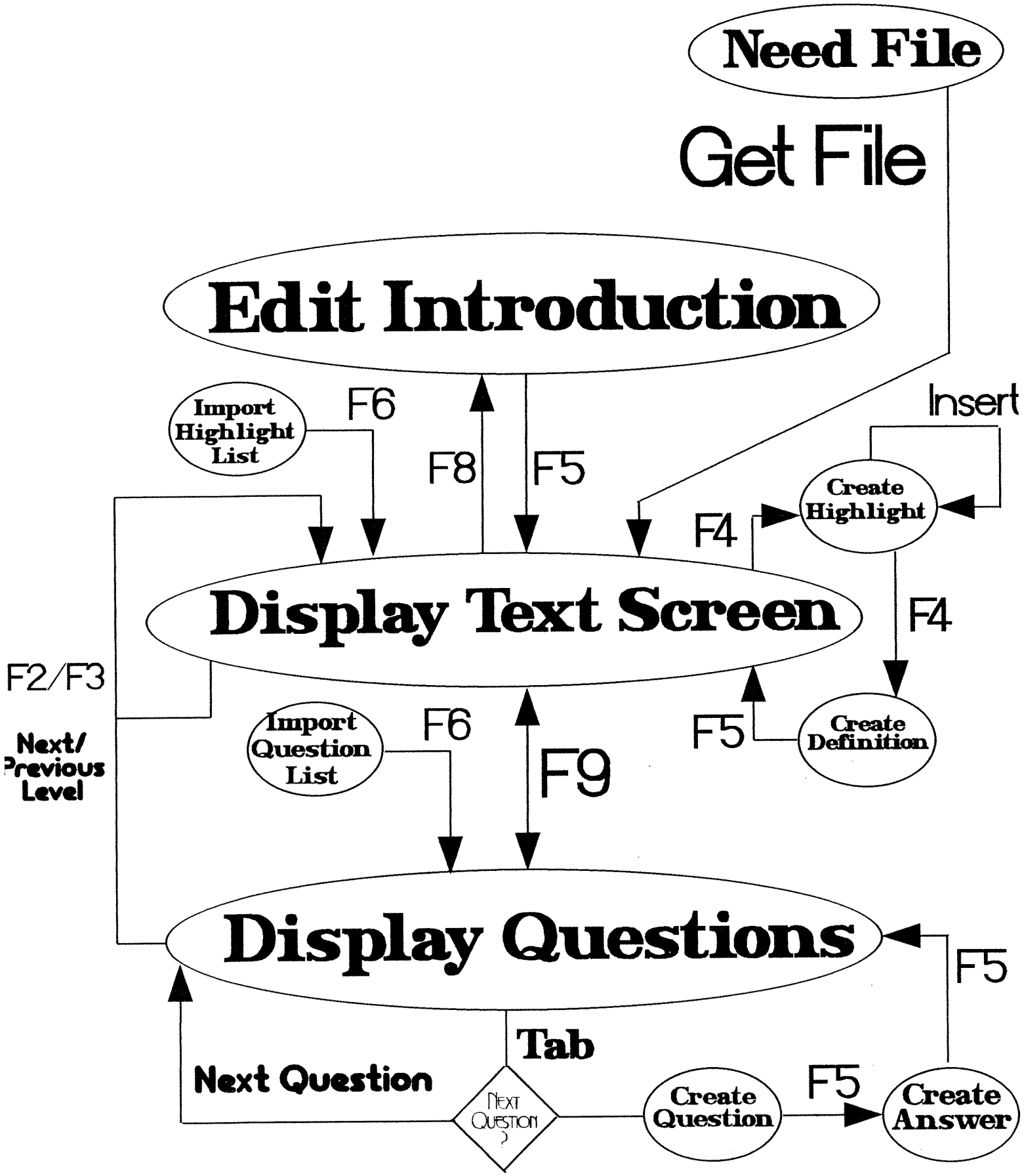
Data Flow Diagrams

On the following two pages, I have created a state diagram for ABRA and for ABRA-AS. I contains most of the functions and discribes the flow of the programs.

ABRA State Diagram



ABRA-AS State Diagram



Appendix C

Source Code

Appendix C01

ABRA.PAS

```
{*************************************************************************}
{ PROGRAM NAME:      Advanced Bilingual Reading Assistant                 }
{ PROGRAMMER:        Raymundo A. Q. Marcelo                               }
{ DATE:              23-April-1995                                        }
{                                                                         }
{ SPONSORS:          German, Russian, and East Asian Languages Department }
{                    Miami University                                     }
{                    Oxford, OH  45                                       }
{                                                                         }
{ PURPOSE:           This program is the engine which creates the various }
{                    highlighted keywords, definitions, and question and  }
{                    answers for the Advanced Bilingual Reading Assistant.}
{                                                                         }
{ LANGUAGE:          Borland's Turbo Pascal 7.0                           }
{*************************************************************************}
{                                                                         }
{                    Function State and Variable Values                   }
{                                                                         }
{TextStat      DataStat          Description

Intro
30               0               Intro w/o highlighted words
30               1               Intro w/ highlighted words
30               6               Creating an intro file
30               7               Modifying an intro file

Highlighted Words
        Normal
10               0               Text w/o highlighted words
10               1               Text w/ highlighted words

        Highlights
11               3               Creating a new highlight - MarkON
11               4               Creating a new highlight - MarkOFF
{
{        Definitions
{12              5                Creating a new definition (Editting)
{
{Question and Answers
{21              1                Displaying question  (no HL)
{
{22              1                Displaying Text Answer
{22              3                Creating Text Answer - MarkON
{22              4                Creating Text Answer - MarkOFF

{23              1                Displaying Multiple Choice Answer
{23              3                Creating Multiple Choice Answers - MarkON
{23              4                Creating Multiple Choice Answers - MarkOFF

{24              1                Displaying Text Response
{24              5                Creating Text Response}

{$V-}
uses
  Dos, Crt, TP_REF, ABRAVARS, ABRABACK, ABRAED, ABRATEXT,
  ABRAINTR, ABRALEV, ABRAHW, ABRAQUES;

const
   {Window Dimensions}
   LtMarginConst = 4;
   ScrLineLenConst = 72;
   NormTextLinesConst = 18;
   InterlinearLinesConst = 15;

   {Text Colors}
   NormalBackgroundConst = Blue;
```

```
   NormalTextConst = Yellow;
   SelectableBackgroundConst = Blue;
   SelectableTextConst = LightRed;
   SelectedBackgroundConst = Magenta;
   SelectedTextConst = White xor 128;

   {Help Colors}
   HelpBackgroundConst = Red;
   HelpTextConst = White;
   AnswerBackgroundConst = LightGray;
   AnswerTextConst = Red;

   {Status Line Colors}
   StatusBackgroundConst = Magenta;
   StatusTextConst = LightCyan;
   SelectedStatusTextConst = Yellow;

var
   { Program variables }
   command          :  KeyType;
   ch               :  char;
   SpecKey,
   Done             :  boolean;
   TString          :  string;
   i                :  integer;

{  InitConst loads the colors and other constants which were previously
   defined.  This allows the constants to be changed within ABRA and ABRA-AS
   without requiring all units to also be recompiled.                    }

procedure InitConst;

   begin
      LtMargin := LtMarginConst;
      ScrLineLen := ScrLineLenConst;
      NormTextLines := NormTextLinesConst;
      InterlinearLines := InterlinearLinesConst;

      NormalBackground := NormalBackgroundConst;
      NormalText := NormalTextConst;
      SelectableBackground := SelectableBackgroundConst;
      SelectableText := SelectableTextConst;
      SelectedBackground := SelectedBackgroundConst;
      SelectedText := SelectedTextConst;

      HelpBackground := HelpBackgroundConst;
      HelpText := HelpTextConst;
      AnswerBackground := AnswerBackgroundConst;
      AnswerText := AnswerTextConst;

      StatusBackground := StatusBackgroundConst;
      StatusText := StatusTextConst;
      SelectedStatusText := SelectedStatusTextConst;
   end;

begin
   InitConst;

   Cursor_Off;
   WithCursor := FALSE;
   ScrTextLines := NormTextLines;

   clrscr;
   LoadFile;
   if (TextStat < 40) then
      begin
```

```
   InitIntro;
   BottomBackground;
   FirstPage;
   DisplayPage;
   StatusLine;
   end;

while (TextStat < 40) do
   begin
   ReDisplayPage := FALSE;

   InKey(SpecKey, command, ch);

   if (command = TextKey) then
      case ch of
         '+' :  command := F3;
         '-' :  command := F2;
         end;

   case command of
      {Scrolling Commands}
      UpArrow          :  ScrollUp;
      DownArrow        :  ScrollDown;
      LeftArrow        :  if (TextStat < 20) then
                             PrevHotWord
                          else
                             PrevQuestion;
      RightArrow       :  if (TextStat < 20) then
                             NextHotWord
                          else
                             if Stats.CurrentQuestion^.NextQuestion = nil then
                                NextHelpLevel
                             else
                                NextQuestion;
      PgUp             :  PageUp;
      PgDn             :  PageDown;
      HomeKey          :  FirstPage;
      EndKey           :  LastPage;

      { Help }
      F1   :  begin
              case TextStat of
                 10, 11, 12 :  KeyboardHelpHW;
                 21, 22, 23,24 :  KeyboardHelpQ;
              end;
              ReDisplayPage := True;
              end;
      {  Change Levels  }
      F2   :  if TextStat < 30 then
                 PrevHelpLevel;
      F3   :  if TextStat < 30 then
                 if Files.CurrLevel^.NextLev <> nil then
                    NextHelpLevel
                 else
                    begin
                    QuitProgram;
                    Cursor_Small;
                    NormVideo;
                    clrscr;
                    Halt;
                    end;
      F9   :  {Enter Question and Answer Section}
              if ((TextStat < 20) and (ArticleEnd)) then
                 begin
                 ScrTextLines := InterlinearLines;
                 if Stats.LastLine - Stats.FirstLine <> ScrTextLines - 2 then
```

```
                AdjustScrTextLines;
              BottomBackground;
              FirstQuestion;
              end;
   F10 :   begin
           TextStat := 40;
           end;

   {Other Keys}
   ShiftTab  :  if (TextStat < 20) then
                   PrevHotWord
                else
                   PrevQuestion;
   Tab       :  if (TextStat < 20) then
                   NextHotWord
                else
                   if Stats.CurrentQuestion^.NextQuestion <> nil then
                      NextQuestion
                   else
                      if Files.CurrLevel^.NextLev <> nil then
                         NextHelpLevel
                      else
                         TextStat := 40;
   CarriageReturn :  if (TextStat > 19) and (TextStat < 30) then
                        begin
                        if TextStat = 21 then
                           DisplayAnswer
                        else
                           if Stats.CurrentQuestion^.NextQuestion <> nil then
                              NextQuestion
                           else
                             if Files.CurrLevel^.NextLev <> nil then
                                NextHelpLevel
                             else
                                TextStat := 40;
                        end
                     else if (TextStat > 29) and (TextStat < 40) then
                        begin
                        Page := 0;
                        InitText;
                        Level := 1;
                        GetLevel;
                        BottomBackground;
                        if (Files.CurrLevel <> nil) then
                           begin
                           GetLevel;
                           InitLevel;
                           end;
                        TextStat := 10;
                        if H.HelpNodeList <> nil then
                           DataStat := 1
                        else
                           DataStat := 0;
                        FirstPage;
                        ReDisplayPage := TRUE;
                        end;
   end;
if (TextStat = 40) then
   begin
   Cursor_Small;
   NormVideo;
   clrscr;
   Halt;
   end;
if (Stats.LastLine - Stats.FirstLine <> ScrTextLines - 2) and
   (Stats.LastPos < TotalChar) then
```

```
         AdjustScrTextLines;
      UpdateScrMatrix;
      if (DataStat < 3) and (Stats.LastHotWord <> nil) and
         (TextStat < 20) and (ScreenHelp[Stats.LocX, Stats.LocY] <> 0) then
         begin
         TestNode := Stats.FirstHotWord;
         while TestNode^.Index <> ScreenHelp[Stats.LocX, Stats.LocY] do
            TestNode := TestNode^.NextNode;
         Stats.CurrentHotWord := TestNode;
         RedisplayPage := True;
         end;

      if (TextStat < 40) and (ReDisplayPage) then
         DisplayPage;
      if (TextStat < 20) and (not BeginNewLevel) then
         if (DataStat < 3) and (ScreenHelp[Stats.LocX, Stats.LocY] <> 0) then
            DisplayDefinition
         else
            BottomBackground;
      StatusLine;
      GoToXY(Stats.LocX, Stats.LocY);
      end;
end.
```

Appendix C02

ABRA-AS.PAS

```
{************************************************************************}
{ PROGRAM NAME:      Advanced Bilingual Reading Assistant - Authoring System }
{ PROGRAMMER:        Raymundo A. Q. Marcelo                                  }
{ DATE:              23-April-1995                                           }
{                                                                            }
{ SPONSORS:          German, Russian, and East Asian Languages Department    }
{                    Miami University                                        }
{                    Oxford, OH  45                                          }
{                                                                            }
{ PURPOSE:           This program is the engine which creates the various    }
{                    highlighted keywords, definitions, and question and     }
{                    answers for the Advanced Bilingual Reading Assistant.   }
{                                                                            }
{ LANGUAGE:          Borland's Turbo Pascal 7.0                              }
{************************************************************************}
{                                                                            }
{                    Function State and Variable Values                      }
{                                                                            }
{TextStat      DataStat         Description

Intro
30               0              Intro w/o highlighted words
30               1              Intro w/ highlighted words
30               6              Creating an intro file
30               7              Modifying an intro file

Highlighted Words
        Normal
10               0              Text w/o highlighted words
10               1              Text w/ highlighted words

        Highlights
11               3              Creating a new highlight - MarkON
11               4              Creating a new highlight - MarkOFF
{
{        Definitions
{12              5               Creating a new definition (Editting)
{
{Question and Answers
{21              1               Displaying question  (no HL)
{
{22              1               Displaying Text Answer
{22              3               Creating Text Answer - MarkON
{22              4               Creating Text Answer - MarkOFF

{23              1               Displaying Multiple Choice Answer
{23              3               Creating Multiple Choice Answers - MarkON
{23              4               Creating Multiple Choice Answers - MarkOFF

{24              1               Displaying Text Response
{24              5               Creating Text Response}

{$V-}
uses
  Dos, Crt, TP_REF, ABRAVARS, ABRABACK, ABRAED, ABRATEXT,
  ABRAINTR, ABRALEV, ABRAHW, ABRAQUES;

const
   {Window Dimensions}
   LtMarginConst = 4;
   ScrLineLenConst = 72;
   NormTextLinesConst = 18;
   InterlinearLinesConst = 15;

   {Text Colors}
   NormalBackgroundConst = Blue;
```

```
   NormalTextConst = Yellow;
   SelectableBackgroundConst = Blue;
   SelectableTextConst = LightRed;
   SelectedBackgroundConst = Magenta;
   SelectedTextConst = White xor 128;

   {Help Colors}
   HelpBackgroundConst = Red;
   HelpTextConst = White;
   AnswerBackgroundConst = LightGray;
   AnswerTextConst = Red;

   {Status Line Colors}
   StatusBackgroundConst = Magenta;
   StatusTextConst = LightCyan;
   SelectedStatusTextConst = Yellow;

var
   { Program variables }
   command          :  KeyType;
   ch               :  char;
   SpecKey,
   Done             :  boolean;
   TString          :  string;
   i                :  integer;

procedure InitConst;

   begin
      LtMargin := LtMarginConst;
      ScrLineLen := ScrLineLenConst;
      NormTextLines := NormTextLinesConst;
      InterlinearLines := InterlinearLinesConst;

      NormalBackground := NormalBackgroundConst;
      NormalText := NormalTextConst;
      SelectableBackground := SelectableBackgroundConst;
      SelectableText := SelectableTextConst;
      SelectedBackground := SelectedBackgroundConst;
      SelectedText := SelectedTextConst;

      HelpBackground := HelpBackgroundConst;
      HelpText := HelpTextConst;
      AnswerBackground := AnswerBackgroundConst;
      AnswerText := AnswerTextConst;

      StatusBackground := StatusBackgroundConst;
      StatusText := StatusTextConst;
      SelectedStatusText := SelectedStatusTextConst;
   end;

begin
   InitConst;

   Modified := FALSE;
   WithCursor := TRUE;
   ScrTextLines := NormTextLines;

   clrscr;
   LoadFile;

   if (TextStat < 40) then
      begin
      InitText;
      BottomBackground;
      if (DataStat > -1) then
```

```
      GetLevel
   else
      begin
      BottomBackground;
      TString := 'There are presently no highlighted words present.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 1, TString[i],
                   HelpText, HelpBackground);
      TString := 'Please scroll to the word(s) to be highlighted.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 2, TString[i],
                   HelpText, HelpBackground);
      TString := 'Mark and Unmark the word(s) using the F4 key.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 3, TString[i],
                   HelpText, HelpBackground);
      TString := 'Type in the definition within this lower block.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 4, TString[i],
                   HelpText, HelpBackground);
      end;
   Level := 1;
   InitLevel;
   FirstPage;
   DisplayPage;
   StatusLine;
   GoToXY(Stats.LocX, Stats.LocY);
   end;

while (TextStat < 40) do
   begin
   ReDisplayPage := FALSE;

   InKey(SpecKey, command, ch);

   if (command = TextKey) then
      case ch of
         '+' :  command := F3;
         '-' :  command := F2;
         end;

   case command of
      {Scrolling Commands}
      UpArrow        :  ScrollUp_AS;
      DownArrow      :  ScrollDown_AS;
      LeftArrow      :  LastChar;
      RightArrow     :  NextChar;
      PgUp           :  PageUp;
      PgDn           :  PageDown;
      HomeKey        :  FirstPage;
      EndKey         :  LastPage;

      { Help }
      F1   :  begin
              case TextStat of
                 10, 11, 12 :  KeyboardHelpHW;
                 21, 22, 23,24 :  KeyboardHelpQ;
              end;
              ReDisplayPage := True;
              end;
      {  Change Levels  }
      F2,
      F3   :  begin
              if ModifiedLevel or ModifiedQuestion then
                 begin
                 BottomBackground;
```

```
            TextColor(HelpText);
            TextBackground(HelpBackground);
            GoToXY(LtMargin, ScrTextLines + 1);
            write('You have made changes to this level which were not saved!!');
            GoToXY(LtMargin, ScrTextLines + 2);
            write('Do you want to SAVE your work?? (Y/N) ');
            readln(ch);
            if UpCase(ch) = 'Y' then
               begin
               if ModifiedLevel then
                  SaveLevel;
               if ModifiedQuestion then
                  SaveLevelQuestions;
               end;
            end;
         if (command = F2) then
            PrevHelpLevel
         else
            NextHelpLevel;
         end;
{  Beginning and Ending Highlighting  }
F4   :   if (DataStat <> 3) then
            begin
            DataStat := 3;
            if (TextStat = 10) then
               begin
               TextStat := 11;
               BeginNewHotWord(TmpNewNode);
               end
            else if (TextStat = 22) then
               BeginNewTextAnswer;
            end
         else
            begin
            {GoBack to Normal text Output}
            if (TextStat = 11) then
               begin
               EndNewHotWord(TmpNewNode);
               TextStat := 10;
               DataStat := 1;
               end;
            end;
F5   :   {Edit Definition}
         begin
         if (TextStat < 20) then
            EditHWDefn
         else
            EditQuestion;
         end;
F6   :   {Get Files}
         begin
         if (TextStat < 20) then
            NewHotWordFile
         else
            NewLevelQuestions;
         end;
F7   :   { Save Files }
         begin
         if (TextStat < 20) then
            SaveLevel
         else
            SaveLevelQuestions;
         end;
F8   :   {Edit Intro}
         begin
         EditIntro;
```

```
        end;
F9   :  {Enter Question and Answer Section}
        if (TextStat > 19) and (TextStat < 30) then
           begin
           if Files.CurrLevel^.Interlinear then
              ScrTextLines := InterlinearLines
           else
              ScrTextLines := NormTextLines;
           if Stats.LastLine - Stats.FirstLine <> ScrTextLines - 2 then
              AdjustScrTextLines;
           BottomBackground;
           end
        else
           begin
           ScrTextLines := InterlinearLines;
           if Stats.LastLine - Stats.FirstLine <> ScrTextLines - 2 then
              AdjustScrTextLines;
           BottomBackground;
           FirstQuestion;
           end;
F10  :  begin
        if ModifiedLevel or ModifiedQuestion then
           begin
           BottomBackground;
           TextColor(HelpText);
           TextBackground(HelpBackground);
           GoToXY(LtMargin, ScrTextLines + 1);
           write('You have made changes to this level which were not saved!!');
           GoToXY(LtMargin, ScrTextLines + 2);
           write('Do you want to SAVE your work?? (Y/N) ');
           readln(ch);
           if UpCase(ch) = 'Y' then
              begin
              if ModifiedLevel then
                 SaveLevel;
              if ModifiedQuestion then
                 SaveLevelQuestions;
              end;
           end;
        QuitProgram;
        NormVideo;
        clrscr;
        Halt;
        end;

{Other Keys}
InsertKey : if (TextStat = 11) then
               if (DataStat = 3) then
                  begin
                  TmpNewNode^.EndRange := Stats.CurrentPos - 1;
                  DataStat := 4;
                  end
               else
                  begin
                  BeginNewHotWord(TmpNewNode);
                  DataStat := 3;
                  end;
ShiftTab  :  if (TextStat < 20) then
                PrevHotWord
             else
                PrevQuestion;
Tab       :  if (TextStat < 20) then
                NextHotWord
             else
                NextQuestion;
DeleteKey :  if (TextStat < 20) then
```

```
                        DeleteHotWord
                     else
                        DeleteQuestion;
      CarriageReturn :  if (TextStat > 19) and (TextStat < 30) then
                           begin
                           if TextStat = 21 then
                              DisplayAnswer
                           else
                              NextQuestion;
                           end;
      Esc :  begin
             if (DataStat = 3) or (DataStat = 4) then
                begin
                if (TextStat > 19) and (TextStat < 30) then
                   begin
                   TextStat := 21;
                   DataStat := 0;
                   end
                else
                   begin
                   TextStat := 10;
                   if H.HelpNodeList <> nil then
                      DataStat := 1
                   else
                      DataStat := 0;
                   end;
                while TmpNode <> nil do
                   begin
                   TmpNewNode := TmpNode;
                   TmpNode := TmpNode^.NextPartner;
                   Dispose(TmpNewNode);
                   end;
                ReDisplayPage := TRUE;
                end;
             end;
      end;
   if (Stats.LastLine - Stats.FirstLine <> ScrTextLines - 2) and
      (Stats.LastPos < TotalChar) then
      AdjustScrTextLines;
   UpdateScrMatrix;
   if (DataStat < 3) and (Stats.LastHotWord <> nil) and
      (TextStat < 20) and (ScreenHelp[Stats.LocX, Stats.LocY] <> 0) then
      begin
      TestNode := Stats.FirstHotWord;
      while TestNode^.Index <> ScreenHelp[Stats.LocX, Stats.LocY] do
         TestNode := TestNode^.NextNode;
      Stats.CurrentHotWord := TestNode;
      RedisplayPage := True;
      end;

   if (TextStat < 40) and (ReDisplayPage) then
      DisplayPage;
   if (TextStat < 20) and (not BeginNewLevel) then
      if (DataStat < 3) and (ScreenHelp[Stats.LocX, Stats.LocY] <> 0) then
         DisplayDefinition
      else if IntegratingFile then
         begin
         end
      else
         BottomBackground;
   StatusLine;
   GoToXY(Stats.LocX, Stats.LocY);
   end;
end.
```

Appendix C03

ABRAVARS.PAS

```
{************************************************************************}
{ PARENT PROGRAM:    Advanced Bilingual Reading Assistant                }
{                    Advanced Bilingual Reading Assistant - Authoring System }
{ PROGRAMMER:        Raymundo A. Q. Marcelo                              }
{ DATE:              23-April-1995                                       }
{                                                                        }
{ UNIT NAME:         ABRAVARS                                            }
{                                                                        }
{ PURPOSE:           This unit contains the required variables used by both  }
{                    ABRA and ABRA-AS programs.  It also contains the    }
{                    constants and variable structures which are referred to }
{                    within all the units.                               }
{************************************************************************}

unit ABRAVARS;

interface

uses Dos, Crt;

const
   MaxTextSize      = 3000;  {Max Characters in Text}
   MaxPageSize      = 2000;
   MaxLines         = 4000;
   MaxStringBufSize = 2000;  {Max Characters in Help Levels}

   {Full Screen Dimensions}
   MaxTextLines     = 25;
   MaxLineLen       = 80;

   BufSize = 16384;

type
   FilesPtr = ^FilesType;
   FilesType = record
      Fname     :  string;
      Lname     :  string;
      NextFile  :  FilesPtr;
      end;

   LevelPtr = ^LevelRec;
   LevelRec = record
      LevName,
      QuesName,
      HWName       :  string;
      Interlinear  :  boolean;
      PrevLev,
      NextLev      :  LevelPtr;
      end;

   FilesRec = record
      Intro,
      Article      :  FilesPtr;
      CurrLevel,
      Levels       :  LevelPtr;
      end;

   EditString = ^EditStringRec;
   EditStringRec = record
      x, y,
      index  :  integer;
      str    :  string;
      prev,
      next   :  EditString;
      end;
```

```
   StringBufferType = array[1..MaxStringBufSize] of char;

   HelpNodePtr = ^HelpNodeType;
   HelpNodeType = record
      BeginRange,
      EndRange       :  LongInt;
      PrevPartner,
      NextPartner,
      PrevNode,
      NextNode,
      PrevDisplay,
      NextDisplay    :  HelpNodePtr;
      Index          :  1..MaxStringBufSize;
      end;

   TextNodePtr = ^TextNodeType;
   TextNodeType = record
      HotWord,
      Defn       :  string;
      Next       :  TextNodePtr;
      end;

   QuestionPtr = ^QuestionType;
   QuestionType = record
      QuestionStart,
      AnswerStart    :  integer;
      PrevQuestion,
      NextQuestion   :  QuestionPtr;
      end;

   LevelType = record
      LevelName      :  string;
      HelpNodeList,
      DisplayList    :  HelpNodePtr;
      Questions      :  QuestionPtr;
      BufferEnd      :  Longint;
      QBEnd          :  integer;
      end;

   ScreenHelpArray = array[1..MaxLineLen, 1..MaxTextLines] of integer;

   StatusType = record
      FirstPos,
      CurrentPos,
      LastPos,
      FirstLine,
      CurrentLine,
      LastLine,
      LocX,
      LocY               :  LongInt;
      CurrentHotWord,
      FirstHotWord,
      LastHotWord,
      FirstDisplay,
      LastDisplay        :  HelpNodePtr;
      CurrentQuestion    :  QuestionPtr;
      end;

var
   {   Variables Referencing Colors on the Screen }
   NormalBackground,
   NormalText,
   SelectableBackground,
   SelectableText,
   SelectedBackground,
   SelectedText,
```

```
HelpBackground,
HelpText,
AnswerBackground,
AnswerText,
StatusBackground,
StatusText,
SelectedStatusText : byte;

{   Variables Referencing the Number of Lines for the Text}
NormTextLines,
InterlinearLines    :  integer;

Files               :  FilesRec;
IntroFile,
TextFile,
LevelFile,
QuestionFile,
FileFile            :  TEXT;
D                   :  DirStr;
N                   :  NameStr;

H                   :  LevelType;

{ Characteristics of the Text }
TextBuffer          :  Array[1..MaxTextSize] of char;
TextPage            :  array[1..MaxLines] of LongInt;
Page                :  integer;
TotalChar           :  LongInt;
TotalLines          :  1..MaxLines;
Buf                 :  array[1..BufSize] of byte;

StringBuffer        :  StringBufferType;

{ Characteristics of the Screen }
ScreenHelp          :  ScreenHelpArray;
Stats               :  StatusType;
LtMargin,
ScrLineLen          :  integer;
ScrTextLines        :  integer;
ReDisplayPage       :  boolean;

{  Status Variables }
TextStat            :  integer;
DataStat            :  integer;

{  Modification Variables   }
ArticleEnd,
WithCursor,
IntegratingQFile,
IntegratingFile,
BeginNewLevel       :  boolean;

TestNode,
TmpNode,
TmpNewNode,
TmpCurrent          :  HelpNodePtr;

HeadNewHW           :  TextNodePtr;

Level               :  integer;

Modified,
ModifiedIntro,
ModifiedLevel,
ModifiedQuestion    :   boolean;
```

```
implementation

end.
```

Appendix C04

ABRATEXT.PAS

```
{*************************************************************************}
{ PARENT PROGRAM:   Advanced Bilingual Reading Assistant                   }
{                   Advanced Bilingual Reading Assistant - Authoring System }
{ PROGRAMMER:       Raymundo A. Q. Marcelo                                 }
{ DATE:             23-April-1995                                          }
{                                                                          }
{ UNIT NAME:        ABRATEXT                                               }
{                                                                          }
{ PURPOSE:          This unit contains the functions and procedures required}
{                   by the ABRA and ABRA-AS programs to display information }
{                   from those programs onto the screen.                   }
{*************************************************************************}

unit ABRATEXT;

interface

uses Dos, Crt, TP_REF, ABRAVARS, ABRABACK, ABRAED;

procedure InitText;
procedure GetText;
procedure FindPage(var HotWord    :  HelpNodePtr);

procedure FirstPage;
procedure LastPage;

procedure ScrollUp;
procedure ScrollUp_AS;
procedure ScrollDown;
procedure ScrollDown_AS;
procedure LastChar;
procedure NextChar;
procedure PageUp;
procedure PageDown;

implementation

{   procedure InitText initializes the variables for displaying the article
    In addition, it sets the text at page 0                                }
procedure InitText;


   var
      ALine,
      tempstr    :   string;
      i, pi, j   :   Longint;
      TooBig     :   boolean;

   begin
   Assign(TextFile, Files.Article^.Fname);
   SetTextBuf(TextFile, Buf);
   Reset(TextFile);
   pi := 1;
   i := 1;

   TooBig := FALSE;
   WHILE ((not EOF(TextFile)) or (TooBig)) DO
      BEGIN
      {Place text in the TextBuffer}
      Readln(TextFile, ALine);
      TextPage[pi] := i;
      for j := 1 to Length(ALine) do
         begin
         if ((i >= Page * MaxPageSize) and
             (i <= (Page * MaxPageSize) + MaxTextSize)) then
            TextBuffer[i] := ALine[j];
```

```
        i := i + 1;
        end;
     if ALine[Length(ALine)] <> ' ' then
        begin
        if ((i >= Page * MaxPageSize) and
            (i <= (Page * MaxPageSize) + MaxTextSize)) then
           TextBuffer[i] := ' ';
        i := i + 1;
        end;
     pi := pi + 1;
     end;

  Close(TextFile);
  TextPage[pi] := i;
  TotalChar := i - 1;
  TotalLines := pi - 1;
  END;

{   procedure GetText reads the article file and populates the article
    buffer with the text from the page to be read                        }
procedure GetText;

  var
     ALine,
     tempstr    :   string;
     i, j, pi   :   LongInt;
     BIndex     :   LongInt;
     GTDone     :   boolean;

  begin
  Reset(TextFile);
  pi := 1;
  i := 1;
  GTDone := FALSE;

  WHILE (not EOF(TextFile) and (not GTDone)) DO
     BEGIN
     {Place text in the TextBuffer}
     Readln(TextFile, ALine);
     for j := 1 to Length(ALine) do
        begin
        if ((pi >= Page * MaxPageSize + 1) and
            (pi <= (Page * MaxPageSize) + MaxTextSize)) then
           begin
           TextBuffer[i] := ALine[j];
           i := i + 1;
           end;
        pi := pi + 1;
        end;
     if ALine[Length(ALine)] <> ' ' then
        begin
        if ((pi >= Page * MaxPageSize + 1) and
            (pi <= (Page * MaxPageSize) + MaxTextSize)) then
           begin
           TextBuffer[i] := ' ';
           i := i + 1;
           end;
        pi := pi + 1;
        end;
     if (pi >= (Page * MaxPageSize) + MaxTextSize) then
        GTDone := TRUE;
     end;

  Close(TextFile);
  END;
```

```
{  Based upon the hot words, this procedure finds the page on which a
   particular hot word is present and positions the page so that the
   line on which the hot word appears is on the third line              }
procedure FindPage(var HotWord    :  HelpNodePtr);

   begin
   if (HotWord^.BeginRange < Stats.FirstPos) then
      begin
      while (HotWord^.BeginRange <= TextPage[Stats.CurrentLine]) do
         Stats.CurrentLine := Stats.CurrentLine - 1;
      if Page <> TRUNC(HotWord^.BeginRange/MaxPageSize) then
         begin
         Page := TRUNC(HotWord^.BeginRange/MaxPageSize);
         GetText;
         end;
      end
   else if (HotWord^.BeginRange > Stats.LastPos) then
      begin
      while (HotWord^.BeginRange >= TextPage[Stats.CurrentLine]) do
         Stats.CurrentLine := Stats.CurrentLine + 1;
      if Page <> TRUNC(HotWord^.BeginRange/MaxPageSize) then
         begin
         Page := TRUNC(HotWord^.BeginRange/MaxPageSize);
         GetText;
         end;
      end;
   if Stats.CurrentLine - 3 <= 0 then
      FirstPage
   else if Stats.CurrentLine + ScrTextLines - 5 > TotalLines then
      LastPage
   else
      begin
      Stats.FirstLine := Stats.CurrentLine - 3;
      Stats.FirstPos := TextPage[Stats.FirstLine];
      Stats.LastLine := Stats.FirstLine + ScrTextLines - 2;
      Stats.LastPos := TextPage[Stats.LastLine + 1] - 1;
      end;
   FindXYBeginRange(HotWord, Stats.CurrentLine, Stats.LocX, Stats.LocY);
   if (TextStat < 20) or (TextStat > 29) then
      GetHelpRange;
   end;


{  This procedure positions the text being read at the beginning of the text }
procedure FirstPage;

   var
      i  :  integer;

   begin
   if Page > 0 then
      begin
      Page := 0;
      GetText;
      end;
   Stats.FirstPos := TextPage[1];
   Stats.FirstLine := 1;

   if TotalLines > ScrTextLines - 1 then
      begin
      Stats.LastPos := TextPage[ScrTextLines] - 1;
      Stats.LastLine := ScrTextLines - 1;
      end
   else
      begin
      Stats.LastPos := TotalChar;
```

```
      Stats.LastLine := TotalLines;
      end;
   GetHelpRange;
   if Stats.CurrentHotWord <> nil then
      begin
      if Stats.CurrentHotWord^.BeginRange <= Stats.LastPos then
         begin
         FindXYBeginRange(Stats.CurrentHotWord, Stats.CurrentLine,
                          Stats.LocX, Stats.LocY);
         Stats.CurrentPos := Stats.CurrentHotWord^.BeginRange;
         end
      else
         Stats.CurrentPos := TextPage[Stats.LocY - 1] + Stats.LocX - LtMargin;
      end
   else
      begin
      Stats.LocX := LtMargin;
      Stats.LocY := 2;
      Stats.CurrentLine := 1;
      Stats.CurrentPos := TextPage[1];
      end;
   UpdateScrMatrix;
   ReDisplayPage := TRUE;
   end;

{  This procedure positions the text being read at the end of the text }
procedure LastPage;

   begin
   ArticleEnd := TRUE;
   Stats.LastPos := TotalChar;
   Stats.CurrentPos := TotalChar;
   Stats.LastLine := TotalLines;
   Stats.CurrentLine := TotalLines;
   if TotalLines + 3 > ScrTextLines then
      begin
      Stats.FirstLine := TotalLines + 3 - ScrTextLines;
      Stats.FirstPos := TextPage[Stats.FirstLine];
      Stats.LocY := ScrTextLines - 1;
      end
   else
      begin
      Stats.FirstPos := TextPage[1];
      Stats.FirstLine := 1;
      Stats.LocY := Stats.LastLine + 1;
      end;
   Stats.LocX := TotalChar - TextPage[Stats.LastLine] + LtMargin;
   GetHelpRange;
   if Page <> TRUNC(Stats.FirstPos/MaxPageSize) then
      begin
      Page := TRUNC(Stats.FirstPos/MaxPageSize);
      GetText;
      end;
   ReDisplayPage := TRUE;
   end;

{   Scrolling Procedures   }

procedure ScrollUp;

   var
      TestNode  :  HelpNodePtr;

   begin
   if Stats.CurrentLine > 1 then
      begin
```

```
      ReDisplayPage := TRUE;
      Stats.FirstLine := Stats.FirstLine - 1;
      Stats.FirstPos := TextPage[Stats.FirstLine];
      Stats.LastLine := Stats.LastLine - 1;
      Stats.LastPos := TextPage[Stats.LastLine + 1] - 1;
      Stats.CurrentLine := Stats.CurrentLine - 1;
      end;
   if ((Page > 0) and
      (TextPage[Stats.LastLine] < ((Page - 1) * MaxPageSize) +
                                                  MaxTextSize)) then
      begin
      Page := Page - 1;
      GetText;
      end;
   GetHelpRange;
   end;

procedure ScrollUp_AS;

   var
      TestNode   :  HelpNodePtr;

   begin
   if Stats.CurrentLine > Stats.FirstLine then
      begin
      Stats.LocY := Stats.LocY - 1;
      Stats.CurrentLine := Stats.CurrentLine - 1;
      end
   else
      begin
      if Stats.CurrentLine > 1 then
         begin
         ReDisplayPage := TRUE;
         Stats.FirstLine := Stats.FirstLine - 1;
         Stats.FirstPos := TextPage[Stats.FirstLine];
         Stats.LastLine := Stats.LastLine - 1;
         Stats.LastPos := TextPage[Stats.LastLine + 1] - 1;
         Stats.CurrentLine := Stats.CurrentLine - 1;
         end;
      if ((Page > 0) and
         (TextPage[Stats.LastLine] < ((Page - 1) * MaxPageSize) +
                                        MaxTextSize)) then
         begin
         Page := Page - 1;
         GetText;
         end;
      end;
   if TextPage[Stats.CurrentLine] + Stats.LocX - LtMargin > TextPage[Stats.CurrentLine + 1] t
      begin
      Stats.CurrentPos := TextPage[Stats.CurrentLine + 1] - 1;
      Stats.LocX := Stats.CurrentPos - TextPage[Stats.CurrentLine] + LtMargin;
      end
   else
      Stats.CurrentPos := TextPage[Stats.CurrentLine] + Stats.LocX - LtMargin;
   if (((TextStat = 11) or (TextStat = 22)) and (DataStat = 3)) and
      (Stats.CurrentPos > TmpNewNode^.BeginRange) then
      begin
      TmpNewNode^.EndRange := Stats.CurrentPos;
      ReDisplayPage := TRUE;
      end;
   GetHelpRange;
   end;

procedure ScrollDown;

   var
```

```
      TestNode   :  HelpNodePtr;

   begin
   ReDisplayPage := TRUE;
   if Stats.CurrentLine < TotalLines then
      begin
      Stats.FirstLine := Stats.FirstLine + 1;
      Stats.FirstPos := TextPage[Stats.FirstLine];
      Stats.CurrentLine := Stats.CurrentLine + 1;
      Stats.LastLine := Stats.LastLine + 1;
      Stats.LastPos := TextPage[Stats.LastLine + 1] - 1;
      end;
   if (TextPage[Stats.FirstLine] > ((Page + 1) * MaxPageSize)) then
      begin
      Page := Page + 1;
      GetText;
      end;
   GetHelpRange;
   end;

procedure ScrollDown_AS;

   var
      TestNode   :  HelpNodePtr;

   begin
   if Stats.CurrentLine < Stats.LastLine then
      begin
      Stats.LocY := Stats.LocY + 1;
      Stats.CurrentLine := Stats.CurrentLine + 1;
      end
   else
      begin
      ReDisplayPage := TRUE;
      if Stats.CurrentLine < TotalLines then
         begin
         Stats.FirstLine := Stats.FirstLine + 1;
         Stats.FirstPos := TextPage[Stats.FirstLine];
         Stats.CurrentLine := Stats.CurrentLine + 1;
         Stats.LastLine := Stats.LastLine + 1;
         Stats.LastPos := TextPage[Stats.LastLine + 1] - 1;
         end;
      if (TextPage[Stats.FirstLine] > ((Page + 1) * MaxPageSize)) then
         begin
         Page := Page + 1;
         GetText;
         end;
      end;
   if TextPage[Stats.CurrentLine] + Stats.LocX - LtMargin > TextPage[Stats.CurrentLine + 1] then
      begin
      Stats.CurrentPos := TextPage[Stats.CurrentLine + 1] - 1;
      Stats.LocX := Stats.CurrentPos - TextPage[Stats.CurrentLine] + LtMargin;
      end
   else
      Stats.CurrentPos := TextPage[Stats.CurrentLine] + Stats.LocX - LtMargin;
   if ((TextStat = 11) or (TextStat = 22)) and (DataStat = 3) then
      begin
      TmpNewNode^.EndRange := Stats.CurrentPos;
      ReDisplayPage := TRUE;
      end;
   GetHelpRange;
   end;

procedure LastChar;

   {Determine if the Character will be the first character on the Stats,
```

```
     on the line, and of the paper}

    var
       TestNode   :  HelpNodePtr;

    begin
    if Stats.CurrentPos > 1 then
       begin
       Stats.CurrentPos := Stats.CurrentPos - 1;
       if Stats.CurrentPos < TextPage[Stats.CurrentLine] then
          begin
          Stats.CurrentLine := Stats.CurrentLine - 1;
          Stats.LocX := Stats.CurrentPos - TextPage[Stats.CurrentLine] + LtMargin;
          if Stats.CurrentPos < Stats.FirstPos then
             begin
             Stats.FirstLine := Stats.FirstLine - 1;
             Stats.FirstPos := TextPage[Stats.FirstLine];
             Stats.LastLine := Stats.LastLine - 1;
             Stats.LastPos := TextPage[Stats.LastLine + 1] - 1;
             ReDisplayPage := TRUE;
             end
          else
             Stats.LocY := Stats.LocY - 1;
          end
       else
          Stats.LocX := Stats.LocX - 1;
       if ((Page > 0) and
          (Stats.LastPos < ((Page - 1) * MaxPageSize) + MaxTextSize)) then
          begin
          Page := Page - 1;
          GetText;
          ReDisplayPage := TRUE;
          end;
       if ((TextStat = 11) or (TextStat = 22)) and (DataStat = 3) then
          begin
          FastWrite(Stats.LocX, Stats.LocY,
                    TextBuffer[Stats.CurrentPos - (Page * MaxPageSize)],
                    NormalText, NormalBackground);
          end;
       GoToXY(Stats.LocX, Stats.LocY);
       end;
    GetHelpRange;
    end;

procedure NextChar;

    {Determine if the Character is the last character on the Stats,
     on the line, and of the paper}

    var
       TestNode   :  HelpNodePtr;

    begin
    if Stats.CurrentPos < TotalChar then
       begin
       if (((TextStat = 11) and (DataStat = 3)) or
          ((TextStat = 22) and (DataStat = 3))) and
          (Stats.CurrentPos < Stats.LastPos) then
          begin
          FastWrite(Stats.LocX, Stats.LocY,
                    TextBuffer[Stats.CurrentPos - (Page * MaxPageSize)],
                    SelectedText xor 128, SelectedBackground);
          end;
       Stats.CurrentPos := Stats.CurrentPos + 1;
       if Stats.CurrentPos = TextPage[Stats.CurrentLine + 1] then
          begin
```

```
            Stats.LocX := LtMargin;
            Stats.CurrentLine := Stats.CurrentLine + 1;
            if Stats.CurrentPos > Stats.LastPos then
               begin
               Stats.FirstLine := Stats.FirstLine + 1;
               Stats.FirstPos := TextPage[Stats.FirstLine];
               Stats.LastLine := Stats.LastLine + 1;
               Stats.LastPos := TextPage[Stats.LastLine + 1] - 1;
               TmpNewNode^.EndRange := Stats.CurrentPos - 1;
               ReDisplayPage := TRUE;
               end
            else
               begin
               Stats.LocY := Stats.LocY + 1;
               GoToXY(Stats.LocX, Stats.LocY);
               end
            end
         else
            begin
            Stats.LocX := Stats.LocX + 1;
            GoToXY(Stats.LocX, Stats.LocY);
            end;
         end;
      if (Stats.FirstPos > ((Page + 1) * MaxPageSize)) then
         begin
         Page := Page + 1;
         GetText;
         ReDisplayPage := TRUE;
         end;
      GetHelpRange;
      end;

procedure PageUp;

   var
      TestNode   :  HelpNodePtr;

   begin
   if Stats.FirstLine - ScrTextLines < 0 then
      FirstPage
   else
      begin
      Stats.LastLine := Stats.FirstLine;
      Stats.FirstLine := Stats.FirstLine - ScrTextLines + 2;
      Stats.FirstPos := TextPage[Stats.FirstLine];
      Stats.LastPos := TextPage[Stats.LastLine + 1] - 1;
      Stats.CurrentLine := Stats.FirstLine + Stats.LocY - 2;
      if TextPage[Stats.CurrentLine] + Stats.LocX - LtMargin > TextPage[Stats.CurrentLine +
         begin
         Stats.CurrentPos := TextPage[Stats.CurrentLine + 1] - 1;
         Stats.LocX := LtMargin + TextPage[Stats.CurrentLine + 1] - TextPage[Stats.CurrentLi
         end
      else
         Stats.CurrentPos := TextPage[Stats.CurrentLine] + Stats.LocX - LtMargin;
      GetHelpRange;
      if ((Page > 0) and
         (TextPage[Stats.LastLine] < ((Page - 1) * MaxPageSize) +
                                     MaxTextSize)) then
         begin
         Page := Page - 1;
         GetText;
         end;
      end;
   ReDisplayPage := TRUE;
   end;
```

```
procedure PageDown;

   var
      TestNode   :  HelpNodePtr;

   begin
   if Stats.LastLine + ScrTextLines - 3 > TotalLines then
      LastPage
   else
      begin
      Stats.FirstLine := Stats.LastLine;
      Stats.LastLine := Stats.LastLine + ScrTextLines - 2;
      Stats.FirstPos := TextPage[Stats.FirstLine];
      if Stats.LastLine < TotalLines then
         Stats.LastPos := TextPage[Stats.LastLine + 1] - 1
      else
         Stats.LastPos := TotalChar;
      Stats.CurrentLine := Stats.FirstLine + Stats.LocY - 2;
      if TextPage[Stats.CurrentLine] + Stats.LocX - LtMargin > TextPage[Stats.CurrentLine + 1
         begin
         Stats.CurrentPos := TextPage[Stats.CurrentLine + 1] - 1;
         Stats.LocX := LtMargin + TextPage[Stats.CurrentLine + 1] - TextPage[Stats.CurrentLi
         end
      else
         Stats.CurrentPos := TextPage[Stats.CurrentLine] + Stats.LocX - LtMargin;
      GetHelpRange;
      if (TextPage[Stats.FirstLine] > ((Page + 1) * MaxPageSize)) then
         begin
         Page := Page + 1;
         GetText;
         end;
      end;
   ReDisplayPage := TRUE;
   end;

end.
```

Appendix C05

ABRAHW.PAS

```
{*************************************************************************}
{ PARENT PROGRAM:    Advanced Bilingual Reading Assistant                 }
{                    Advanced Bilingual Reading Assistant - Authoring System }
{ PROGRAMMER:        Raymundo A. Q. Marcelo                               }
{ DATE:              23-April-1995                                        }
{                                                                         }
{ UNIT NAME:         ABRAHW                                               }
{                                                                         }
{ PURPOSE:           This unit contains the functions and procedures required}
{                    by the ABRA and ABRA-AS programs to display information }
{                    from those programs onto the screen.                 }
{*************************************************************************}

unit ABRAHW;

interface

uses Dos, Crt, TP_REF, ABRAVARS, ABRABACK, ABRAED, ABRATEXT, ABRALEV;

{   Procedures used by ABRA and ABRA-AS   }
procedure NextHotWord;
procedure PrevHotWord;
procedure DisplayDefinition;

{   Procedures to mark the beginnings and ends of the highlighted words  }
{                          Used by ABRA-AS only                          }
procedure BeginNewHotWord(var TmpNewNode   :  HelpNodePtr);
procedure EndNewHotWord(var TmpNewNode   :  HelpNodePtr);

{   Procedures to define and edit the highlighted word                   }
procedure GetHotWordDefinition;
procedure EditHWDefn;
procedure LoadEditor(var FirstStr  :  EditString;
                         i         :  integer);

{   Procedures to import a Highlighted Word File into ABRA-AS            }
procedure NewHotWordFile; {NewLevel}
procedure AskForHotWord;

{   Other Procedures to edit the level for the highlighted words         }
procedure DeleteHotWord;
procedure StoreDefinition(var FirstStr :  EditString);

implementation

var
   TestNode,
   TmpNode,
   TmpNewNode,
   TmpCurrent     :  HelpNodePtr;

{  Procedures involved with the Hot Words  }
procedure BeginNewHotWord(var TmpNewNode   :  HelpNodePtr);

   var
      NewNode   :  HelpNodePtr;

   begin
   ModifiedLevel := TRUE;
   New(NewNode);
   NewNode^.BeginRange := Stats.CurrentPos;
   NewNode^.EndRange := Stats.CurrentPos - 1;
   NewNode^.PrevNode := nil;
   NewNode^.NextNode := nil;
   NewNode^.NextPartner := nil;
   NewNode^.PrevDisplay := nil;
```

```
   NewNode^.NextDisplay := nil;
   NewNode^.Index := H.BufferEnd;

   if TmpNode = nil then
      begin
      TmpNode := NewNode;
      NewNode^.PrevPartner := nil;
      ReDisplayPage := True;
      Stats.CurrentHotWord := TmpNode;
      if IntegratingFile then
         begin
         HeadNewHW^.Defn := HeadNewHW^.Defn + '>';
         move(HeadNewHW^.Defn[1], StringBuffer[H.BufferEnd], Length(HeadNewHW^.Defn));
         end
      else
         BottomBackground;
      end
   else
      begin
      NewNode^.PrevPartner := TmpNewNode;
      TmpNewNode^.NextPartner := NewNode;
      end;
   TmpNewNode := NewNode;
   end;

procedure DisplayDefinition;

   var
      Code,
      i, j, k    :   integer;
      NumStr     :   string;

   begin
   BottomBackground;
   TextBackground(HelpBackground);
   TextColor(HelpText);
   if Stats.CurrentHotWord <> nil then
      begin
      if (Stats.CurrentHotWord^.BeginRange < Stats.FirstPos) or
         (Stats.CurrentHotWord^.EndRange > Stats.LastPos) then
         FindPage(Stats.CurrentHotWord);
      i := Stats.CurrentHotWord^.Index;
      j := 1;
      k := LtMargin;
      while (StringBuffer[i] <> '>') do
         begin
         GoToXY(k, ScrTextLines + j);
         if (StringBuffer[i] = '}') then
            begin
            k := LtMargin;
            j := j + 1;
            i := i + 1;
            end
         else if (StringBuffer[i] = '[') then
            begin
            GetXYLocation(StringBuffer, i, k, j);
            k := k + LtMargin;
            i := i + 1;
            end
         else
            begin
            write(StringBuffer[i]);
            i := i + 1;
            k := k + 1;
            end;
         end;
```

```
      end;
   end;

procedure LoadEditor(var FirstStr   :  EditString;
                         i          :  integer);

   var
      ch1 : string;
      j, x, y :  integer;
      NewStr,
      PresStr  :  EditString;

   begin
   y := 1;
   x := LtMargin;
   FirstStr := nil;
   ch1 := ' ';

   while (StringBuffer[i] <> '>') do
      begin
      if (StringBuffer[i] = '}') then
         begin
         x := LtMargin;
         y := y + 1;
         i := i + 1;
         new(NewStr);
         NewStr^.str := '';
         NewStr^.x := LtMargin;
         NewStr^.y := ScrTextLines + 1;
         NewStr^.next := nil;
         if FirstStr = nil then
            begin
            FirstStr := NewStr;
            NewStr^.prev := nil;
            end
         else
            begin
            PresStr^.next := NewStr;
            NewStr^.prev := PresStr;
            end;
         PresStr := NewStr;
         end
      else if (StringBuffer[i] = '[') then
         begin
         GetXYLocation(StringBuffer, i, x, y);
         i := i + 1;

         for j := 1 to y do
            begin
            new(NewStr);
            NewStr^.str := '';
            NewStr^.x := LtMargin;
            NewStr^.y := ScrTextLines + j;
            NewStr^.next := nil;
            if FirstStr = nil then
               begin
               FirstStr := NewStr;
               NewStr^.prev := nil;
               end
            else
               begin
               PresStr^.next := NewStr;
               NewStr^.prev := PresStr;
               end;
            PresStr := NewStr;
            end;
```

```
            for j := 1 to x do
               PresStr^.str := PresStr^.Str + ' ';
            GoToXY(x + LtMargin, y + ScrTextLines);
            end
         else
            begin
            if FirstStr = nil then
               begin
               new(NewStr);
               NewStr^.str := '';
               NewStr^.x := LtMargin;
               NewStr^.y := ScrTextLines + 1;
               NewStr^.next := nil;
               if FirstStr = nil then
                  begin
                  FirstStr := NewStr;
                  NewStr^.prev := nil;
                  end
               else
                  begin
                  PresStr^.next := NewStr;
                  NewStr^.prev := PresStr;
                  end;
               PresStr := NewStr;
               end;
            PresStr^.str := PresStr^.str + StringBuffer[i];
            i := i + 1;
            x := x + 1;
            end;
         end;
      end;

procedure StoreDefinition(var FirstStr :  EditString);

   var
      PresStr  :  EditString;
      i, j, x  :  integer;
      NumStr   :  string;
      ASpace   :  boolean;

   begin
   ModifiedLevel := TRUE;
   while FirstStr <> nil do
      begin
      x := LtMargin;
      if FirstStr^.str <> '' then
         begin
         ASpace := True;
         for i := 1 to Length(FirstStr^.str) do
            begin
            if (ASpace) and (FirstStr^.str[i] = ' ') then
               x := x + 1
            else if (ASpace) and (FirstStr^.str[i] <> ' ') then
               begin
               ASpace := False;
               StringBuffer[H.BufferEnd] := '[';
               H.BufferEnd := H.BufferEnd + 1;
               str(i, NumStr);
               for j := 1 to Length(NumStr) do
                  begin
                  StringBuffer[H.BufferEnd] := NumStr[j];
                  H.BufferEnd := H.BufferEnd + 1;
                  end;
               StringBuffer[H.BufferEnd] := ',';
               H.BufferEnd := H.BufferEnd + 1;
               str(FirstStr^.y - ScrTextLines, NumStr);
```

```
                for j := 1 to Length(NumStr) do
                   begin
                   StringBuffer[H.BufferEnd] := NumStr[j];
                   H.BufferEnd := H.BufferEnd + 1;
                   end;
                StringBuffer[H.BufferEnd] := ']';
                H.BufferEnd := H.BufferEnd + 1;
                StringBuffer[H.BufferEnd] := FirstStr^.str[i];
                H.BufferEnd := H.BufferEnd + 1;
                end
             else
                begin
                StringBuffer[H.BufferEnd] := FirstStr^.str[i];
                H.BufferEnd := H.BufferEnd + 1;
                end;
             end;
          end;
       PresStr := FirstStr;
       FirstStr := FirstStr^.next;
       dispose(PresStr);
       end;
    StringBuffer[H.BufferEnd] := '>';
    H.BufferEnd := H.BufferEnd + 1;
    end;

procedure GetHotWordDefinition;

    var
       i, j, x, y  :  integer;
       S           :  string;
       ctype       :  KeyType;
       spec        :  boolean;
       ch          :  char;
       ch1         :  string;
       InsertOn,
       ASpace,
       GHWDone     :  boolean;
       TmpStr,
       FirstStr,
       PresStr,
       NewStr      :  EditString;
       PrevX,
       PrevY,
       StrPos      :  integer;
       TmpHW       :  HelpNodePtr;
       TString,
       NumStr      :  string;

    begin
    ModifiedLevel := TRUE;
    InsertOn := False;
    GHWDone := False;
    FirstStr := nil;
    PrevX := Stats.LocX;
    PrevY := Stats.LocY;
    TString := 'Editing a New Highlighted Word(s) Definition';
    j := 40 - TRUNC(Length(TString)/2);
    for i := 2 to j do
       FastWrite(i, ScrTextLines, ' ', StatusText, StatusBackground);
    for i := 1 to Length(TString) do
       FastWrite(j + i - 1, ScrTextLines, TString[i], StatusText,
                 StatusBackground);
    for i := j + Length(TString) to MaxLineLen - 1 do
       FastWrite(i, ScrTextLines, ' ', StatusText, StatusBackground);

    Stats.LocX := LtMargin;
```

```
   Stats.LocY := ScrTextLines + 1;
   BottomBackground;
   AbraEditor(FirstStr, LtMargin, ScrTextLines + 1, MaxLineLen - LtMargin,
              MaxTextLines - 1, LtMargin, HelpText, HelpBackground);
   StoreDefinition(FirstStr);
   FindXYBeginRange(Stats.CurrentHotWord, Stats.CurrentLine, Stats.LocX, Stats.LocY);
   end;

procedure EditHWDefn;

   var
      i, j, x, y,
      PrevX,
      PrevY      :   integer;
      EHWDone    :   boolean;
      ctype      :  KeyType;
      spec       :  boolean;
      ch         :  char;
      ch1        :  string;
      InsertOn,
      ASpace     :  boolean;
      TmpStr,
      FirstStr,
      PresStr,
      NewStr     :  EditString;
      StrPos     :  integer;
      TmpHW      :  HelpNodePtr;
      TString,
      NumStr     :  string;

   begin
   ModifiedLevel := TRUE;
   InsertOn := True;
   EHWDone := False;
   PrevX := Stats.LocX;
   PrevY := Stats.LocY;
   TString := 'Editing the Highlighted Word(s) Definition';
   j := 40 - TRUNC(Length(TString)/2);
   for i := 2 to j do
      FastWrite(i, ScrTextLines, ' ', StatusText, StatusBackground);
   for i := 1 to Length(TString) do
      FastWrite(j + i - 1, ScrTextLines, TString[i], StatusText,
                StatusBackground);
   for i := j + Length(TString) to MaxLineLen - 1 do
      FastWrite(i, ScrTextLines, ' ', StatusText, StatusBackground);

   i := Stats.CurrentHotWord^.Index;
   LoadEditor(FirstStr, i);
   BottomBackground;
   AbraEditor(FirstStr, LtMargin, ScrTextLines + 1, MaxLineLen - LtMargin,
              MaxTextLines - 1, FirstStr^.x, HelpText, HelpBackground);
   TmpHW := Stats.CurrentHotWord;
   while TmpHW <> nil do
      begin
      TmpHW^.Index := H.BufferEnd;
      TmpHW := TmpHW^.NextPartner;
      end;
   ChangeIndex;
   StoreDefinition(FirstStr);
   Stats.LocX := PrevX;
   Stats.LocY := PrevY;
   end;

procedure EndNewHotWord(var TmpNewNode    :  HelpNodePtr);

   var
```

```
   TmpDisplay,
   TmpCurrent,
   HotWord  :  HelpNodePtr;
   PosFound :  boolean;

begin
ModifiedLevel := True;
if (DataStat = 3) then
   TmpNewNode^.EndRange := Stats.CurrentPos - 1;

{  Checks to see if there are any Hot Words in the list  }
if H.HelpNodeList = nil then
   begin
   H.HelpNodeList := TmpNode;
   Stats.CurrentHotWord := TmpNode;
   Stats.FirstHotWord := TmpNode;
   end

{  Check to see if TmpNode range is between the First and Last HotWords }
else if (TmpNode^.BeginRange > Stats.FirstHotWord^.BeginRange) and
   (TmpNode^.BeginRange < Stats.LastHotWord^.BeginRange) then
   begin
   HotWord := Stats.FirstHotWord;
   PosFound := False;
   while (not PosFound) do
      if (TmpNode^.BeginRange > HotWord^.NextNode^.BeginRange) then
         HotWord := HotWord^.NextNode
      else
         PosFound := True;
   TmpNode^.NextNode := HotWord^.NextNode;
   HotWord^.NextNode^.PrevNode := TmpNode;
   HotWord^.NextNode := TmpNode;
   TmpNode^.PrevNode := HotWord;
   end

{  Check to see if new node fits before the First Hot Word on screen  }
else if (TmpNode^.BeginRange < Stats.FirstHotWord^.BeginRange) then
   begin
   if H.HelpNodeList = Stats.FirstHotWord then
      begin
      TmpNode^.NextNode := H.HelpNodeList;
      H.HelpNodeList^.PrevNode := TmpNode;
      H.HelpNodeList := TmpNode;
      end
   else
      begin
      Stats.FirstHotWord^.PrevNode^.NextNode := TmpNode;
      TmpNode^.PrevNode := Stats.FirstHotWord^.PrevNode;
      TmpNode^.NextNode := Stats.FirstHotWord;
      Stats.FirstHotWord^.PrevNode := TmpNode;
      end;
   Stats.FirstHotWord := TmpNode;
   end

{  Check to see if the new node fits after the Last Hot Word on screen  }
else if (TmpNode^.BeginRange > Stats.LastHotWord^.BeginRange) then
   begin
   TmpNode^.PrevNode := Stats.LastHotWord;
   TmpNode^.NextNode := Stats.LastHotWord^.NextNode;
   Stats.LastHotWord^.NextNode := TmpNode;
   if TmpNode^.NextNode <> nil then
      Stats.LastHotWord^.NextNode^.PrevNode := TmpNode;
   Stats.LastHotWord := TmpNode;
   end

{  Check to see if there are any Hot Words on the page  }
```

```
else if Stats.LastHotWord = nil then
   begin
   Stats.FirstHotWord^.NextNode := TmpNode;
   TmpNode^.PrevNode := Stats.FirstHotWord;
   Stats.FirstHotWord := TmpNode;
   end;

Stats.CurrentHotWord := TmpNode;
TmpCurrent := TmpNode;
while (TmpCurrent <> nil) do
   begin
   {  Empty Display List  }
   if (H.DisplayList = nil) or
      (TmpCurrent^.BeginRange < H.DisplayList^.BeginRange) then
      begin
      TmpCurrent^.PrevDisplay := nil;
      if (H.DisplayList = nil) then
         TmpCurrent^.NextDisplay := nil
      else
         TmpCurrent^.NextDisplay := H.DisplayList;
      H.DisplayList := TmpCurrent;
      end
   else
      {   Hot Word part begins prior to the first display on page  }
      begin
      if (TmpCurrent^.BeginRange < Stats.FirstDisplay^.BeginRange) then
         begin
         TmpDisplay := Stats.FirstDisplay;
         PosFound := FALSE;
         while (not PosFound) do
            if (TmpDisplay^.BeginRange > TmpCurrent^.BeginRange) then
               TmpDisplay := TmpDisplay^.PrevDisplay
            else
               PosFound := TRUE;
         end
      else
         {   Hot Word part begins after the last displayed word }
         begin
         if (TmpCurrent^.BeginRange > Stats.LastDisplay^.BeginRange) then
            TmpDisplay := Stats.LastDisplay
         {   Hot Word part is between the first and last displayed word  }
         else
            TmpDisplay := Stats.FirstDisplay;
         PosFound := FALSE;
         while (not PosFound) do
            if (TmpDisplay^.BeginRange < TmpCurrent^.BeginRange) then
               if (TmpDisplay^.NextDisplay <> nil) then
                  TmpDisplay := TmpDisplay^.NextDisplay
               else
                  PosFound := TRUE
            else
               begin
               PosFound := TRUE;
               TmpDisplay := TmpDisplay^.PrevDisplay;
               end;
         end;
      TmpCurrent^.NextDisplay := TmpDisplay^.NextDisplay;
      TmpDisplay^.NextDisplay := TmpCurrent;
      TmpCurrent^.PrevDisplay := TmpDisplay;
      TmpCurrent^.NextDisplay^.PrevDisplay := TmpCurrent;
      end;
   TmpCurrent := TmpCurrent^.NextPartner;
   end;

GetHelpRange;
UpdateScrMatrix;
```

```
   ReDisplayPage := TRUE;
   if (not IntegratingFile) then
      begin
      GetHotWordDefinition;
      Stats.CurrentPos := Stats.CurrentHotWord^.BeginRange;
      end
   else
      begin
      DisplayDefinition;
      H.BufferEnd := H.BufferEnd + Length(HeadNewHW^.Defn);
      end;
   TmpNode := nil;
   TmpNewNode := nil;
   end;

procedure DeleteHotWord;

   var
      TempPtr,
      TmpPtr    :   HelpNodePtr;

   begin
   if Stats.CurrentHotWord <> nil then
      begin
      ModifiedLevel := True;
      TempPtr := Stats.CurrentHotWord;
      while (TempPtr <> nil) do
         begin
         if (TempPtr^.NextNode <> nil) or (TempPtr^.PrevNode <> nil) then
            if (TempPtr = H.HelpNodeList) then
               begin
               if H.HelpNodeList^.NextNode <> nil then
                  begin
                  H.HelpNodeList := H.HelpNodeList^.NextNode;
                  H.HelpNodeList^.PrevNode := nil;
                  Stats.CurrentHotWord := Stats.CurrentHotWord^.NextNode;
                  Stats.FirstHotWord := H.HelpNodeList;
                  end
               else
                  begin
                  H.HelpNodeList := nil;
                  Stats.CurrentHotWord := nil;
                  Stats.FirstHotWord := nil;
                  end
               end
            else
               begin
               Stats.CurrentHotWord^.PrevNode^.NextNode := Stats.CurrentHotWord^.NextNode;
               Stats.CurrentHotWord^.NextNode^.PrevNode := Stats.CurrentHotWord^.PrevNode;
               Stats.CurrentHotWord := Stats.CurrentHotWord^.NextNode;
               end;

         {  Relink other Display Pointers around the Deleted Node  }
         if (TempPtr = H.DisplayList) then
            if (H.DisplayList^.NextDisplay <> nil) then
               begin
               H.DisplayList := TempPtr^.NextDisplay;
               H.DisplayList^.PrevDisplay := nil;
               Stats.FirstDisplay := H.DisplayList;
               end
            else
               begin
               H.DisplayList := nil;
               Stats.FirstDisplay := nil;
               end
         else
```

```
            begin
            if (TempPtr = Stats.FirstDisplay) then
               Stats.FirstDisplay := Stats.FirstDisplay^.NextDisplay;
            if (TempPtr^.NextDisplay <> nil) then
               begin
               TmpPtr := TempPtr;
               TempPtr^.NextDisplay^.PrevDisplay := TmpPtr^.PrevDisplay;
               end;
            TempPtr^.PrevDisplay^.NextDisplay := TmpPtr^.NextDisplay;
            end;
         TmpPtr := TempPtr^.NextPartner;
         Dispose(TempPtr);
         TempPtr := TmpPtr;
         end;
      GetHelpRange;
      UpdateScrMatrix;
      ReDisplayPage := True;
      end;
   end;

procedure NextHotWord;

   var
      TmpNewLevelNode   :  TextNodePtr;
      TString           :  string;
      i                 :  integer;

   begin
   BottomBackground;
   if (IntegratingFile) and ((DataStat <> 3) or (DataStat <> 4)) and (HeadNewHW <> nil) then
      begin
      TmpNewLevelNode := HeadNewHW;
      HeadNewHW := HeadNewHW^.Next;
      Dispose(TmpNewLevelNode);
      AskForHotWord;
      end
   else if (IntegratingFile) and ((DataStat <> 3) or (DataStat <> 4)) and (HeadNewHW = nil) t
      IntegratingFile := False
   else if (H.HelpNodeList = nil) then
      begin
      BottomBackground;
      TString := 'There are presently no highlighted words present.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 1, TString[i],
                   HelpText, HelpBackground);
      TString := 'Please scroll to the word(s) to be highlighted.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 2, TString[i],
                   HelpText, HelpBackground);
      TString := 'Mark and Unmark the word(s) with F4.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 3, TString[i],
                   HelpText, HelpBackground);
      TString := 'Type in the definition within this lower block.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 4, TString[i],
                   HelpText, HelpBackground);
      end
   else if (Stats.CurrentHotWord <> nil) and
      (Stats.CurrentHotWord^.NextNode <> nil) then
      begin
      Stats.CurrentHotWord := Stats.CurrentHotWord^.NextNode;
      Stats.CurrentPos := Stats.CurrentHotWord^.BeginRange;
      if (Stats.CurrentHotWord^.BeginRange < Stats.FirstPos) or
         (Stats.CurrentHotWord^.EndRange > Stats.LastPos) then
         FindPage(Stats.CurrentHotWord)
```

```
      else
         FindXYBeginRange(Stats.CurrentHotWord, Stats.CurrentLine, Stats.LocX, Stats.LocY);
      ReDisplayPage := TRUE;
      end;
   end;

procedure PrevHotWord;

   var
      TString   :   string;
      i         :   integer;

   begin
   BottomBackground;
   if Stats.CurrentHotWord <> nil then
      begin
      if Stats.CurrentHotWord^.PrevNode <> nil then
         begin
         Stats.CurrentHotWord := Stats.CurrentHotWord^.PrevNode;
         Stats.CurrentPos := Stats.CurrentHotWord^.BeginRange;
         if (Stats.CurrentHotWord^.BeginRange < Stats.FirstPos) or
            (Stats.CurrentHotWord^.EndRange > Stats.LastPos) then
            FindPage(Stats.CurrentHotWord)
         else
            FindXYBeginRange(Stats.CurrentHotWord, Stats.CurrentLine, Stats.LocX, Stats.LocY
         ReDisplayPage := TRUE;
         end;
      end
   else if (H.HelpNodeList = nil) then
      begin
      BottomBackground;
      TString := 'There are presently no highlighted words present.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 1, TString[i],
                   HelpText, HelpBackground);
      TString := 'Please scroll to the word(s) to be highlighted.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 2, TString[i],
                   HelpText, HelpBackground);
      TString := 'Mark and Unmark the word(s) with F4.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 3, TString[i],
                   HelpText, HelpBackground);
      TString := 'Type in the definition within this lower block.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 4, TString[i],
                   HelpText, HelpBackground);
      end;
   end;

procedure NewHotWordFile;

   var
      ALine,
      tempstr    :   String;
      i, j,
      StartQuestion,
      HelpLevel,
      ErrCode,
      B_rng,
      E_rng,
      BIndex,
      SemiPos        :   integer;
      NewNewHW,
      CurrNewHW      :   TextNodePtr;
      ch             :   char;
```

```
begin
KillHelpNodeList;
H.HelpNodeList := nil;
H.BufferEnd := 1;
ModifiedLevel := True;
IntegratingFile := True;

DisplayDirectory('*.*');
BottomBackground;
TextBackground(HelpBackground);
TextColor(HelpText);
GoToXY(LtMargin, ScrTextLines + 2);
if Files.CurrLevel^.LevName = '' then
   begin
   write('Please write the name for this level.  ');
   readln(Files.CurrLevel^.LevName);
   end;
write('Does displaying your definitions require more than 6 lines? (Y/N) ');
readln(ch);
if UpCase(ch) = 'Y' then
   Files.CurrLevel^.Interlinear := TRUE
else
   Files.CurrLevel^.Interlinear := FALSE;
GoToXY(LtMargin, ScrTextLines + 4);
write('Which file contains the hot words and definitions?  ');
readln(tempstr);

Assign(LevelFile, tempstr);
{$I-} Reset(LevelFile); {$I+}

CurrNewHW := nil;
HeadNewHW := nil;

WHILE NOT EOF(LevelFile) DO
   BEGIN
   readln(LevelFile, ALine);
   SemiPos := 1;
   while (ALine[SemiPos] <> '|') and (SemiPos <= Length(ALine)) do
      SemiPos := SemiPos + 1;
   if SemiPos > Length(ALine) then
      SemiPos := 0;
   if SemiPos > 0 then
      begin
      new(NewNewHW);
      NewNewHW^.HotWord := '';
      NewNewHW^.Defn := '';
      NewNewHW^.Next := nil;
      NewNewHW^.HotWord := Copy(ALine, 1, SemiPos - 1);
      NewNewHW^.Defn := Copy(ALine, SemiPos + 1, Length(ALine));
      end
   else
      begin
      if ALine <> '' then
         begin
         if NewNewHW^.Defn <> '' then
            begin
            insert('}', NewNewHW^.Defn, Length(NewNewHW^.Defn) + 1);
            insert(ALine, NewNewHW^.Defn, Length(NewNewHW^.Defn) + 1);
            end;
         end
      else
         begin
         if HeadNewHW = nil then
            HeadNewHW := NewNewHW
         else
```

```
                CurrNewHW^.Next := NewNewHW;
              CurrNewHW := NewNewHW;
              end;
           end;
        end;
     if HeadNewHW = nil then
        HeadNewHW := NewNewHW
     else
        CurrNewHW^.Next := NewNewHW;
     CurrNewHW := NewNewHW;

     close(LevelFile);
     ResetScrMatrix;
     AskForHotWord;
     FirstPage;
     ReDisplayPage := TRUE;
     end;

procedure AskForHotWord;

     begin
     BottomBackground;
     TextBackground(HelpBackground);
     TextColor(HelpText);
     GoToXY(LtMargin, ScrTextLines + 1);
     writeln('Please Mark New Hot Word');
     GoToXY(LtMargin, ScrTextLines + 2);
     writeln('   Hot Word :  ', HeadNewHW^.HotWord);
     GoToXY(LtMargin, ScrTextLines + 3);
     writeln('   Definition :  ', HeadNewHW^.Defn);
     ReDisplayPage := TRUE;
     end;

end.
```

Appendix C06

ABRAQUES.PAS

```
{*************************************************************************}
{ PARENT PROGRAM:    Advanced Bilingual Reading Assistant                 }
{                    Advanced Bilingual Reading Assistant - Authoring System }
{ PROGRAMMER:        Raymundo A. Q. Marcelo                               }
{ DATE:              23-April-1995                                        }
{                                                                         }
{ UNIT NAME:         ABRAQUES                                             }
{                                                                         }
{ PURPOSE:           This unit contains the functions and procedures required}
{                    by the ABRA and ABRA-AS programs to display question and}
{                    answer information on the screen.                    }
{*************************************************************************}

unit ABRAQUES;

interface

uses Dos, Crt, TP_REF, ABRAVARS, ABRABACK, ABRAED, ABRATEXT, ABRALEV;

procedure PrevQuestion;
procedure NextQuestion;

procedure GetNewQuestion;
procedure EditQuestion;
procedure GetLevelQuestions;
procedure SaveLevelQuestions;
procedure NewLevelQuestions;
procedure DeleteQuestion;

procedure FirstQuestion;
procedure DisplayQuestion;
procedure LoadQuestion(var FirstStr  :  EditString);

procedure DisplayAnswer;

procedure StoreNewAnswer(var FirstStr  :  EditString);
procedure BeginNewAnswer(FirstStr :  EditString;
                         PresStr  :  EditString;
                         P        :  integer);
procedure EndNewAnswer(FirstStr :  EditString;
                       PresStr  :  EditString;
                       P        :  integer);
procedure BeginNewTextAnswer;
procedure EndNewTextAnswer;

procedure GetMultipleChoiceAnswer(var FirstStr  :  EditString);
procedure LoadQandAAnswer(var FirstStr  :  EditString;
                              PStr      :  EditString);
procedure QandAAnswer(var FirstStr  :  EditString;
                      var PresStr   :  EditString);
procedure GetNewAnswer;
procedure StoreNewQuestion(var FirstStr  :  EditString);

implementation

var
   QuestionBuffer     :   StringBufferType;
   QuesPage           :   array[1..11] of integer;

{ QUESTION Procedures }

procedure DisplayQuestion;

   var
      i, x, y   :   integer;
```

```
    begin
    TextStat := 21;
    DataStat := 0;
    ReDisplayPage := TRUE;
    BottomBackground;
    TextBackground(HelpBackground);
    TextColor(HelpText);
    for i := 1 to 11 do
       QuesPage[i] := 0;

    i := Stats.CurrentQuestion^.QuestionStart;
    y := 1;
    QuesPage[y] := i;
    x := LtMargin;
    while (QuestionBuffer[i] <> '|') do
       begin
       if (QuestionBuffer[i] = '}') then
          begin
          x := LtMargin;
          y := y + 1;
          i := i + 1;
          QuesPage[y] := i;
          end
       else
          begin
          GoToXY(x, ScrTextLines + y);
          write(QuestionBuffer[i]);
          i := i + 1;
          x := x + 1;
          end;
       end;
    y := y + 1;
    QuesPage[y] := i;
    end;

procedure DisplayAnswer;

    var
       i, j, x, y,
       b, e, Code   :  integer;
       NewNode,
       HotWord      :  HelpNodePtr;
       NumStr       :  string;

    begin
    TextColor(AnswerText);
    TextBackground(AnswerBackground);
    i := Stats.CurrentQuestion^.AnswerStart;
    if Stats.CurrentQuestion^.QuestionStart = i then
       begin
       Stats.LocX := LtMargin;
       Stats.LocY := ScrTextLines + 1;
       end
    else if QuestionBuffer[i] = '<' then
       begin
       {Text Referenced Question}
       TextStat := 22;
       DataStat := 0;
       Stats.CurrentHotWord := nil;
       TmpNewNode := nil;
       HotWord := nil;
       while QuestionBuffer[i] <> '>' do
          begin
          New(NewNode);
          NewNode^.PrevNode := nil;
          NewNode^.NextNode := nil;
```

```
      NewNode^.NextPartner := nil;
      NewNode^.PrevDisplay := nil;
      NewNode^.NextDisplay := nil;
      NewNode^.Index := H.QBEnd + 1;
      if TmpNewNode = nil then
         begin
         NewNode^.PrevPartner := nil;
         TmpNewNode := NewNode;
         end
      else
         begin
         NewNode^.PrevPartner := HotWord;
         HotWord^.NextPartner := NewNode;
         end;
      HotWord := NewNode;
      GetRange(QuestionBuffer, i, HotWord^.BeginRange, HotWord^.EndRange);
      end;
   FindPage(TmpNewNode);
   UpdateScrMatrix;
   ReDisplayPage := TRUE;
   end
else if QuestionBuffer[i] = '[' then
   begin
   j := i;
   while QuestionBuffer[j] <> ']' do
      j := j + 1;
   j := j + 1;
   if QuestionBuffer[j] <> '>' then
      begin
      {Answer Description Question}
      TextStat := 24;
      DataStat := 0;
      GetXYLocation(QuestionBuffer, i, x, y);
      x := x + LtMargin;
      y := y + ScrTextLines;
      i := i + 1;

      while (QuestionBuffer[i] <> '>') do
         begin
         if (QuestionBuffer[i] = '}') then
            begin
            x := LtMargin;
            y := y + 1;
            i := i + 1;
            end
         else
            begin
            GoToXY(x, y);
            write(QuestionBuffer[i]);
            i := i + 1;
            x := x + 1;
            end;
         end;
      end
   else
      begin
      TextStat := 23;
      DataStat := 0;
      {Multiple Choice Question}
      while (QuestionBuffer[i] <> ']') do
         begin
         GetXYLocation(QuestionBuffer, i, b, e);
         y := 1;
         b := Stats.CurrentQuestion^.QuestionStart + b;
         e := Stats.CurrentQuestion^.QuestionStart + e;
         while QuesPage[y+1] <= b do
```

```
            y := y + 1;
          x := b - QuesPage[y] + LtMargin;
          for j := b to e do
            begin
            GoToXY(x, y + ScrTextLines);
            if QuestionBuffer[j] = '}' then
              begin
              x := LtMargin;
              y := y + 1;
              end
            else
              begin
              write(QuestionBuffer[j]);
              x := x + 1;
              end;
            end;
          end;
        end;
      end;
  end;

procedure GetLevelQuestions;

  var
    ALine,
    tmpstr       :  string;
    i, j,
    BIndex       :  integer;
    BeforeQuestion,
    NowQuestion,
    NewQuestion  :  QuestionPtr;

  begin
  H.Questions := nil;
  H.LevelName := Files.CurrLevel^.LevName;
  BIndex := 1;

  Assign(QuestionFile, Files.CurrLevel^.QuesName);
  Reset(QuestionFile);

  WHILE NOT EOF(QuestionFile) DO
    BEGIN
    ReadLn(QuestionFile, ALine);
    if (ALine[1] = '<') and (ALine[2] = '?') then
      begin
      i := 3;
      new(NewQuestion);
      NewQuestion^.PrevQuestion := nil;
      NewQuestion^.NextQuestion := nil;
      NewQuestion^.QuestionStart := BIndex;
      if H.Questions = nil then
        begin
        NowQuestion := NewQuestion;
        H.Questions := NewQuestion;
        end
      else
        begin
        BeforeQuestion := NowQuestion;
        NowQuestion := NewQuestion;
        BeforeQuestion^.NextQuestion := NowQuestion;
        NowQuestion^.PrevQuestion := BeforeQuestion;
        end;
      end
    else
      i := 1;
```

```
      for j := i to Length(ALine) do
         begin
         QuestionBuffer[BIndex] := ALine[j];
         if QuestionBuffer[BIndex] = '|' then
            NowQuestion^.AnswerStart := BIndex + 1;
         BIndex := BIndex + 1;
         end;
      end;

   H.QBEnd := BIndex;
   Close(QuestionFile);
   END;

procedure SaveLevelQuestions;

   var
      LS          :  string;
      i, j,
      BIndex      :  integer;
      BeforeQuestion,
      NowQuestion,
      NewQuestion :  QuestionPtr;
      E           :  ExtStr;

   begin
   if ModifiedQuestion then
      begin
      NowQuestion := H.Questions;
      if Files.CurrLevel^.QuesName = '' then
         begin
         Modified := TRUE;
         Str(Level, LS);
         if Level < 10 then
            E := '.&0' + LS
         else
            E := '.&' + LS;
         Files.CurrLevel^.QuesName := D + N + E;
         end;
      Assign(QuestionFile, Files.CurrLevel^.QuesName);
      Rewrite(QuestionFile);

      WHILE NowQuestion <> nil DO
         BEGIN
         i := NowQuestion^.QuestionStart;
         write(QuestionFile, '<?');
         while QuestionBuffer[i] <> '|' do
            begin
            write(QuestionFile, QuestionBuffer[i]);
            i := i + 1;
            end;
         write(QuestionFile, QuestionBuffer[i]);
         i := i + 1;
         i := NowQuestion^.AnswerStart;
         while QuestionBuffer[i] <> '>' do
            begin
            write(QuestionFile, QuestionBuffer[i]);
            i := i + 1;
            end;
         writeln(QuestionFile, QuestionBuffer[i]);
         i := i + 1;
         NowQuestion := NowQuestion^.NextQuestion;
         end;
      Close(QuestionFile);
      ModifiedQuestion := FALSE;
      end;
   END;
```

```
procedure StoreNewAnswer(var FirstStr  :  EditString);

   var
      i  :  integer;
      TmpString  :  EditString;

   begin
   while FirstStr <> nil do
      begin
      if FirstStr^.str <> '' then
         for i := 1 to Length(FirstStr^.str) do
            begin
            QuestionBuffer[H.QBEnd] := FirstStr^.str[i];
            H.QBEnd := H.QBEnd + 1;
            end;
      if FirstStr^.next <> nil then
         QuestionBuffer[H.QBEnd] := '}'
      else
         QuestionBuffer[H.QBEnd] := '>';
      H.QBEnd := H.QBEnd + 1;
      TmpString := FirstStr;
      FirstStr := FirstStr^.next;
      dispose(TmpString);
      end;
   end;

procedure BeginNewAnswer(FirstStr :  EditString;
                         PresStr  :  EditString;
                         P        :  integer);

   var
      NewNode   :  HelpNodePtr;

   begin
   New(NewNode);
   NewNode^.BeginRange := PresStr^.index + P - 1 - FirstStr^.index;
   NewNode^.EndRange := PresStr^.index + P - 1 - FirstStr^.index;
   NewNode^.PrevNode := nil;
   NewNode^.NextNode := nil;
   NewNode^.NextPartner := nil;
   NewNode^.Index := 1;

   if TmpNode = nil then
      begin
      TmpNode := NewNode;
      NewNode^.PrevPartner := nil;
      end
   else
      begin
      NewNode^.PrevPartner := TmpNewNode;
      TmpNewNode^.NextPartner := NewNode;
      end;
   TmpNewNode := NewNode;
   end;

procedure EndNewAnswer(FirstStr :  EditString;
                       PresStr  :  EditString;
                       P        :  integer);

   begin
   ModifiedQuestion := True;
   if (DataStat = 3) then
      TmpNewNode^.EndRange := PresStr^.index + P - 1 - FirstStr^.index;
   end;
```

```
procedure BeginNewTextAnswer;

   var
      NewNode    :  HelpNodePtr;

   begin
   TextStat := 22;
   DataStat := 3;
   New(NewNode);
   NewNode^.BeginRange := Stats.CurrentPos;
   NewNode^.EndRange := Stats.CurrentPos - 1;
   NewNode^.PrevNode := nil;
   NewNode^.NextNode := nil;
   NewNode^.NextPartner := nil;

   if TmpNode = nil then
      begin
      TmpNode := NewNode;
      NewNode^.PrevPartner := nil;
      ReDisplayPage := True;
      Stats.CurrentHotWord := TmpNode;
      end
   else
      begin
      NewNode^.PrevPartner := TmpNewNode;
      TmpNewNode^.NextPartner := NewNode;
      end;
   TmpNewNode := NewNode;
   end;

procedure EndNewTextAnswer;

   var
      HotWord   :  HelpNodePtr;
      PosFound  :  boolean;

   begin
   ModifiedQuestion := True;
   if (DataStat = 3) then
      TmpNewNode^.EndRange := Stats.CurrentPos - 1;

   UpdateScrMatrix;
   ReDisplayPage := TRUE;
   end;

procedure LoadQuestion(var FirstStr   :  EditString);

   var
      NewStr,
      PresStr  :  EditString;
      i        :  integer;

   begin
   new(NewStr);
   NewStr^.x := LtMargin;
   NewStr^.y := ScrTextLines + 1;
   NewStr^.index := Stats.CurrentQuestion^.QuestionStart;
   NewStr^.str := '';
   NewStr^.prev := nil;
   NewStr^.next := nil;
   FirstStr := NewStr;
   PresStr := NewStr;
   i := Stats.CurrentQuestion^.QuestionStart;
   while QuestionBuffer[i] <> '|' do
      begin
      if QuestionBuffer[i] = '}' then
```

```
         begin
         i := i + 1;
         new(NewStr);
         NewStr^.x := LtMargin;
         NewStr^.y := PresStr^.y + 1;
         NewStr^.index := i;
         NewStr^.str := '';
         NewStr^.prev := PresStr;
         PresStr^.next := NewStr;
         NewStr^.next := nil;
         PresStr := NewStr;
         end
      else
         begin
         PresStr^.str := PresStr^.str + QuestionBuffer[i];
         i := i + 1;
         end;
      end;
   end;

procedure LoadQandAAnswer(var FirstStr  :  EditString;
                              PStr      :  EditString);

   var
      NewStr,
      PresStr :  EditString;
      i       :  integer;

   begin
   new(NewStr);
   NewStr^.x := PStr^.x + Length(PStr^.str) + 1;
   NewStr^.y := PStr^.y;
   NewStr^.index := Stats.CurrentQuestion^.AnswerStart;
   NewStr^.str := '';
   NewStr^.prev := nil;
   NewStr^.next := nil;
   FirstStr := NewStr;
   PresStr := NewStr;
   i := Stats.CurrentQuestion^.AnswerStart;
   while QuestionBuffer[i] <> '>' do
      begin
      if QuestionBuffer[i] = '}' then
         begin
         i := i + 1;
         new(NewStr);
         NewStr^.x := LtMargin;
         NewStr^.y := PresStr^.y + 1;
         NewStr^.index := i;
         NewStr^.str := '';
         NewStr^.prev := PresStr;
         PresStr^.next := NewStr;
         NewStr^.next := nil;
         PresStr := NewStr;
         end
      else
         begin
         PresStr^.str := PresStr^.str + QuestionBuffer[i];
         i := i + 1;
         end;
      end;
   end;

procedure QandAAnswer(var FirstStr  :  EditString;
                      var PresStr   :  EditString);

   var
```

```
      P    :   integer;
      NumStr  :   string;

   begin
   AbraEditor(FirstStr, PresStr^.x, PresStr^.y,
              MaxLineLen - LtMargin, MaxTextLines - 1,
              Length(PresStr^.str) + 1 + LtMargin,
              AnswerText, AnswerBackground);
   new(PresStr);
   PresStr^.x := 0;
   PresStr^.y := 0;
   PresStr^.str := '[';
   PresStr^.prev := nil;
   PresStr^.next := nil;
   P := 1;
   while FirstStr^.str[1] = ' ' do
      begin
      Delete(FirstStr^.str, 1, 1);
      P := P + 1;
      end;
   str(FirstStr^.x + P - 1 - LtMargin, NumStr);
   insert(NumStr, PresStr^.str, Length(PresStr^.str) + 1);
   insert(',',PresStr^.str, Length(PresStr^.str) + 1);
   str(FirstStr^.y - ScrTextLines, NumStr);
   insert(NumStr, PresStr^.str, Length(PresStr^.str) + 1);
   insert(']',PresStr^.str, Length(PresStr^.str) + 1);
   FirstStr^.str := PresStr^.str + FirstStr^.str;
   end;

procedure GetNewAnswer;

   var
      P, i, j     :  integer;
      AnswerType  :  char;
      FirstStr,
      PresStr     :  EditString;
      TSTring,
      NumStr      :  string;
      GNADone,
      SKey        :  boolean;
      KBKey       :  KeyType;
      ch          :  char;

   begin
   ModifiedQuestion := TRUE;
   TmpNode := nil;
   AnswerType := ' ';
   TString := '  Answer Type? (M)ultiple Choice, (Q)uestion/Answer, In (T)ext  ';
   for i := 1 to Length(TString) do
      FastWrite(LtMargin + i - 1, MaxTextLines - 1, TString[i], HelpText,
                HelpBackground);
   TextBackground(HelpBackground);
   TextColor(HelpText);
   GoToXY(LtMargin + Length(TString) + 1, MaxTextLines - 1);
   readln(AnswerType);
   for i := LtMargin to MaxLineLen - 1 do
      FastWrite(i, MaxTextLines - 1, ' ', HelpText, HelpBackground);
   if (not ((IntegratingQFile) and (UpCase(AnswerType) = 'Q'))) then
      Stats.CurrentQuestion^.AnswerStart := H.QBEnd;
   case UpCase(AnswerType) of
      'M' : begin
            TextStat := 23;
            DataStat := 5;
            TString := 'Choosing Multiple Choice Answer(s)';
            j := 40 - TRUNC(Length(TString)/2);
            for i := 2 to j do
```

```
         FastWrite(i, ScrTextLines, ' ', StatusText, StatusBackground);
      for i := 1 to Length(TString) do
         FastWrite(j + i - 1, ScrTextLines, TString[i], StatusText,
                   StatusBackground);
      for i := j + Length(TString) to MaxLineLen - 1 do
         FastWrite(i, ScrTextLines, ' ', StatusText, StatusBackground);
      LoadQuestion(FirstStr);
      GetMultipleChoiceAnswer(FirstStr);
      end;
'Q' : begin
      TextStat := 24;
      DataStat := 5;
      TString := 'Editing The Answer Plus An Explanation';
      j := 40 - TRUNC(Length(TString)/2);
      for i := 2 to j do
         FastWrite(i, ScrTextLines, ' ', StatusText, StatusBackground);
      for i := 1 to Length(TString) do
         FastWrite(j + i - 1, ScrTextLines, TString[i], StatusText,
                   StatusBackground);
      for i := j + Length(TString) to MaxLineLen - 1 do
         FastWrite(i, ScrTextLines, ' ', StatusText, StatusBackground);
      LoadQuestion(FirstStr);
      while FirstStr^.next <> nil do
         begin
         PresStr := FirstStr;
         FirstStr := FirstStr^.next;
         Dispose(PresStr);
         end;
      PresStr := FirstStr;
      FirstStr := nil;
      if IntegratingQFile then
         begin
         LoadQandAAnswer(FirstStr, PresStr);
         Stats.CurrentQuestion^.AnswerStart := H.QBEnd;
         end;
      QandAAnswer(FirstStr, PresStr);
      end;
'T' : begin
      TextStat := 22;
      DataStat := 5;
      TString := 'Choosing Word(s)/Phrase(s) From Text';
      j := 40 - TRUNC(Length(TString)/2);
      for i := 2 to j do
         FastWrite(i, ScrTextLines, ' ', StatusText, StatusBackground);
      for i := 1 to Length(TString) do
         FastWrite(j + i - 1, ScrTextLines, TString[i], StatusText,
                   StatusBackground);
      for i := j + Length(TString) to MaxLineLen - 1 do
         FastWrite(i, ScrTextLines, ' ', StatusText, StatusBackground);
      TString := 'F4 to complete highlighting   INSERT to ';
      TString := TString + 'start ';
      TString := TString + 'a marked section';
      j := 40 - TRUNC(Length(TString)/2);
      for i := 2 to j do
         FastWrite(i, MaxTextLines, ' ', StatusText, StatusBackground);
      for i := 1 to Length(TString) do
         FastWrite(j + i - 1, MaxTextLines, TString[i], StatusText,
                   StatusBackground);
      for i := j + Length(TString) to MaxLineLen - 1 do
         FastWrite(i, MaxTextLines, ' ', StatusText, StatusBackground);
      TmpNode := nil;
      GoToXY(Stats.LocX, Stats.LocY);
      GNADone := False;
      while (not GNADone) do
         begin
         ReDisplayPage := FALSE;
```

```
InKey(SKey, KBKey, ch);

case KBKey of
   UpArrow         :  ScrollUp_AS;
   DownArrow       :  ScrollDown_AS;
   LeftArrow       :  LastChar;
   RightArrow      :  NextChar;
   PgUp            :  PageUp;
   PgDn            :  PageDown;
   HomeKey         :  FirstPage;
   EndKey          :  LastPage;

   F1   :  begin
           KeyboardHelpQ;
           ReDisplayPage := True;
           end;
   F4   :  begin
           if (DataStat <> 3) then
              begin
              BeginNewTextAnswer;
              DataStat := 3;
              end
           else
              begin
              EndNewTextAnswer;
              DataStat := 4;
              GNADone := TRUE;
              end;
           if (TextStat = 22) then
              begin
              TString := 'F4 to complete highlighting   INSERT to ';
              if (DataStat = 3) then
                 TString := TString + 'finish '
              else
                 TString := TString + 'start ';
              TString := TString + 'a marked section';
              end;
           j := 40 - TRUNC(Length(TString)/2);
           for i := 2 to j do
              FastWrite(i, MaxTextLines, ' ', StatusText,
                        StatusBackground);
           for i := 1 to Length(TString) do
              FastWrite(j + i - 1, MaxTextLines, TString[i],
                        StatusText, StatusBackground);
           for i := j + Length(TString) to MaxLineLen - 1 do
              FastWrite(i, MaxTextLines, ' ', StatusText,
                        StatusBackground);
           end;
   InsertKey : if (TextStat = 22) then
                  if DataStat = 3 then
                     begin
                     TmpNewNode^.EndRange := Stats.CurrentPos - 1;
                     DataStat := 4;
                     end
                  else
                     begin
                     BeginNewTextAnswer;
                     DataStat := 3;
                     end;
   Esc  :  begin
           if TextStat = 22 then
              begin
              while TmpNode <> nil do
                 begin
                 TmpNewNode := TmpNode;
```

```
                              TmpNode := TmpNode^.NextPartner;
                              Dispose(TmpNewNode);
                              end;
                           ReDisplayPage := TRUE;
                           end;
                        end;
                  end;
               if (not GNADone) and (ReDisplayPage) then
                  DisplayPage
               else
                  GoToXY(Stats.LocX, Stats.LocY);
               end;
            new(PresStr);
            PresStr^.x := 0;
            PresStr^.y := 0;
            PresStr^.str := '<';
            PresStr^.prev := nil;
            PresStr^.next := nil;
            while TmpNode <> nil do
               begin
               TmpNewNode := TmpNode;
               str(TmpNode^.BeginRange, NumStr);
               insert(NumStr, PresStr^.str, Length(PresStr^.str) + 1);
               insert(',',PresStr^.str, Length(PresStr^.str) + 1);
               str(TmpNode^.EndRange, NumStr);
               insert(NumStr, PresStr^.str, Length(PresStr^.str) + 1);
               TmpNode := TmpNode^.NextPartner;
               Dispose(TmpNewNode);
               if TmpNode <> nil then
                  insert(',',PresStr^.str, Length(PresStr^.str) + 1);
               end;
            insert('>>',PresStr^.str, Length(PresStr^.str) + 1);
            FirstStr := PresStr;
            end;
      end;
   StoreNewAnswer(FirstStr);
   DisplayQuestion;
   DisplayAnswer;
   end;

procedure GetMultipleChoiceAnswer(var FirstStr  :  EditString);

   var
      I, J, X, Y :  integer;
      Split,
      NumStr : string;
      IsSpecial,
      MCDone  :  boolean;
      LastSpace : integer;
      P  : integer;
      TheKey  :  KeyType;
      LinkStr,
      NewStr,
      PresStr,
      TmpStr  :  EditString;
      TString :  string;
      Ch      :  char;

   begin
   BottomBackground;
   StatusLine;
   P := 0;
   PresStr := FirstStr;
   TmpStr := FirstStr;
   while TmpStr <> nil do
      begin
```

```
   GoToXY(TmpStr^.x, TmpStr^.y);
   write(TmpStr^.str);
   TmpStr := TmpStr^.next;
   end;

MCDone := False;
while not MCDone do
   begin
   GoToXY(PresStr^.x + P, PresStr^.y);

   InKey(IsSpecial, TheKey, Ch);

   case TheKey of
      F1  :  begin
             end;
      F4  :  if (DataStat <> 3) then
                begin
                DataStat := 3;
                BeginNewAnswer(FirstStr, PresStr, P);
                end
             else
                begin
                EndNewAnswer(FirstStr, PresStr, P);
                while FirstStr <> nil do
                   begin
                   PresStr := FirstStr;
                   FirstStr := FirstStr^.next;
                   dispose(PresStr);
                   end;
                new(NewStr);
                NewStr^.x := 0;
                NewStr^.y := 0;
                NewStr^.str := '[';
                NewStr^.prev := nil;
                NewStr^.next := nil;
                FirstStr := NewStr;
                while TmpNode <> nil do
                   begin
                   TmpNewNode := TmpNode;
                   str(TmpNode^.BeginRange, NumStr);
                   insert(NumStr, FirstStr^.str, Length(FirstStr^.str) + 1);
                   insert(',',FirstStr^.str, Length(FirstStr^.str) + 1);
                   str(TmpNode^.EndRange, NumStr);
                   insert(NumStr, FirstStr^.str, Length(FirstStr^.str) + 1);
                   TmpNode := TmpNode^.NextPartner;
                   Dispose(TmpNewNode);
                   if TmpNode <> nil then
                      insert(',',FirstStr^.str, Length(FirstStr^.str) + 1);
                   end;
                insert(']',FirstStr^.str, Length(FirstStr^.str) + 1);
                MCDone := true;
                end;
      LeftArrow :
         begin
         if (TextStat = 23) and (DataStat = 3) then
            begin
            TextBackground(HelpBackground);
            TextColor(HelpText);
            GoToXY(PresStr^.x + P, PresStr^.y);
            if QuestionBuffer[PresStr^.index + P] <> '}' then
               write(QuestionBuffer[PresStr^.index + P]);
            end;
         if P > 1 then
            P := P - 1
         else
            begin
```

```
            PresStr := PresStr^.prev;
            P := Length(PresStr^.str) + 1;
            end;
         GoToXY(PresStr^.x + P, PresStr^.y);
         end;

      RightArrow :
         begin
         if (TextStat = 23) and (DataStat = 3) then
            begin
            TextBackground(AnswerBackground);
            TextColor(AnswerText);
            GoToXY(PresStr^.x + P, PresStr^.y);
            if QuestionBuffer[PresStr^.index + P] <> '}' then
               write(QuestionBuffer[PresStr^.index + P]);
            end;
         if (P < Length(PresStr^.str)) then
            P := P + 1
         else
            begin
            PresStr := PresStr^.next;
            P := 0;
            end;
         GoToXY(PresStr^.x + P, PresStr^.y);
         end;

      InsertKey :
         begin
         if (TextStat = 23) then
            if DataStat = 3 then
               begin
               TmpNewNode^.EndRange := PresStr^.index + P - 1 - FirstStr^.index;
               DataStat := 4;
               end
            else
               begin
               BeginNewAnswer(FirstStr, PresStr, P);
               DataStat := 3;
               end;
         end;

      UpArrow    :
             if PresStr^.prev <> nil then
               begin
               PresStr := PresStr^.prev;
               if P > Length(PresStr^.str) then
                  P := Length(PresStr^.str) + 1;
               end;
      DownArrow  :
             if PresStr^.next <> nil then
               begin
               PresStr := PresStr^.next;
               if P > Length(PresStr^.str) then
                  P := Length(PresStr^.str) + 1;
               end;
      end;
   StatusLine;
   end;
end;

procedure StoreNewQuestion(var FirstStr  :  EditString);

   var
      i          :  integer;
      TmpString  :  EditString;
```

```
    begin
    ModifiedQuestion := TRUE;
    while FirstStr <> nil do
       begin
       if FirstStr^.str <> '' then
          begin
          FirstStr^.str := FirstStr^.str;
          for i := 1 to Length(FirstStr^.str) do
             begin
             QuestionBuffer[H.QBEnd] := FirstStr^.str[i];
             H.QBEnd := H.QBEnd + 1;
             end;
          end;
       if FirstStr^.next <> nil then
          QuestionBuffer[H.QBEnd] := '}'
       else
          QuestionBuffer[H.QBEnd] := '|';
       H.QBEnd := H.QBEnd + 1;
       TmpString := FirstStr;
       FirstStr := FirstStr^.next;
       dispose(TmpString);
       end;
    end;

procedure GetNewQuestion;

    var
       i, j  :  integer;
       FirstStr : EditString;
       NewQuestion  :  QuestionPtr;
       TString      :  string;

    begin
    ModifiedQuestion := TRUE;
    if TmpNode <> nil then
       begin
       Dispose(TmpNode);
       TmpNode := nil;
       end;
    ResetScrMatrix;
    DisplayPage;
    BottomBackground;

    new(NewQuestion);
    NewQuestion^.PrevQuestion := nil;
    NewQuestion^.NextQuestion := nil;
    NewQuestion^.QuestionStart := H.QBEnd;
    NewQuestion^.AnswerStart := H.QBEnd;
    if H.Questions = nil then
       begin
       Stats.CurrentQuestion := NewQuestion;
       H.Questions := NewQuestion;
       end
    else
       begin
       NewQuestion^.PrevQuestion := Stats.CurrentQuestion;
       if Stats.CurrentQuestion^.NextQuestion <> nil then
          begin
          NewQuestion^.NextQuestion := Stats.CurrentQuestion^.NextQuestion;
          NewQuestion^.NextQuestion^.PrevQuestion := NewQuestion;
          end;
       Stats.CurrentQuestion^.NextQuestion := NewQuestion;
       end;
    Stats.CurrentQuestion := NewQuestion;
    FirstStr := nil;
    BottomBackground;
```

```
   TString := 'Editing A New Question';
   j := 40 - TRUNC(Length(TString)/2);
   for i := 2 to j do
      FastWrite(i, ScrTextLines, ' ', StatusText, StatusBackground);
   for i := 1 to Length(TString) do
      FastWrite(j + i - 1, ScrTextLines, TString[i], StatusText,
                StatusBackground);
   for i := j + Length(TString) to MaxLineLen - 1 do
      FastWrite(i, ScrTextLines, ' ', StatusText, StatusBackground);
   AbraEditor(FirstStr, LtMargin, ScrTextLines + 1, MaxLineLen - LtMargin,
              MaxTextLines - 1, LtMargin, HelpText, HelpBackground);
   StoreNewQuestion(FirstStr);
   GetNewAnswer;
   end;

procedure EditQuestion;

   var
       FirstStr,
       PresStr    :  EditString;
       P          :  integer;
       NumStr     :  string;

   begin
   ModifiedQuestion := TRUE;
   LoadQuestion(FirstStr);
   Stats.CurrentQuestion^.QuestionStart := H.QBEnd;
   AbraEditor(FirstStr, FirstStr^.x, FirstStr^.y, MaxLineLen - LtMargin,
              MaxTextLines - 1, LtMargin, HelpText, HelpBackground);
   StoreNewQuestion(FirstStr);
   GetNewAnswer;
   end;

procedure NextQuestion;

   begin
   if TmpNode <> nil then
      begin
      Dispose(TmpNode);
      TmpNode := nil;
      end;
   ResetScrMatrix;
   DisplayPage;
   BottomBackground;

   if ((Stats.CurrentQuestion <> nil) and
      (Stats.CurrentQuestion^.NextQuestion <> nil)) or
      (IntegratingQFile) then
      begin
      if Stats.CurrentQuestion^.NextQuestion = nil then
         IntegratingQFile := False
      else
         Stats.CurrentQuestion:= Stats.CurrentQuestion^.NextQuestion;
      DisplayQuestion;
      if IntegratingQFile then
         GetNewAnswer;
      end
   else
      GetNewQuestion;
   StatusLine;
   end;

procedure PrevQuestion;

   begin
   if TmpNode <> nil then
```

```
      begin
      Dispose(TmpNode);
      TmpNode := nil;
      end;
   UpdateScrmatrix;
   DisplayPage;
   BottomBackground;
   if (Stats.CurrentQuestion <> nil) and
      (Stats.CurrentQuestion^.PrevQuestion <> nil) then
      begin
      Stats.CurrentQuestion := Stats.CurrentQuestion^.PrevQuestion;
      DisplayQuestion;
      end;
   StatusLine;
   end;

procedure NewLevelQuestions;

   var
      ALine,
      tempstr       :  string;
      i, j,
      BIndex        :  integer;
      BeforeQuestion,
      NowQuestion,
      NewQuestion   :  QuestionPtr;
      WasAnswered,
      AnewQuestion  :  boolean;

   begin
   IntegratingQFile := TRUE;
   if H.Questions <> nil then
      KillQuestionList;
   H.Questions := nil;
   H.LevelName := Files.CurrLevel^.LevName;
   BIndex := 1;

   DisplayDirectory('*.*');
   BottomBackground;
   TextBackground(HelpBackground);
   TextColor(HelpText);
   GoToXY(LtMargin, ScrTextLines + 2);
   if Files.CurrLevel^.LevName = '' then
      begin
      write('Please write the name for this level.  ');
      readln(Files.CurrLevel^.LevName);
      end;
   write('Which file contains the questions for this level?  ');
   readln(tempstr);
   Assign(QuestionFile, tempstr);
   Reset(QuestionFile);

   AnewQuestion := True;
   WHILE NOT EOF(QuestionFile) DO
      BEGIN
      ReadLn(QuestionFile, ALine);
      if AnewQuestion then
         begin
         AnewQuestion := False;
         WasAnswered := False;
         new(NewQuestion);
         NewQuestion^.PrevQuestion := nil;
         NewQuestion^.NextQuestion := nil;
         NewQuestion^.QuestionStart := BIndex;
         if H.Questions = nil then
            begin
```

```
            NowQuestion := NewQuestion;
            H.Questions := NewQuestion;
            end
         else
            begin
            BeforeQuestion := NowQuestion;
            NowQuestion := NewQuestion;
            BeforeQuestion^.NextQuestion := NowQuestion;
            NowQuestion^.PrevQuestion := BeforeQuestion;
            end;
         end
      else if ALine = '' then
         begin
         AnewQuestion := True;
         if WasAnswered then
            QuestionBuffer[BIndex - 1] := '>'
         else
            QuestionBuffer[BIndex - 1] := '|';
         end;

      if Length(ALine) > 0 then
         begin
         for j := 1 to Length(ALine) do
            begin
            QuestionBuffer[BIndex] := ALine[j];
            if QuestionBuffer[BIndex] = '|' then
               begin
               NowQuestion^.AnswerStart := BIndex + 1;
               WasAnswered := True;
               end;
            BIndex := BIndex + 1;
            end;
         QuestionBuffer[BIndex] := '}';
         BIndex := BIndex + 1;
         end;
      end;

   if WasAnswered then
      QuestionBuffer[BIndex - 1] := '>'
   else
      QuestionBuffer[BIndex - 1] := '|';

   H.QBEnd := BIndex;
   Close(QuestionFile);
   Stats.CurrentQuestion := H.Questions;
   ScrTextLines := InterlinearLines;
   if Stats.LastLine - Stats.FirstLine <> ScrTextLines - 2 then
      AdjustScrTextLines;
   if TmpNode <> nil then
      begin
      Dispose(TmpNode);
      TmpNode := nil;
      end;
   UpdateScrMatrix;
   DisplayPage;
   BottomBackground;
   DisplayQuestion;
   GetNewAnswer;
   END;

procedure DeleteQuestion;

   var
      TmpQuestion    :    QuestionPtr;

   begin
```

```
   ModifiedQuestion := TRUE;
   BottomBackground;
   TmpQuestion := Stats.CurrentQuestion;
   if TmpQuestion <> nil then
      begin
      if TmpQuestion^.PrevQuestion = H.Questions then
         begin
         H.Questions := TmpQuestion^.NextQuestion;
         TmpQuestion^.NextQuestion^.PrevQuestion := nil;
         if TmpQuestion^.NextQuestion <> nil then
            Stats.CurrentQuestion := TmpQuestion^.NextQuestion
         else
            Stats.CurrentQuestion := H.Questions;
         end
      else
         begin
         TmpQuestion^.PrevQuestion^.NextQuestion := TmpQuestion^.NextQuestion;
         TmpQuestion^.NextQuestion^.PrevQuestion := TmpQuestion^.PrevQuestion;
         if TmpQuestion^.NextQuestion <> nil then
            Stats.CurrentQuestion := TmpQuestion^.NextQuestion
         else
            Stats.CurrentQuestion := TmpQuestion^.PrevQuestion;
         end;
      end;
   Dispose(TmpQuestion);
   end;

procedure FirstQuestion;

   var
      TString    :   string;
      i          :   integer;

   begin
   TextStat := 21;
   if ((H.Questions = nil) and (Files.CurrLevel^.QuesName <> '')) then
      begin
      GetLevelQuestions;
      Stats.CurrentQuestion := H.Questions;
      end;
   if Stats.CurrentQuestion <> nil then
      begin
      ScrTextLines := InterlinearLines;
      DataStat := 0;
      DisplayQuestion;
      end
   else if (H.Questions = nil) and (WithCursor) then
      begin
      DataStat := 5;
      BottomBackground;
      TString := 'There are presently no questions present.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 1, TString[i],
                   HelpText, HelpBackground);
      TString := 'Please press TAB to edit First Question.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 2, TString[i],
                   HelpText, HelpBackground);
      TString := 'Questions are displayed within this lower block.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 4, TString[i],
                   HelpText, HelpBackground);
      end
   else if (H.Questions = nil) and (not WithCursor) then
      begin
      if Files.CurrLevel^.NextLev <> nil then
```

```
         NextHelpLevel
      else
         TextStat := 40;
      end;
   end;

end.
```

Appendix C07

ABRAINTR.PAS

```
{*************************************************************************}
{ PARENT PROGRAM:    Advanced Bilingual Reading Assistant                 }
{                    Advanced Bilingual Reading Assistant - Authoring System }
{ PROGRAMMER:        Raymundo A. Q. Marcelo                               }
{ DATE:              23-April-1995                                        }
{                                                                         }
{ UNIT NAME:         ABRAINTR                                             }
{                                                                         }
{ PURPOSE:           This unit contains the functions and procedures required}
{                    by the ABRA and ABRA-AS programs to display the      }
{                    introduction object.                                 }
{*************************************************************************}

unit ABRAINTR;

interface

uses Dos, Crt, ABRAVARS, TP_REF, ABRABACK, ABRAED;

procedure InitIntro;
procedure GetIntro;
procedure LoadIntro(var FirstStr    :  EditString);
procedure SaveIntro(var FirstStr  :  EditString);
procedure EditIntro;

implementation

procedure InitIntro;

   const
      BufSize = 16384;

   var
      ALine,
      tempstr    :    string;
      i, pi, j   :    integer;
      BIndex     :    integer;
      Buf        :    array[1..BufSize] of byte;

   begin
   Assign(TextFile, Files.Intro^.Fname);
   Reset(TextFile);
   SetTextBuf(TextFile, Buf);
   pi := 1;
   i := 1;

   WHILE not EOF(TextFile) DO
      BEGIN
      {Place text in the TextBuffer}
      Readln(TextFile, ALine);
      TextPage[pi] := i;
      for j := 1 to Length(ALine) do
         begin
         if ((i >= Page * MaxPageSize) and
             (i <= (Page * MaxPageSize) + MaxTextSize)) then
            TextBuffer[i] := ALine[j];
         i := i + 1;
         end;
      if ALine[Length(ALine)] <> ' ' then
         begin
         if ((i >= Page * MaxPageSize) and
             (i <= (Page * MaxPageSize) + MaxTextSize)) then
            TextBuffer[i] := ' ';
         i := i + 1;
         end;
      pi := pi + 1;
```

```
      end;

   Close(TextFile);
   TextPage[pi] := i;
   TotalChar := i - 1;
   TotalLines := pi - 1;
   END;


procedure GetIntro;

   var
      OldLName,
      ALine,
      tempstr    :    string;
      i, j, pi   :    integer;

   begin
   if H.HelpNodeList <> nil then
      begin
      KillHelpNodeList;
      H.HelpNodeList := nil;
      end;

   if Files.Intro^.Lname = '' then
      H.LevelName := '  Introduction  '
   else
      H.LevelName := Files.Intro^.Lname;

   Assign(IntroFile, Files.Intro^.Fname);
   Reset(IntroFile);
   pi := 1;
   i := 1;
   while (not EOF(IntroFile)) do
      begin
      ReadLn(IntroFile, ALine);
      TextPage[pi] := i;
      for j := 1 to Length(ALine) do
         begin
         TextBuffer[i] := ALine[j];
         i := i + 1;
         end;
      pi := pi + 1;
      end;

   close(IntroFile);
   TextPage[pi] := i;
   TotalChar := i - 1;
   TotalLines := pi - 1;
   end;

procedure LoadIntro(var FirstStr    :  EditString);

   var
      NewStr,
      PresStr :  EditString;
      pi, i   :  integer;

   begin
   Assign(IntroFile, Files.Intro^.Fname);
   Reset(IntroFile);
   pi := 2;
   i := 1;
   FirstStr := nil;

   WHILE (not EOF(IntroFile)) DO
```

```
      BEGIN
      new(NewStr);
      NewStr^.x := LtMargin;
      NewStr^.y := pi;
      NewStr^.index := i;
      NewStr^.next := nil;
      if FirstStr = nil then
         begin
         FirstStr := NewStr;
         NewStr^.prev := nil;
         end
      else
         begin
         NewStr^.prev := PresStr;
         PresStr^.next := NewStr;
         end;
      PresStr := NewStr;
      Readln(IntroFile, PresStr^.str);
      i := i + Length(PresStr^.str) + 1;
      pi := pi + 1;
      end;

   Close(IntroFile);
   end;

procedure SaveIntro(var FirstStr  :  EditString);

   var
      PresStr  :  EditString;

   begin
   Rewrite(IntroFile);
   while FirstStr <> nil do
      begin
      writeln(IntroFile, FirstStr^.str);
      PresStr := FirstStr;
      FirstStr := FirstStr^.next;
      dispose(PresStr);
      end;
   close(IntroFile);
   end;

procedure EditIntro;

   var
      FirstStr   :  EditString;
      TempStr    :  string;

   begin
   TempStr := H.LevelName;
   if Files.Intro^.Fname <> '' then
      begin
      H.LevelName := Files.Intro^.Lname;
      LoadIntro(FirstStr);
      end
   else
      H.LevelName := '  Introduction  ';
   TopBackground;
   BottomBackground;
   AbraEditor(FirstStr, LtMargin, 2, MaxLineLen - LtMargin, ScrTextLines,
              LtMargin, NormalText, NormalBackground);
   SaveIntro(FirstStr);
   H.LevelName := TempStr;
   end;

end.
```

Appendix C08

ABRALEV.PAS

```
{************************************************************************}
{ PARENT PROGRAM:   Advanced Bilingual Reading Assistant                  }
{                   Advanced Bilingual Reading Assistant - Authoring System }
{ PROGRAMMER:       Raymundo A. Q. Marcelo                                }
{ DATE:             23-April-1995                                         }
{                                                                         }
{ UNIT NAME:        ABRALEV                                               }
{                                                                         }
{ PURPOSE:          The level object contains the information pertaining to }
{                   storage and retrieval of each level used by the ABRA and}
{                   ABRA-AS programs.                                     }
{************************************************************************}

unit ABRALEV;

interface

uses Dos, Crt, TP_REF, ABRAVARS, ABRABACK, ABRAED, ABRATEXT;

procedure InitLevel;
procedure LoadFile;
procedure GetLevel;
procedure SaveLevel;
procedure PrevHelpLevel;
procedure NextHelpLevel;

implementation

procedure GetLevel;

   var
      ALine,
      tempstr    :   string;
      i, j,
      StartQuestion,
      HelpLevel,
      ErrCode,
      B_rng,
      E_rng,
      BIndex         :   integer;
      TmpDisplay,
      NewNode,
      BeforeNode,
      NowNode,
      BeforePart,
      NowPart        :  HelpNodePtr;
      NewQuestion,
      BeforeQuestion,
      NowQuestion    :  QuestionPtr;
      Done,
      NewGroup       :  Boolean;
      {Buf             :   array[1..BufSize] of byte;}
   begin
   H.HelpNodeList := nil;
   H.DisplayList := nil;
   H.LevelName := '  '+Files.CurrLevel^.LevName+'  ';

   Assign(LevelFile, Files.CurrLevel^.HWName);
   Reset(LevelFile);

   TmpDisplay := nil;
   BeforeNode := nil;
   NowNode := nil;
   BeforePart := nil;
   NowPart := nil;
   BIndex := 1;
```

```
                  TmpDisplay := TmpDisplay^.NextDisplay
               else
                  Done := TRUE;
            if (TmpDisplay^.NextDisplay = nil) then
               NewNode^.NextDisplay := nil
            else
               NewNode^.NextDisplay := TmpDisplay^.NextDisplay;
            TmpDisplay^.NextDisplay := NewNode;
            NewNode^.PrevDisplay := TmpDisplay;
            end;
      end;

   i := i + 1;
   for j := i to Length(ALine) do
      begin
      StringBuffer[BIndex] := ALine[j];
      BIndex := BIndex + 1;
      end;
   end;

H.BufferEnd := BIndex;
close(LevelFile);
end;

procedure SaveLevel;

   var
      TNode,
      HotWord :  HelpNodePtr;
      E       :  ExtStr;
      TmpStr,
      LS      :  string;
      i       :  integer;

   begin
   if ModifiedLevel then
      begin
      HotWord := H.HelpNodeList;
      if Files.CurrLevel^.HWName = '' then
         begin
         Modified := TRUE;
         Str(Level, LS);
         if Level < 10 then
            E := '.$0' + LS
         else
            E := '.$' + LS;
         Files.CurrLevel^.HWName := D + N + E;
         end;
      Assign(LevelFile, Files.CurrLevel^.HWName);
      {$I-} Rewrite(LevelFile); {$I+}

      HotWord := H.HelpNodeList;
      while HotWord <> nil do
         begin
         TNode := HotWord;
         HotWord := HotWord^.NextNode;
         write(LevelFile, '<<');
         i := TNode^.Index;
         while TNode^.NextPartner <> nil do
            begin
            write(LevelFile, TNode^.BeginRange, ',',TNode^.EndRange, ',');
            TNode := TNode^.NextPartner;
            end;
         write(LevelFile, TNode^.BeginRange, ',',TNode^.EndRange, '>');
         TmpStr := '';
         while (StringBuffer[i] <> '>') do
```

```
            begin
            insert(StringBuffer[i], TmpStr, Length(TmpStr) + 1);
            i := i + 1;
            end;
         insert(StringBuffer[i], TmpStr, Length(TmpStr) + 1);
         writeln(LevelFile, TmpStr);
         end;
      close(LevelFile);
      end;
   ModifiedLevel := False;
   end;

procedure LoadFile;

   var
      P0, P1: PathStr;
      E: ExtStr;
      Fname,
      temp     :    string;
      NewFile  :  FilesPtr;
      NewLevelFile  :  LevelPtr;
      BarPos,
      NameBeg,
      NameEnd  :  integer;

   begin
   if (not WithCursor) then
      Cursor_Small;
   Files.Intro := nil;
   Files.Article := nil;
   Files.Levels := nil;
   Files.CurrLevel := nil;
   DisplayDirectory('*.TXT');
   BottomBackground;
   TextBackground(HelpBackground);
   TextColor(HelpText);
   GoToXY(20, ScrTextLines + 3);
   if (WithCursor) then
      write('Which file would you like to highlight?  ')
   else
      write('Which file would you like to read?  ');
   readln(P0);
   if (not WithCursor) then
      Cursor_Off;
   FSplit(P0, D, N, E);
   Assign(TextFile, P0);
   {$I-} Reset(TextFile); {$I+}
   if IOResult = 0 then
      begin
      E := '.FLS';
      P1 := D + N + E;
      Assign(FileFile, P1);
      {$I-} Reset(FileFile); {$I+}
      if IOResult = 0 then
         begin
         {   ***   Group File EXISTS  ***   }
         while not EOF(FileFile) do
            begin
            readln(FileFile, temp);
            if (temp[1] = '@') and (UpCase(temp[2]) = 'I') then
               begin
               if (not WithCursor) then
                  begin
                  TextStat := 30;
                  DataStat := 0;
                  end;
```

```
            new(NewFile);
            NewFile^.NextFile := nil;
            NewFile^.Lname := copy(temp, 4, Length(temp));
            readln(FileFile, NewFile^.Fname);
            Files.Intro := NewFile;
            end
         else if (temp[1] = '@') and (UpCase(temp[2]) = 'T') then
            begin
            new(NewFile);
            NewFile^.NextFile := nil;
            readln(FileFile, NewFile^.Fname);
            NewFile^.Lname := '';
            Files.Article := NewFile;
            end
         else if (temp[1] = '@') and (UpCase(temp[2]) = 'H') then
            begin
            if (WithCursor) then
               begin
               TextStat := 10;
               DataStat := 1;
               end;
            new(NewLevelFile);
            NewLevelFile^.NextLev := nil;
            if UpCase(temp[4]) = 'I' then
               NewLevelFile^.Interlinear := true
            else
               NewLevelFile^.Interlinear := false;
            if Length(temp) > 6 then
               NewLevelFile^.LevName := copy(temp, 6, Length(temp))
            else
               NewLevelFile^.LevName := '';
            readln(FileFile, temp);
            BarPos := 1;
            while (temp[BarPos] <> '|') do
               BarPos := BarPos + 1;
            if BarPos > 1 then
               NewLevelFile^.HWName := copy(temp, 1, BarPos - 1)
            else
               NewLevelFile^.HWName := '';
            if BarPos < Length(temp) then
               NewLevelFile^.QuesName := copy(temp, BarPos + 1, Length(temp))
            else
               NewLevelFile^.QuesName := '';

            if Files.Levels = nil then
               begin
               NewLevelFile^.PrevLev := nil;
               Files.Levels := NewLevelFile;
               end
            else
               begin
               NewLevelFile^.PrevLev := Files.CurrLevel;
               Files.CurrLevel^.NextLev := NewLevelFile;
               end;
            Files.CurrLevel := NewLevelFile;
            end;
         end;
      Files.CurrLevel := Files.Levels;
      close(FileFile);
      end
   else
      begin
      {   ***   Group File DOES NOT EXIST  ***   }
      TextStat := 10;
      DataStat := -1;
      new(NewFile);
```

```
            NewFile^.NextFile := nil;
            NewFile^.Lname := '';
            NewFile^.Fname := P0;
            Files.Article := NewFile;
            Files.Intro := nil;
            Files.CurrLevel := nil;
            Files.Levels := nil;
            end
         end
      else
         begin
         GoToXY(30, ScrTextLines + 3);
         writeln('This File Does NOT EXIST');
         TextStat := 40;
         end;
      end;

procedure InitLevel;

      begin
      ArticleEnd := FALSE;
      ModifiedLevel := FALSE;
      ModifiedQuestion := FALSE;
      Stats.CurrentHotWord := H.HelpNodeList;
      Stats.CurrentQuestion := H.Questions;
      Stats.FirstHotWord := H.HelpNodeList;
      Stats.LastHotWord := nil;
      Stats.FirstDisplay := H.DisplayList;
      Stats.LastDisplay := nil;
      TmpCurrent := nil;
      end;

procedure PrevHelpLevel;

      var
         i        :  integer;
         ch       :  char;
         TString  :  string;
         TNode    :  HelpNodePtr;

      begin
      if Page <> 0 then
         begin
         Page := 0;
         GetText;
         end;
      BottomBackground;
      Modified := False;
      IntegratingFile := False;
      IntegratingQFile := False;
      while TmpNewNode <> nil do
         begin
         TNode := TmpNewNode;
         TmpNewNode := TmpNewNode^.NextPartner;
         Dispose(TNode);
         end;
      if Files.CurrLevel^.PrevLev <> nil then
         begin
         if H.HelpNodeList <> nil then
            KillHelpNodeList;
         if H.Questions <> nil then
            KillQuestionList;
         Level := Level - 1;
         Files.CurrLevel := Files.CurrLevel^.PrevLev;
         if Files.CurrLevel^.HWName = '' then
            begin
```

```
      BeginNewLevel := TRUE;
      if Files.CurrLevel^.LevName = '' then
         begin
         write('Please write the name for this level.  ');
         readln(Files.CurrLevel^.LevName);
         H.LevelName := '  '+Files.CurrLevel^.LevName+'  ';
         end;
      write('Does displaying your definitions require more than 6 lines? (Y/N) ');
      readln(ch);
      if UpCase(ch) = 'Y' then
         begin
         Files.CurrLevel^.Interlinear := TRUE;
         ScrTextLines := InterlinearLines;
         end
      else
         begin
         Files.CurrLevel^.Interlinear := FALSE;
         ScrTextLines := NormTextLines;
         end;

      BottomBackground;
      TString := 'There are presently no highlighted words present.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 1, TString[i],
                   HelpText, HelpBackground);
      TString := 'Please scroll to the word(s) to be highlighted.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 2, TString[i],
                   HelpText, HelpBackground);
      TString := 'Mark and Unmark the word(s) with F4.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 3, TString[i],
                   HelpText, HelpBackground);
      TString := 'Type in the definition within this lower block.';
      for i := 1 to Length(TString) do
         FastWrite(LtMargin + i, ScrTextLines + 4, TString[i],
                   HelpText, HelpBackground);
      end
   else
      begin
      BeginNewLevel := FALSE;
      GetLevel;
      end;
   if Files.CurrLevel^.Interlinear then
      ScrTextLines := InterlinearLines
   else
      ScrTextLines := NormTextLines;
   InitLevel;
   TextStat := 10;
   if H.HelpNodeList <> nil then
      DataStat := 1
   else
      DataStat := 0;
   FirstPage;
   ReDisplayPage := TRUE;
   end;
end;

procedure NextHelpLevel;

   var
      NewLevelFile  :  LevelPtr;
      E             :  ExtStr;
      TString,
      LS            :  string;
      ch            :  char;
```

```
   i              :  integer;
   TNode          :  HelpNodePtr;

begin
if Page <> 0 then
   begin
   Page := 0;
   GetText;
   end;
BottomBackground;
Modified := False;
IntegratingFile := False;
IntegratingQFile := False;
while TmpNewNode <> nil do
   begin
   TNode := TmpNewNode;
   TmpNewNode := TmpNewNode^.NextPartner;
   if TNode <> nil then
      Dispose(TNode);
   end;
Level := Level + 1;
if H.HelpNodeList <> nil then
   begin
   KillHelpNodeList;
   ResetScrMatrix;
   FirstPage;
   DisplayPage;
   end;
if H.Questions <> nil then
   KillQuestionList;
if Files.CurrLevel^.NextLev <> nil then
   begin
   BeginNewLevel := FALSE;
   Files.CurrLevel := Files.CurrLevel^.NextLev;
   GetLevel;
   if Files.CurrLevel^.Interlinear then
      ScrTextLines := InterlinearLines
   else
      ScrTextLines := NormTextLines;
   end
else
   begin
   BeginNewLevel := TRUE;
   new(NewLevelFile);
   NewLevelFile^.NextLev := nil;
   NewLevelFile^.LevName := '';
   NewLevelFile^.HWName := '';
   NewLevelFile^.QuesName := '';
   Files.CurrLevel^.NextLev := NewLevelFile;
   NewLevelFile^.PrevLev := Files.CurrLevel;
   Files.CurrLevel := NewLevelFile;
   BottomBackground;
   TString := 'There are presently no highlighted word(s) present.';
   for i := 1 to Length(TString) do
      FastWrite(LtMargin+i,ScrTextLines+1,TString[i],HelpText,HelpBackground);
   TextBackground(HelpBackground);
   TextColor(HelpText);
   GoToXY(LtMargin, ScrTextLines + 2);
   if Files.CurrLevel^.LevName = '' then
      begin
      write('Please write the name for this level.  ');
      readln(Files.CurrLevel^.LevName);
      if Files.CurrLevel^.LevName = '' then
         H.LevelName := ''
      else
         H.LevelName := '  '+Files.CurrLevel^.LevName+'  ';
```

```
      end;
   GoToXY(LtMargin, ScrTextLines + 3);
   write('Does displaying your definitions require more than 6 lines? (Y/N) ');
   readln(ch);
   if UpCase(ch) = 'Y' then
      begin
      Files.CurrLevel^.Interlinear := TRUE;
      ScrTextLines := InterlinearLines;
      end
   else
      begin
      Files.CurrLevel^.Interlinear := FALSE;
      ScrTextLines := NormTextLines;
      end;

   BottomBackground;
   TString := 'There are presently no highlighted words present.';
   for i := 1 to Length(TString) do
      FastWrite(LtMargin + i, ScrTextLines + 1, TString[i],
                HelpText, HelpBackground);
   TString := 'Please scroll to the word(s) to be highlighted.';
   for i := 1 to Length(TString) do
      FastWrite(LtMargin + i, ScrTextLines + 2, TString[i],
                HelpText, HelpBackground);
   TString := 'Mark and Unmark the word(s) with F4.';
   for i := 1 to Length(TString) do
      FastWrite(LtMargin + i, ScrTextLines + 3, TString[i],
                HelpText, HelpBackground);
   TString := 'Type in the definition within this lower block.';
   for i := 1 to Length(TString) do
      FastWrite(LtMargin + i, ScrTextLines + 4, TString[i],
                HelpText, HelpBackground);
   end;

InitLevel;
TextStat := 10;
if H.HelpNodeList <> nil then
   DataStat := 1
else
   DataStat := 0;
FirstPage;
ReDisplayPage := TRUE;
end;

end.
```

Appendix C09

ABRAED.PAS

```
{*************************************************************************}
{ PARENT PROGRAM:    Advanced Bilingual Reading Assistant                 }
{                    Advanced Bilingual Reading Assistant - Authoring System }
{ PROGRAMMER:        Raymundo A. Q. Marcelo                               }
{ DATE:              23-April-1995                                        }
{                                                                         }
{ UNIT NAME:         ABRAED                                               }
{                                                                         }
{ PURPOSE:           This unit contains the basic editor functions and    }
{                    procedures used by the ABRA and ABRA-AS programs for }
{                    displaying information on the screen.                }
{*************************************************************************}

unit ABRAED;

interface

uses Dos, Crt, ABRAVARS, TP_REF, ABRABACK;

procedure DisplayDirectory(FileTemplate  :  string);

procedure InputString(var S : string;
                      var P : integer;
                      var Split : string;
                      var DoConcat : boolean;
                      var InsertOn : boolean;
                      var LastKey  : KeyType;
                      MaxLength,
                      X, Y :  integer;
                      FGround, BGround :  integer);
procedure AbraEditor(var FirstStr      :  EditString;
                     x1, y1, x2, y2,
                     FstX              :  integer;
                     FGround, BGround  :  integer{;
                     Scrollable        :  boolean});

implementation

procedure InputString(var S : string;
                      var P : integer;
                      var Split : string;
                      var DoConcat : boolean;
                      var InsertOn : boolean;
                      var LastKey  : KeyType;
                      MaxLength,
                      X, Y :  integer;
                      FGround, BGround :  integer);

   var
      I, J, P1 :  integer;
      Ch : char;
      IsSpecial :  boolean;
      TheKey :  KeyType;
      LastSpace : integer;
      LastPos   : integer;

   procedure SetString;
      var
         I  :  integer;
      begin
         I := Length(S);
         while S[I] = ' ' do
            I := I - 1;
         S[0] := char(I);
      end;
```

```
begin
if S <> '' then
   LastPos := Length(S) + 1
else
   LastPos := 2;
Split := '';
DoConcat := FALSE;
J := Length(S) + 1;
for I := J to MaxLength do
   S[I] := ' ';
S[0] := char(MaxLength);

for I := 1 to MaxLength do
   FastWrite(X+I-1, Y, S[I], FGround, BGround);
P := P + 1;
InsertOn := True;
LastSpace := 0;

repeat
   GoToXY(X + P - 1, Y);

   InKey(IsSpecial, TheKey, Ch);

   case TheKey of
      NumberKey, TextKey, Space :
         begin
         if (LastPos < MaxLength) and (P < MaxLength) then
            begin
            if (TheKey = Space) and (P < MaxLength) then
               LastSpace := P;
            if InsertOn then
               Insert(Ch, S, P)
            else
               begin
               Delete(S, P, 1);
               Insert(Ch, S, P);
               end;
            if P < MaxLength then
               begin
               P := P + 1;
               LastPos := LastPos + 1;
               end;
            for I := 1 to MaxLength do
               FastWrite(X+I-1, Y, S[I], FGround, BGround);
            end
         else if LastPos = MaxLength then
            begin
            if (InsertON) or (P = LastPos) then
               begin
               P1 := MaxLength - LastSpace;
               Split := copy(S, LastSpace, P1);
               Split := Split + ch;
               Delete(S, LastSpace, P1);
               LastPos := LastSpace;
               if LastPos < P then
                  P := -1 * (P - LastPos + 1);
               for i := 1 to P1 do
                  S := S + ' ';
               for I := 1 to MaxLength do
                  FastWrite(X+I-1, Y, S[I], FGround, BGround);
               LastKey := TheKey;
               SetString;
               Exit;
               end
            else
               begin
```

```
            Delete(S, P, 1);
            Insert(Ch, S, P);
            if P < MaxLength then
               begin
               P := P + 1;
               LastPos := LastPos + 1;
               end;
            for I := 1 to MaxLength do
               FastWrite(X+I-1, Y, S[I], FGround, BGround);
            end;
         end;
      end;
   Bksp :
      if P > 1 then
         begin
         P := P - 1;
         LastPos := LastPos - 1;
         Delete(S, P, 1);
         S := S + ' ';
         for I := 1 to MaxLength do
            FastWrite(X+I-1, Y, S[I], FGround, BGround);
         Ch := ' ';
         end
      else
         begin
         SetString;
         DoConcat := TRUE;
         P := 0;
         LastKey := TheKey;
         Exit;
         end;

   LeftArrow :
      if P > 1 then
         begin
         P := P - 1;
         for I := 1 to MaxLength do
            FastWrite(X+I-1, Y, S[I], FGround, BGround);
         end
      else
         begin
         LastKey := TheKey;
         SetString;
         Exit;
         end;

   RightArrow :
      if (P <= LastPos) and
         (P <= MaxLength) then
         begin
         P := P + 1;
         for I := 1 to MaxLength do
            FastWrite(X+I-1, Y, S[I], FGround, BGround);
         end
      else
         begin
         LastKey := TheKey;
         SetString;
         Exit;
         end;

   DeleteKey :
      begin
      if P < LastPos then
         begin
         LastPos := LastPos - 1;
```

```
                Delete(S, P, 1);
                S := S + ' ';
                for I := 1 to MaxLength do
                   FastWrite(X+I-1, Y, S[I], FGround, BGround);
                end
             else
                begin
                LastKey := TheKey;
                SetString;
                DoConcat := TRUE;
                Exit;
                end;
             end;

          InsertKey :
             if InsertOn then
                InsertOn := False
             else
                InsertOn := True;

          else
             if not (TheKey in [CarriageReturn, UpArrow, DownArrow,
                                PgDn, PgUp, NullKey, Esc, Tab,
                                F1, F2, F3, F4, F5, F6, F7, F8, F9, F10]) then
       end;

    until (TheKey in [CarriageReturn, UpArrow, DownArrow,
                      PgDn, PgUp, NullKey, Esc, Tab,
                      F1, F2, F3, F4, F5, F6, F7, F8, F9, F10]);

    LastKey := TheKey;
    SetString;
    end;

procedure AbraEditor(var FirstStr      :  EditString;
                     x1, y1, x2, y2,
                     FstX              :  integer;
                     FGround, BGround  :  integer{;
                     Scrollable        :  boolean});

    var
       I, J, K, X, Y :  integer;
       TString,
       Split,
       NumStr : string;
       InsertOn,
       DoConcat,
       AEDone  :  boolean;
       LastSpace : integer;
       P  : integer;
       LastKey  :  KeyType;
       LinkStr,
       NewStr,
       PresStr,
       TmpStr  :  EditString;

    begin
    TString := 'Press F5 when complete; ESC to start over';
    j := 40 - TRUNC(Length(TString)/2);
    for i := 2 to j do
       FastWrite(i, MaxTextLines, ' ', StatusText, StatusBackground);
    for i := 1 to Length(TString) do
       FastWrite(j + i - 1, MaxTextLines, TString[i], StatusText,
                 StatusBackground);
    for i := j + Length(TSTring) to MaxLineLen - 1 do
       FastWrite(i, MaxTextLines, ' ', StatusText, StatusBackground);
```

```
InsertOn := FALSE;
P := 0;
if FirstStr = nil then
   begin
   new(NewStr);
   NewStr^.x := FstX;
   NewStr^.y := y1;
   NewStr^.str := '';
   NewStr^.next := nil;
   NewStr^.prev := nil;
   FirstStr := NewStr;
   end;
PresStr := FirstStr;

AEDone := False;
while not AEDone do
   begin
   i := FstX;
   for j := y1 to y2 do
      begin
      for k := i to x2 do
         FastWrite(k, j, ' ', FGround, BGround);
      i := x1;
      end;
   TmpStr := FirstStr;
   while TmpStr <> nil do
      begin
      for i := 1 to Length(TmpStr^.str) do
         if (TmpStr^.y <= y2) then
            FastWrite(TmpStr^.x + i - 1, TmpStr^.y, TmpStr^.str[i],
                      FGround, BGround);
      TmpStr := TmpStr^.next;
      end;

   GoToXY(LtMargin + P, PresStr^.y);
   if (PresStr = FirstStr) and (x1 <> FstX) then
      InputString(PresStr^.str, P, Split, DoConcat, InsertOn, LastKey,
                  x2 - FstX, PresStr^.x, PresStr^.y, FGround, BGround)
   else
      InputString(PresStr^.str, P, Split, DoConcat, InsertOn, LastKey,
                  x2 - x1, PresStr^.x, PresStr^.y, FGround, BGround);

   if Split <> '' then
      begin
      new(NewStr);
      NewStr^.str := Split;
      NewStr^.x := LtMargin;
      NewStr^.y := PresStr^.y + 1;
      Split := '';
      if PresStr^.next = nil then
         begin
         PresStr^.next := NewStr;
         NewStr^.next := nil;
         NewStr^.prev := PresStr;
         end
      else
         begin
         NewStr^.next := PresStr^.next;
         PresStr^.next^.prev := NewStr;
         NewStr^.prev := PresStr;
         end;
      if P < 0 then
         begin
         PresStr := NewStr;
         P := P * -1;
         end;
```

```
   end
else if (DoConcat) and (P = 0) then
   begin
   if PresStr^.prev <> nil then
      begin
      P := Length(PresStr^.prev^.str) + 1;
      PresStr^.prev^.str := PresStr^.prev^.str + PresStr^.str;
      if (Length(PresStr^.prev^.str) > (x2 - x1)) then
         begin
         i := x2 - x1;
         while PresStr^.prev^.str[i] <> ' ' do
            i := i - 1;
         PresStr^.str := copy(PresStr^.prev^.str, i, Length(PresStr^.prev^.str) - i);
         delete(PresStr^.prev^.str, i, Length(PresStr^.prev^.str) - i);
         end
      else
         begin
         LinkStr := PresStr;
         PresStr := PresStr^.prev;
         PresStr^.next := LinkStr^.next;
         LinkStr^.next^.prev := PresStr;
         dispose(LinkStr);
         end
      end
   end
else if (DoConcat) and (P > 1) then
   begin
   if PresStr^.next <> nil then
      begin
      PresStr^.str := PresStr^.str + PresStr^.next^.str;
      if (Length(PresStr^.str) > (x2 - x1)) then
         begin
         i := x2 - x1;
         while PresStr^.str[i] <> ' ' do
            i := i - 1;
         PresStr^.next^.str := copy(PresStr^.str, i, Length(PresStr^.str) - i);
         delete(PresStr^.str, i, Length(PresStr^.str) - i);
         end
      else
         begin
         LinkStr := PresStr^.next;
         PresStr^.next := LinkStr^.next;
         LinkStr^.next^.prev := PresStr;
         dispose(LinkStr);
         end
      end
   end
else
   begin
   case LastKey of
      F1  :  begin
             end;
      F5  :  AEDone := true;
      CarriageReturn  :
          begin
          if (InsertON) or (PresStr^.next = nil) then
             begin
             new(NewStr);
             if (P = 1) then
                begin
                NewStr^.str := '';
                NewStr^.prev := PresStr^.prev;
                if NewStr^.prev = nil then
                   FirstStr := NewStr
                else
                   NewStr^.prev^.next := NewStr;
```

```
            NewStr^.next := PresStr;
            PresStr^.prev := NewStr;
            NewStr^.x := LtMargin;
            NewStr^.y := PresStr^.y;
            PresStr^.y := PresStr^.y + 1;
            TmpStr := PresStr^.next;
            while TmpStr <> nil do
               begin
               TmpStr^.y := TmpStr^.y + 1;
               TmpStr := TmpStr^.next;
               end;
            P := 0;
            end
         else if P = Length(PresStr^.str) then
            begin
            NewStr^.str := '';
            NewStr^.next := PresStr^.next;
            if NewStr^.next <> nil then
               NewStr^.next^.prev := NewStr;
            PresStr^.next := NewStr;
            NewStr^.prev := PresStr;
            NewStr^.x := LtMargin;
            NewStr^.y := PresStr^.y + 1;
            PresStr := NewStr;
            TmpStr := PresStr^.next;
            while TmpStr <> nil do
               begin
               TmpStr^.y := TmpStr^.y + 1;
               TmpStr := TmpStr^.next;
               end;
            end
         else
            begin
            NewStr^.str := copy(PresStr^.str, P, Length(PresStr^.str) - P + 1);
            delete(PresStr^.str, P, Length(PresStr^.str) - P + 1);
            NewStr^.next := PresStr^.next;
            if NewStr^.next <> nil then
               NewStr^.next^.prev := NewStr;
            PresStr^.next := NewStr;
            NewStr^.prev := PresStr;
            NewStr^.x := LtMargin;
            NewStr^.y := PresStr^.y + 1;
            PresStr := NewStr;
            TmpStr := PresStr^.next;
            while TmpStr <> nil do
               begin
               TmpStr^.y := TmpStr^.y + 1;
               TmpStr := TmpStr^.next;
               end;
            P := 0;
            end;
         end;

      Stats.LocX := LtMargin;
      Stats.LocY := Stats.LocY + 1;
      GoToXY(Stats.LocX, Stats.LocY);
      end;
   LeftArrow  :
      if PresStr^.prev <> nil then
         begin
         PresStr := PresStr^.prev;
         P := Length(PresStr^.str) + 1;
         end;
   RightArrow  :
      if PresStr^.next <> nil then
         begin
```

```
                PresStr := PresStr^.next;
                P := 0;
                end;
          UpArrow    :
              if PresStr^.prev <> nil then
                begin
                PresStr := PresStr^.prev;
                end;
          DownArrow  :
              if PresStr^.next = nil then
                begin
                P := 0;
                new(NewStr);
                NewStr^.str := '';
                NewStr^.next := nil;
                NewStr^.x := LtMargin;
                NewStr^.y := PresStr^.y + 1;
                PresStr^.next := NewStr;
                NewStr^.prev := PresStr;
                PresStr := NewStr;
                end
              else
                begin
                PresStr := PresStr^.next;
                if P > Length(PresStr^.str) then
                   P := Length(PresStr^.str) + 1;
                end;
          end;
       end;
    end;
end;

procedure DisplayDirectory(FileTemplate  :  string);

   var
      Srec :  SearchRec;
      TimeStamp : DateTime;
      DS : DirStr;
      FN : NameStr;
      EN : ExtStr;
      FullName : PathStr;
      X, Y :  integer;
      FF  :  boolean;

   begin
   if ScrTextLines <> NormTextLines then
      begin
      ScrTextLines := NormTextLines;
      AdjustScrTextLines;
      end;
   TopBackground;
   TextBackground(NormalBackground);
   TextColor(NormalText);
   X := 8;
   Y := 5;
   FF := True;
   GoToXY(10, 3);
   write('Directory Listing  :  ');
   FindFirst(FileTemplate, AnyFile, Srec);
   while DosError = 0 do
      begin
      with Srec do
         begin
         if X > 70 then
            begin
            X := 8;
```

```
            Y := Y + 1;
            end;
         FullName := FExpand(Name);
         FSplit(FullName, DS, FN, EN);
         if FF then
            begin
            writeln(DS);
            FF := false;
            end;
         GoToXY(X, Y);
         write(FN + EN);
         X := X + 16;
         end;
      FindNext(Srec);
      end;
   end;

end.
```

Appendix C10

ABRABACK.PAS

```
{*************************************************************************}
{ PARENT PROGRAM:   Advanced Bilingual Reading Assistant                   }
{                   Advanced Bilingual Reading Assistant - Authoring System }
{ PROGRAMMER:       Raymundo A. Q. Marcelo                                 }
{ DATE:             23-April-1995                                          }
{                                                                          }
{ UNIT NAME:        ABRABACK                                               }
{                                                                          }
{ PURPOSE:          This unit contains the functions and procedures required}
{                   by the ABRA and ABRA-AS programs to display the        }
{                   information onto the screen.                           }
{*************************************************************************}

unit ABRABACK;

interface

uses Dos, Crt, ABRAVARS, TP_REF;

{   Procedures used for displaying information }
procedure TopBackground;
procedure BottomBackground;
procedure StatusLine;
procedure DisplayPage;

{   Procedure to change the number of text lines displayed }
{   Usually used when the number of text lines change between Q&A and text }
procedure AdjustScrTextLines;
procedure ResetScrMatrix;
procedure UpdateScrMatrix;
procedure ChangeIndex;

{   Procedures used to display help information to the users }
procedure KeyboardHelpHW;
procedure KeyboardHelpQ;


procedure GetHelpRange;
procedure FindXYBeginRange(HotWord   :  HelpNodePtr;
                           var CurrentLine, x, y  :  LongInt);
procedure GetRange(const  S      :  StringBufferType;
                   var    SPos   :  integer;
                   var    Brange :  LongInt;
                   var    Erange :  LongInt);
procedure GetRangeL(var  S      :  string;
                    var  SPos   :  integer;
                    var  Brange :  LongInt;
                    var  Erange :  LongInt);
procedure GetXYLocation(const  S      :  StringBufferType;
                        var    SPos   :  integer;
                        var    x      :  integer;
                        var    y      :  integer);
procedure GetXYLocationL(var  S      :  string;
                         var  SPos   :  integer;
                         var  x      :  integer;
                         var  y      :  integer);

procedure KillHelpNodeList;
procedure KillQuestionList;

procedure QuitProgram;

implementation

{   PROCEDURES for SCREEN displays   }
```

```
procedure TopBackground;

   var
      i, j, k    :    integer;
      s          :    string;

   begin
   FastWrite(1, 1, #201, NormalText, NormalBackGround);
   s := ' F1 - Help ';
   for j := 1 to Length(s) do
      FastWrite(1 + j, 1, s[j], NormalText, NormalBackGround);
   if H.LevelName = '' then
      begin
      for i := j + 2 to 67 do
         FastWrite(i, 1, #205, NormalText, NormalBackGround);
      end
   else
      begin
      k := 40 - (TRUNC(Length(H.LevelName)/2)) - 1;
      for i := j + 2 to k - 1 do
         FastWrite(i, 1, #205, NormalText, NormalBackGround);
      for i := 1 to Length(H.LevelName) do
         FastWrite(k - 1 + i, 1, H.LevelName[i], NormalText, NormalBackGround);
      k := k + i;
      for i := k to 67 do
         FastWrite(i, 1, #205, NormalText, NormalBackGround);
      end;
   s := ' F10 - EXIT ';
   for j := 1 to Length(s) do
      FastWrite(67 + j, 1, s[j], NormalText, NormalBackGround);

   FastWrite(MaxLineLen, 1, #187, NormalText, NormalBackGround);
   for i := 2 to ScrTextLines do
      begin
      FastWrite(1, i, #186, NormalText, NormalBackGround);
      for j := 2 to MaxLineLen - 1 do
         begin
         FastWrite(j, i, ' ', NormalText, NormalBackGround);
         end;
      FastWrite(MaxLineLen, i, #186, NormalText, NormalBackGround);
      end;
   end;

procedure BottomBackground;

   var
      i, j    :    integer;

   begin
   for i := ScrTextLines + 1 to 24 do
      begin
      FastWrite(1, i, #186, HelpText, HelpBackGround);
      for j := 2 to MaxLineLen - 1 do
         begin
         FastWrite(j, i, ' ', HelpText, HelpBackGround);
         end;
      FastWrite(MaxLineLen, i, #186, HelpText, HelpBackGround);
      end;
   FastWrite(1, 25, #200, HelpText, HelpBackGround);
   for j := 2 to MaxLineLen - 1 do
      FastWrite(j, 25, #205, HelpText, HelpBackGround);
   FastWrite(MaxLineLen, 25, #188, HelpText, HelpBackGround);
   end;

procedure StatusLine;
```

```
var
   i, j      :  integer;
   TString   :  string;

begin
TString := '';
{   Status Line for Highlighting the Text or Answers   }
if ((TextStat = 11) or (TextStat = 22) or (TextStat = 23)) and
   ((DataStat = 3) or (DataStat = 4)) then
   begin
   TString := 'F4 to complete highlighting   INSERT to ';
   if (DataStat = 3) then
      TString := TString + 'finish '
   else
      TString := TString + 'start ';
   TString := TString + 'a marked section';
   end
{  Status Line for the Questions   }
else if (TextStat = 21) then
   begin
   if (WithCursor) then
      begin
      TString := 'ENTER for answer       F5 to edit question      ';
      if (Stats.CurrentQuestion^.NextQuestion <> nil) then
         TString := TString + 'TAB for next question'
      else
         TString := TString + 'TAB for NEW question';
      end
   else
      TString := 'Press ENTER for answer';
   end
{  Status Lines for Answers to the questions   }
else if ((TextStat > 21) and (TextStat < 30)) and (DataStat = 0) then
   begin
   if (WithCursor) then
      begin
      if (Stats.CurrentQuestion^.NextQuestion = nil) then
         begin
         if (Files.CurrLevel^.NextLev <> nil) then
            TString := 'F3 for next level'
         else
            TString := 'F3 for NEW level '
         end
      else
         TString := '                    ';
      TString := TString + '      F5 to edit question      ';
      if (Stats.CurrentQuestion^.NextQuestion <> nil) then
         TString := TString + 'TAB for next question'
      else
         TString := TString + 'TAB for NEW question'
      end
   else
      begin
      if (Stats.CurrentQuestion^.NextQuestion <> nil) then
         TString := 'Press ENTER for next question'
      else
         TString := 'Press ENTER for next level';
      end
   end
else if ((TextStat = 23) or (TextStat = 22)) and (DataStat = 5) then
   begin
   TString := 'Scrolling '+#27+#24+#25+#26+'     Press F4 to begin highlighting an answer'
   end
else if (not WithCursor) and (TextStat = 10) and (ArticleEnd) then
   begin
   TString := 'Press F9 for comprehensive questions';
```

```
      end
   else if (not WithCursor) and (TextStat = 10) and (not ArticleEnd) then
      begin
      TString := 'Scrolling PgUp PgDn '#24+#25+'      Previous Highlight '+#27+'      Next Hi
      end
   else if (not WithCursor) and (TextStat = 30) then
      begin
      TString := ' Press ENTER for article ';
      end
   else if (WithCursor) and (TextStat < 20) then
      begin
      if (IntegratingFile) and (HeadNewHW^.Next <> nil) then
         TString := 'Scrolling '+#27+#24+#25+#26+'  TAB Next Highlight  F4 New Highlight  F9
      else if (Stats.CurrentHotWord^.NextNode <> nil) then
         TString := 'Scrolling '+#27+#24+#25+#26+'  TAB Next Highlight  F4 New Highlight  F9
      else
         TString := 'Scrolling '+#27+#24+#25+#26+'                      F4 New Highlight  F9
      end;

   j := 40 - TRUNC(Length(TString)/2);
   for i := 2 to j do
      FastWrite(i, 25, ' ', StatusText, StatusBackground);
   for i := 1 to Length(TString) do
      FastWrite(j + i, 25, TString[i], StatusText, StatusBackground);
   j := j + Length(TString) + 1;
   for i := j to MaxLineLen - 1 do
      FastWrite(i, 25, ' ', StatusText, StatusBackground);
   end;

procedure KeyboardHelpHW;

   var
      ch   :   char;

   begin
   TextColor(AnswerText);
   TextBackground(AnswerBackground);
   GoToXY(15, 2);
   write('ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ Help ÄÄÄÄÄÄÄÄÄ¸ÄÄÄÄÄÄÄÄÄÄÄÄÄ¿');
   GoToXY(15, 3);
   write('³                                                    ³');
   GoToXY(15, 4);
   write('³   Scrolling   ',#27,'/',#26,'    Go left/right one letter    ³');
   GoToXY(15, 5);
   write('³               ',#24,'/',#25,'    Go up/down one line         ³');
   GoToXY(15, 6);
   write('³               PgUp   Page up one screen          ³');
   GoToXY(15, 7);
   write('³               PgDn   Page down one screen        ³');
   GoToXY(15, 8);
   write('³               Home   Go to text beginning        ³');
   GoToXY(15, 9);
   write('³               End    Go to text end              ³');
   GoToXY(15, 10);
   write('³                                                   ³');
   GoToXY(15, 11);
   write('³   FUNCTION KEY    DESCRIPTION                     ³');
   GoToXY(15, 12);
   write('³      F1           Keyboard Help Screen            ³');
   GoToXY(15, 13);
   write('³      F2           Go To Previous Level            ³');
   GoToXY(15, 14);
   write('³      F3           Go To Next Level                ³');
   GoToXY(15, 15);
   write('³      F4, INSERT   Highlights Word(s)/Phrases      ³');
   GoToXY(15, 16);
```

```
    write('³      F5          Edits Highlight Definition     ³');
    GoToXY(15, 17);
    write('³      F6          Gets Highlighted Words File    ³');
    GoToXY(15, 18);
    write('³      F7          Saves Highlights & Definitions ³');
    GoToXY(15, 19);
    write('³      F8          Edits Introduction             ³');
    GoToXY(15, 20);
    write('³      F9          Edits Questions                ³');
    GoToXY(15, 21);
    write('³      F10         Exits Program                  ³');
    GoToXY(15, 22);
    write('³      DELETE      Deletes Highlight              ³');
    GoToXY(15, 23);
    write('³                                                 ³');
    GoToXY(15, 24);
    write('ÀÄÄÄÄÄÄÄÄÄÄÄÄ  Press any key to continue ÄÄÄÄÄÄÄÄÄÄÄÄÙ');

    repeat
    until Keypressed;
    ch := ReadKey;
    TopBackground;
    BottomBackground;
    end;

procedure KeyboardHelpQ;

    var
       ch   :   char;

    begin
    TextColor(AnswerText);
    TextBackground(AnswerBackground);
    GoToXY(15, 2);
    write('ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ Help ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿');
    GoToXY(15, 3);
    write('³                                                 ³');
    GoToXY(15, 4);
    write('³   Scrolling   ',#27,'/',#26,'    Go left/right one letter    ³');
    GoToXY(15, 5);
    write('³               ',#24,'/',#25,'    Go up/down one line         ³');
    GoToXY(15, 6);
    write('³               PgUp   Page up one screen          ³');
    GoToXY(15, 7);
    write('³               PgDn   Page down one screen        ³');
    GoToXY(15, 8);
    write('³               Home   Go to text beginning        ³');
    GoToXY(15, 9);
    write('³               End    Go to text end              ³');
    GoToXY(15, 10);
    write('³                                                 ³');
    GoToXY(15, 11);
    write('³   FUNCTION KEY    DESCRIPTION                   ³');
    GoToXY(15, 12);
    write('³      F1           Keyboard Help Screen          ³');
    GoToXY(15, 13);
    write('³      F2           Go To Previous Level          ³');
    GoToXY(15, 14);
    write('³      F3           Go To Next Level              ³');
    GoToXY(15, 15);
    write('³      F4, INSERT   Highlights Multiple Choices   ³');
    GoToXY(15, 16);
    write('³      F5           Edits Question                ³');
    GoToXY(15, 17);
    write('³      F6           Gets Question File            ³');
    GoToXY(15, 18);
```

```
   write('³      F7          Saves Questions                ³');
   GoToXY(15, 19);
   write('³      F8          Edits Introduction             ³');
   GoToXY(15, 20);
   write('³      F9          Edits Highlighted Words        ³');
   GoToXY(15, 21);
   write('³      F10         Exits Program                  ³');
   GoToXY(15, 22);
   write('³      DELETE      Deletes Question               ³');
   GoToXY(15, 23);
   write('³                                                 ³');
   GoToXY(15, 24);
   write('ÀÄÄÄÄÄÄÄÄÄÄÄÄ  Press any key to continue ÄÄÄÄÄÄÄÄÄÄÄÄÙ');

   repeat
   until Keypressed;
   ch := ReadKey;
   TopBackground;
   BottomBackground;
   end;

{  Adjusts the number of lines displayed for the text area when the screen
   changes from Interlinear to Normal }
procedure AdjustScrTextLines;

   begin
   Stats.LastLine := Stats.FirstLine + ScrTextLines - 2;
   if Stats.LastLine < TotalLines then
      Stats.LastPos := TextPage[Stats.LastLine + 1] - 1
   else
      begin
      Stats.LastLine := TotalLines;
      Stats.LastPos := TotalChar;
      end;
   ReDisplayPage := TRUE;
   end;

procedure FindXYBeginRange(HotWord   :  HelpNodePtr;
                           var CurrentLine, x, y  :  LongInt);

   begin
   y := 1;
   CurrentLine := Stats.FirstLine;
   while (HotWord^.BeginRange >= TextPage[CurrentLine]) do
      begin
      CurrentLine := CurrentLine + 1;
      y := y + 1;
      end;
   CurrentLine := CurrentLine - 1;
   x := HotWord^.BeginRange - TextPage[CurrentLine] + LtMargin;
   end;

procedure ResetScrMatrix;

   var
      x, y, k, line  :  LongInt;

   begin
   for y := 1 to MaxTextLines do
      for x := 1 to MaxLineLen do
         ScreenHelp[x, y] := 0;
   end;

procedure UpdateScrMatrix;

   var
```

```
   x, y, k, line   :  LongInt;
   NodePlace,
   HotWord,
   FinalHotWord    :  HelpNodePtr;
   Done            :  boolean;

begin
NodePlace := nil;
ResetScrMatrix;

if (TextStat < 20) then
   begin
   if Stats.LastDisplay = nil then
      Done := true
   else
      begin
      HotWord := Stats.FirstDisplay;
      FinalHotWord := Stats.LastDisplay;
      Done := False;
      end;

   while not Done do
      begin
      FindXYBeginRange(HotWord, line, x, y);
      k := HotWord^.BeginRange;
      while (k <= HotWord^.EndRange) and (k <= Stats.LastPos) do
         begin
         ScreenHelp[x, y] := HotWord^.Index;
         x := x + 1;
         if TextPage[line] + x - LtMargin = TextPage[line + 1] then
            begin
            x := LtMargin;
            y := y + 1;
            line := line + 1;
            end;
         k := k + 1;
         end;

      if (HotWord = FinalHotWord) then
         Done := TRUE
      else
         HotWord := HotWord^.NextDisplay;
      end;

   if TmpNewNode <> nil then
      begin
      HotWord := TmpNewNode;
      repeat
      FindXYBeginRange(HotWord, line, x, y);
      k := HotWord^.BeginRange;
      while (k <= HotWord^.EndRange) and (k <= Stats.LastPos) do
         begin
         ScreenHelp[x, y] := HotWord^.Index;
         x := x + 1;
         if TextPage[line] + x - LtMargin = TextPage[line + 1] then
            begin
            x := LtMargin;
            y := y + 1;
            line := line + 1;
            end;
         k := k + 1;
         end;

      if (HotWord^.NextPartner <> nil) then
         HotWord := HotWord^.NextPartner
      else
```

```
            HotWord := nil;
         until (k > Stats.LastPos) or (HotWord = nil);
         end;
      end
   else if (TextStat = 22) then
      begin
      if TmpNewNode <> nil then
         begin
         HotWord := TmpNewNode;
         repeat
         FindXYBeginRange(HotWord, line, x, y);
         k := HotWord^.BeginRange;
         while (k <= HotWord^.EndRange) and (k <= Stats.LastPos) do
            begin
            ScreenHelp[x, y] := HotWord^.Index;
            x := x + 1;
            if TextPage[line] + x - LtMargin = TextPage[line + 1] then
               begin
               x := LtMargin;
               y := y + 1;
               line := line + 1;
               end;
            k := k + 1;
            end;

         if (HotWord^.NextPartner <> nil) then
            HotWord := HotWord^.NextPartner
         else
            HotWord := nil;
         until (k > Stats.LastPos) or (HotWord = nil);
         end;
      end;
   end;

procedure ChangeIndex;

   var
      HotWord  :  HelpNodePtr;
      Line, x, y, k  :  LongInt;

   begin
   HotWord := Stats.CurrentHotWord;
   repeat
      FindXYBeginRange(HotWord, line, x, y);
      k := HotWord^.BeginRange;
      while (k <= HotWord^.EndRange) and (k <= Stats.LastPos) do
         begin
         ScreenHelp[x, y] := HotWord^.Index;
         x := x + 1;
         if TextPage[line] + x - LtMargin = TextPage[line + 1] then
            begin
            x := LtMargin;
            y := y + 1;
            line := line + 1;
            end;
         k := k + 1;
         end;

      HotWord := HotWord^.NextPartner;
   until (k > Stats.LastPos) or (HotWord = nil);
   end;

procedure GetHelpRange;

   var
      RangeValue   :   LongInt;
```

```
   PHotWord,
   HotWord       :   HelpNodePtr;
   NPDone,
   RVDone        :   boolean;

begin
HotWord := Stats.FirstHotWord;

{  If there MAY BE Hot Words for this page  }
if (HotWord <> nil) or (TextStat = 22) then
   begin
   RangeValue := HotWord^.BeginRange;

   {  If beginning of range is on screen  }
   if (RangeValue >= Stats.FirstPos) and
      (RangeValue <= Stats.LastPos) then
      begin
      Stats.LastHotWord := HotWord;
      end

   {   If the range value is on the NEXT Page    }
   else if (RangeValue > Stats.LastPos) then
      begin
      RVDone := FALSE;
      while (not RVDone) do
         if (HotWord <> nil) and
            (HotWord^.BeginRange > Stats.LastPos) then
            HotWord := HotWord^.PrevNode
         else
            RVDone := TRUE;
      if ((Stats.CurrentHotWord^.BeginRange > Stats.LastPos) or
         (Stats.CurrentHotWord^.BeginRange < Stats.FirstPos)) then
         Stats.CurrentHotWord := HotWord;
      end

   {   If the range value is on the PREVIOUS page  }
   else if (RangeValue < Stats.FirstPos) then
      begin
      RVDone := FALSE;
      while (not RVDone) do
         if (HotWord <> nil) and
            (HotWord^.BeginRange < Stats.FirstPos) then
            HotWord := HotWord^.NextNode
         else
            RVDone := TRUE;
      if ((Stats.CurrentHotWord^.BeginRange > Stats.LastPos) or
         (Stats.CurrentHotWord^.BeginRange < Stats.FirstPos)) then
         Stats.CurrentHotWord := HotWord;
      end;

   {  Find the First and Last Hot Word on the Page    }
   if (HotWord <> nil) then
      begin
      Stats.FirstHotWord := HotWord;
      Stats.LastHotWord := HotWord;
      RVDone := False;
      while (Stats.FirstHotWord^.PrevNode <> nil) and (not RVDone) do
         if (Stats.FirstHotWord^.PrevNode^.BeginRange >= Stats.FirstPos) then
            Stats.FirstHotWord := Stats.FirstHotWord^.PrevNode
         else
            if Stats.FirstHotWord^.PrevNode^.NextPartner = nil then
               RVDone := TRUE
            else
               begin
               NPDone := FALSE;
               PHotWord := Stats.FirstHotWord^.PrevNode;
```

```
                while (PHotWord^.NextPartner <> nil) and (not NPDone) do
                   begin
                   PHotWord := PHotWord^.NextPartner;
                   if (PHotWord^.BeginRange >= Stats.FirstPos) then
                      NPDone := TRUE;
                      Stats.FirstHotWord := Stats.FirstHotWord^.PrevNode;
                   end;
                end;
          RVDone := False;
          while (Stats.LastHotWord^.NextNode <> nil) and (not RVDone) do
             if (Stats.LastHotWord^.NextNode^.BeginRange <= Stats.LastPos) then
                Stats.LastHotWord := Stats.LastHotWord^.NextNode
             else
                RVDone := TRUE;

          Stats.FirstDisplay := Stats.FirstHotWord;
          RVDone := False;
          while (Stats.FirstDisplay^.PrevDisplay <> nil) and (not RVDone) do
             if (Stats.FirstDisplay^.PrevDisplay^.BeginRange >= Stats.FirstPos) then
                Stats.FirstDisplay := Stats.FirstDisplay^.PrevDisplay
             else
                RVDone := TRUE;
          Stats.LastDisplay := Stats.LastHotWord;
          RVDone := False;
          while (Stats.LastDisplay^.NextDisplay <> nil) and (not RVDone) do
             if (Stats.LastDisplay^.NextDisplay^.BeginRange < Stats.LastPos) then
                Stats.LastDisplay := Stats.LastDisplay^.NextDisplay
             else
                RVDone := TRUE;
          end
       else
          begin
          Stats.LastHotWord := nil;
          Stats.LastDisplay := nil;
          end
       end

    {  If no HotWords are on the page    }
    else
       begin
       Stats.CurrentHotWord := nil;
       Stats.LastHotWord := nil;
       Stats.LastDisplay := nil;
       end;
    end;

procedure GetRange(const  S      :  StringBufferType;
                   var    SPos   :  integer;
                   var    Brange :  LongInt;
                   var    Erange :  LongInt);

    var
       i, Code :   integer;
       NumStr  :   string;

    begin
    i := SPos;
    if (S[i] = ',') or (S[i] = '<') or (S[i] = '[') then
       i := i + 1;
    NumStr := '';
    while S[i] <> ',' do
       begin
       NumStr := NumStr + S[i];
       i := i + 1;
       end;
    i := i + 1;
```

```
   val(NumStr, Brange, Code);
   NumStr := '';
   while (S[i] <> '>') and (S[i] <> ']') and (S[i] <> ',') do
      begin
      NumStr := NumStr + S[i];
      i := i + 1;
      end;
   val(NumStr, Erange, Code);
   SPos := i;
   end;

procedure GetRangeL(var  S      :  string;
                    var  SPos   :  integer;
                    var  Brange :  LongInt;
                    var  Erange :  LongInt);

   var
      i, Code :   integer;
      NumStr  :   string;

   begin
   i := SPos;
   if (S[i] = ',') or (S[i] = '<') or (S[i] = '[') then
      i := i + 1;
   NumStr := '';
   while S[i] <> ',' do
      begin
      NumStr := NumStr + S[i];
      i := i + 1;
      end;
   i := i + 1;
   val(NumStr, Brange, Code);
   NumStr := '';
   while (S[i] <> '>') and (S[i] <> ']') and (S[i] <> ',') do
      begin
      NumStr := NumStr + S[i];
      i := i + 1;
      end;
   val(NumStr, Erange, Code);
   SPos := i;
   end;

procedure GetXYLocation(const  S      :  StringBufferType;
                        var    SPos   :  integer;
                        var    x      :  integer;
                        var    y      :  integer);

   var
      i, Code :   integer;
      NumStr  :   string;

   begin
   i := SPos;
   if (S[i] = ',') or (S[i] = '<') or (S[i] = '[') then
      i := i + 1;
   NumStr := '';
   while S[i] <> ',' do
      begin
      NumStr := NumStr + S[i];
      i := i + 1;
      end;
   i := i + 1;
   val(NumStr, x, Code);
   NumStr := '';
   while (S[i] <> '>') and (S[i] <> ']') and (S[i] <> ',') do
      begin
```

```
      NumStr := NumStr + S[i];
      i := i + 1;
      end;
   val(NumStr, y, Code);
   SPos := i;
   end;

procedure GetXYLocationL(var  S      :  string;
                         var  SPos   :  integer;
                         var  x      :  integer;
                         var  y      :  integer);

   var
      i, Code :   integer;
      NumStr  :   string;

   begin
   i := SPos;
   if (S[i] = ',') or (S[i] = '<') or (S[i] = '[') then
      i := i + 1;
   NumStr := '';
   while S[i] <> ',' do
      begin
      NumStr := NumStr + S[i];
      i := i + 1;
      end;
   i := i + 1;
   val(NumStr, x, Code);
   NumStr := '';
   while (S[i] <> '>') and (S[i] <> ']') and (S[i] <> ',') do
      begin
      NumStr := NumStr + S[i];
      i := i + 1;
      end;
   val(NumStr, y, Code);
   SPos := i;
   end;

procedure KillHelpNodeList;

   var
      TNode,
      T1     :  HelpNodePtr;

   begin
   Stats.FirstHotWord := nil;
   Stats.CurrentHotWord := nil;
   Stats.LastHotWord := nil;
   Stats.FirstDisplay := nil;
   Stats.LastDisplay := nil;
   while H.HelpNodeList <> nil do
      begin
      TNode := H.HelpNodeList;
      H.HelpNodeList := H.HelpNodeList^.NextNode;
      while TNode^.NextPartner <> nil do
         begin
         T1 := TNode;
         TNode := TNode^.NextPartner;
         Dispose(T1);
         end;
      Dispose(TNode);
      end;
   H.HelpNodeList := nil;
   H.DisplayList := nil;
   end;
```

```
procedure KillQuestionList;

   var
      QNode  :  QuestionPtr;

   begin
   while H.Questions <> nil do
      begin
      QNode := H.Questions;
      H.Questions := H.Questions^.NextQuestion;
      Dispose(QNode);
      end;
   Stats.CurrentQuestion := nil;
   end;

procedure DisplayPage;

   var
      k                  :  Longint;
      i, j               :  integer;
      BeginPos,
      EndPos,
      TextNum            :  Longint;
      FGround, BGround   :  integer;

   begin
   TopBackground;
   BeginPos := Stats.FirstPos - (Page * MaxPageSize);
   EndPos := Stats.LastPos - (Page * MaxPageSize);

   k := Stats.FirstLine;
   i := 1;
   j := LtMargin;
   for TextNum := BeginPos to EndPos do
      begin
      if TextNum = TextPage[k] - (Page * MaxPageSize) then
         begin
         i := i + 1;
         j := LtMargin;
         k := k + 1;
         end;

      if ((ScreenHelp[j, i] = 0) or ((DataStat = 0) and (TextStat <> 22))) then
         begin
         FGround := NormalText;
         BGround := NormalBackground;
         end
      else if (ScreenHelp[j, i] > 0) and
              (((Stats.CurrentHotWord <> nil) and
                (ScreenHelp[j,i] = Stats.CurrentHotWord^.Index)) or
               ((TmpNode <> nil) and (ScreenHelp[j,i] = TmpNode^.Index)) or
               ((TmpNewNode <> nil) and ((DataStat = 0) and (TextStat = 22)))) then
         begin
         FGround := SelectedText - Blink;
         BGround := SelectedBackground;
         end
      else
         begin
         FGround := SelectableText;
         BGround := SelectableBackground;
         end;
      FastWrite(j, i, TextBuffer[TextNum], FGround, BGround);

      j := j + 1;
      end;
    end;
```

```
procedure QuitProgram;

   var
     TmpLevel   :  LevelPtr;

   begin
   Rewrite(FileFile);
   if Files.Intro <> nil then
      begin
      writeln(FileFile, '@I ', Files.Intro^.Lname);
      writeln(FileFile, Files.Intro^.Fname);
      end;
   if Files.Article <> nil then
      begin
      writeln(FileFile, '@T ');
      writeln(FileFile, Files.Article^.Fname);
      end;
   TmpLevel := Files.Levels;
   while TmpLevel <> nil do
      begin
      write(FileFile, '@H ');
      if TmpLevel^.Interlinear then
         write(FileFile, 'I ')
      else
         write(FileFile, 'N ');
      writeln(FileFile, TmpLevel^.LevName);
      if TmpLevel^.HWName <> '' then
         write(FileFile, TmpLevel^.HWName);
      if TmpLevel^.QuesName <> '' then
         writeln(FileFile, '|', TmpLevel^.QuesName)
      else
         writeln(FileFile, '|');
      TmpLevel := TmpLevel^.NextLev;
      end;
   close(FileFile);
   end;

end.
```

Appendix C11

TP_REF.PAS

```
{*************************************************************************}
{ PARENT PROGRAM:   Advanced Bilingual Reading Assistant                   }
{                   Advanced Bilingual Reading Assistant - Authoring System }
{ PROGRAMMER:       Raymundo A. Q. Marcelo                                 }
{ DATE:             23-April-1995                                          }
{                                                                          }
{ UNIT NAME:        TP_REF                                                 }
{                                                                          }
{ PURPOSE:          This unit contains the functions and procedures copied  }
{                   or adapted from:                                       }
{                      Turbo Pascal 7:  the Complete Reference             }
{                         by Stephen K. O'Brien & Steve Nameroff           }
{                   These functions and procedures perform many of the input}
{                   and output functions within both programs.             }
{*************************************************************************}

Unit TP_REF;

interface

uses Dos, Crt, ABRAVARS;

{   These constant types and procedures identify the key and
    the constant type of the key pressed                          }
type
   KeyType = (NullKey, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10,
              CarriageReturn, Tab, ShiftTab, Bksp,
              UpArrow, DownArrow, RightArrow, LeftArrow,
              DeleteKey, InsertKey, HomeKey, EndKey, PgUp, PgDn,
              Esc, TextKey, NumberKey, Space);

procedure InKey(var IsSpecial   :  boolean;
                var SpecialKey  :  KeyType;
                var Ch          :  char);

{  These procedures control the size of the cursor on the screen  }

procedure Cursor_Off;
procedure Cursor_Small;
procedure Cursor_Big;

{  These procedures control information written to the screen
   by writing the information onto the video array                }

procedure FastWrite(X, Y     :  byte;
                    Ch       :  char;
                    FGround,
                    BGround  :  byte);

procedure InputStringShift(var S : string;
                           WindowLength,
                           MaxLength,
                           X, Y :  integer;
                           FT   : char;
                           BackgroundChar :  integer);

implementation

var
   VS : word;

{   This procedure turns off the cursor   }
procedure Cursor_Off;

   var
      Regs :  Registers;
```

```
   begin
   with Regs do
      begin
      AH := $01;
      CH := $20;
      CL := $20;
      end;
   intr($10, Regs);
   end;

{   This procedure makes the cursor small as in the default condition  }
procedure Cursor_Small;

   var
      Regs  :  Registers;

   begin
   with Regs do
      begin
      AH := $01;
      CH := 6;
      CL := 7;
      end;
   intr($10, Regs);
   end;

{   This procedure makes the cursor big as when in overwrite mode }
procedure Cursor_Big;

   var
      Regs  :  Registers;

   begin
   with Regs do
      begin
      AH := $01;
      CH := 0;
      CL := 7;
      end;
   intr($10, Regs);
   end;

procedure Beep(Freq, Time : integer);

   begin
   Sound(Freq);
   Delay(Time);
   NoSound;
   end;

{   This function sets the memory location for the screen array for a
    color or black & white monitor                                    }
function VidSeg : word;

   begin
   if Mem[$0000:$0449] = 7 then
      VidSeg := $B000
   else
      VidSeg := $B800;
   end;

{   This procedure writes a character along with its foreground and
    background color onto a memory location in the screen array   }
procedure FastWrite(X, Y     :  byte;
                    Ch       :  char;
```

```
                    FGround,
                    BGround   :  byte);

   var
      W  :  word;
      I, ColAtr : byte;

   begin
   ColAtr := (BGround shl 4) + FGround;    { create attribute byte }
   W := ((Y-1)*80 + (X-1))*2;

   MemW[VS:W] := (ColAtr shl 8) + Ord(Ch);
   end;

{   This procedure is modified from O'Brien & Nameroff's InputStringShift.
    This procedure is called by the editor procedure to perform functions
    on a particular line being editted                                    }
procedure InputStringShift(var S           : string;
                           WindowLength,
                           MaxLength,
                           X, Y            :  integer;
                           FT              : char;
                           BackgroundChar  :  integer);

   var
      XX, I, J, P :  integer;
      Ch          : char;
      InsertOn,
      IsSpecial   :  boolean;
      TheKey      :  KeyType;
      Offset      :  integer;

   procedure XY(X, Y : integer);
      var
         Xsmall : integer;
      begin
         repeat
            Xsmall := X - 80;
            if Xsmall > 0 then
               begin
               Y := Y + 1;
               X := Xsmall;
               end;
         until Xsmall <= 0;
         GoToXY(X, Y);
      end;

   procedure SetString;
      var
         I  :  integer;
      begin
         I := Length(S);
         while S[I] = char(BackgroundChar) do
            I := I - 1;
         S[0] := char(I);
      end;

   begin
   J := Length(S) + 1;
   for I := J to MaxLength do
      S[I] := char(BackgroundChar);
   S[0] := char(MaxLength);

   for I := 1 to WindowLength do
      FastWrite(X+I-1, Y, S[I], Yellow, Black);
   P := 1;
```

```
Offset := 0;
InsertOn := True;

repeat
   XX := X + (P - Offset) - 1;
   if (P - Offset) = WindowLength then
      XX := XX - 1;
   XY(XX, Y);

   InKey(IsSpecial, TheKey, Ch);

   if (FT = 'N') then
      begin
      if (TheKey = TextKey) or
         ((Ch = '-') and ((P > 1) or (S[1] = '-'))) then
         begin
         Beep(100, 250);
         TheKey := NullKey;
         end
      else if (Ch = '.') then
         begin
         if not ((Pos('.', S) = 0) or (Pos('.', S) = P)) then
            begin
            Beep(100, 250);
            TheKey := NullKey;
            end
         else if (Pos('.', S) = P) then
            Delete(S, P, 1);
         end;
      end;

   case TheKey of
      NumberKey, TextKey, Space :
         if Length(S) = MaxLength then
            begin
            if P = MaxLength then
               begin
               Delete(S, MaxLength, 1);
               S := S + Ch;
               if P = WindowLength + Offset then
                  Offset := Offset + 1;
               for I := 1 to WindowLength do
                  FastWrite(X+I-1, Y, S[I+Offset], Yellow, Black);
               end
            else
               begin
               if InsertOn then
                  begin
                  Delete(S, MaxLength, 1);
                  Insert(Ch, S, P);
                  if P = WindowLength + Offset then
                     Offset := Offset + 1;
                  if P < MaxLength then
                     P := P + 1;
                  for I := 1 to WindowLength do
                     FastWrite(X+I-1, Y, S[I+Offset], Yellow, Black);
                  end
               else  {overwrite}
                  begin
                  Delete(S, P, 1);
                  Insert(Ch, S, P);
                  if P = WindowLength + Offset then
                     Offset := Offset + 1;
                  if P < MaxLength then
                     P := P + 1;
                  for I := 1 to WindowLength do
```

```
                FastWrite(X+I-1, Y, S[I+Offset], Yellow, Black);
             end;
          end;
       end
    else  {not the last character}
       begin
       if InsertOn then
          Insert(Ch, S, P)
       else
          begin
          Delete(S, P, 1);
          Insert(Ch, S, P);
          end;
       if P = WindowLength + Offset then
          Offset := Offset + 1;
       if P < MaxLength then
          P := P + 1;
       for I := 1 to WindowLength do
          FastWrite(X+I-1, Y, S[I+Offset], Yellow, Black);
       end;

Bksp :
    if P > 1 then
       begin
       P := P - 1;
       Delete(S, P, 1);
       S := S + Char(BackgroundChar);
       if Offset > 1 then
          Offset := Offset - 1;
       for I := 1 to WindowLength do
          FastWrite(X+I-1, Y, S[I+Offset], Yellow, Black);
       Ch := ' ';
       end
    else
       begin
       beep(100, 250);
       Ch := ' ';
       P := 1;
       end;

LeftArrow :
    if P > 1 then
       begin
       P := P - 1;
       if P < Offset then
          begin
          Offset := Offset - 1;
          for I := 1 to WindowLength do
             FastWrite(X+I-1, Y, S[I+Offset], Yellow, Black);
          end;
       end
    else
       begin
       SetString;
       Exit;
       end;

RightArrow :
    if (S[P] <> char(BackgroundChar)) and
       (P < MaxLength) then
       begin
       P := P + 1;
       if P = (WindowLength + Offset) then
          begin
          Offset := Offset + 1;
          for I := 1 to WindowLength do
```

```
                  FastWrite(X+I-1, Y, S[I+Offset], Yellow, Black);
               end;
            end
         else
            begin
            SetString;
            Exit;
            end;

      DeleteKey :
         begin
         Delete(S, P, 1);
         S := S + char(BackgroundChar);
         for I := 1 to WindowLength do
            FastWrite(X+I-1, Y, S[I+Offset], Yellow, Black);
         end;

      InsertKey :
         if InsertOn then
            InsertOn := False
         else
            InsertOn := True;

      else
         if not (TheKey in [CarriageReturn, UpArrow, DownArrow,
                            PgDn, PgUp, NullKey, Esc, Tab,
                            F1, F2, F3, F4, F5, F6, F7, F8, F9, F10]) then
            Beep(100, 250);
   end;

until (TheKey in [CarriageReturn, UpArrow, DownArrow,
                  PgDn, PgUp, NullKey, Esc, Tab,
                  F1, F2, F3, F4, F5, F6, F7, F8, F9, F10]);
SetString;
end;

{   This procedure is modified from O'Brien & Nameroff's InputString.
    This procedure is called by the editor procedure to perform line
    wrapping and concatenation of consectutive lines                    }
procedure InputString(var S            : string;
                      var P            : integer;
                      var Split        : string;
                      var DoConcat     : boolean;
                      var InsertOn     : boolean;
                      var LastKey      : KeyType;
                      MaxLength,
                      X, Y             :  integer;
                      FGround, BGround :  integer);

var
   I, J, P1  :  integer;
   Ch        : char;
   IsSpecial :  boolean;
   TheKey    :  KeyType;
   LastSpace : integer;
   LastPos   : integer;

procedure SetString;
   var
      I  :  integer;
   begin
      I := Length(S);
      while S[I] = ' ' do
         I := I - 1;
      S[0] := char(I);
   end;
```

```
begin
if S <> '' then
   LastPos := Length(S) + 1
else
   LastPos := 2;
Split := '';
DoConcat := FALSE;
J := Length(S) + 1;
for I := J to MaxLength do
   S[I] := ' ';
S[0] := char(MaxLength);

for I := 1 to MaxLength do
   FastWrite(X+I-1, Y, S[I], FGround, BGround);
P := P + 1;
InsertOn := True;
LastSpace := 0;

repeat
   GoToXY(X + P - 1, Y);

   InKey(IsSpecial, TheKey, Ch);

   case TheKey of
      NumberKey, TextKey, Space :
         begin
         if (LastPos < MaxLength) and (P < MaxLength) then
            begin
            if (TheKey = Space) and (P < MaxLength) then
               LastSpace := P;
            if InsertOn then
               Insert(Ch, S, P)
            else
               begin
               Delete(S, P, 1);
               Insert(Ch, S, P);
               end;
            if P < MaxLength then
               begin
               P := P + 1;
               LastPos := LastPos + 1;
               end;
            for I := 1 to MaxLength do
               FastWrite(X+I-1, Y, S[I], FGround, BGround);
            end
         else if LastPos = MaxLength then
            begin
            if (InsertON) or (P = LastPos) then
               begin
               P1 := MaxLength - LastSpace;
               Split := copy(S, LastSpace, P1);
               Split := Split + ch;
               Delete(S, LastSpace, P1);
               LastPos := LastSpace;
               if LastPos < P then
                  P := -1 * (P - LastPos + 1);
               for i := 1 to P1 do
                  S := S + ' ';
               for I := 1 to MaxLength do
                  FastWrite(X+I-1, Y, S[I], FGround, BGround);
               LastKey := TheKey;
               SetString;
               Exit;
               end
            else
```

```
            begin
            Delete(S, P, 1);
            Insert(Ch, S, P);
            if P < MaxLength then
               begin
               P := P + 1;
               LastPos := LastPos + 1;
               end;
            for I := 1 to MaxLength do
               FastWrite(X+I-1, Y, S[I], FGround, BGround);
            end;
         end;
      end;
   Bksp :
      if P > 1 then
         begin
         P := P - 1;
         LastPos := LastPos - 1;
         Delete(S, P, 1);
         S := S + ' ';
         for I := 1 to MaxLength do
            FastWrite(X+I-1, Y, S[I], FGround, BGround);
         Ch := ' ';
         end
      else
         begin
         SetString;
         DoConcat := TRUE;
         P := 0;
         LastKey := TheKey;
         Exit;
         end;

   LeftArrow :
      if P > 1 then
         begin
         P := P - 1;
         for I := 1 to MaxLength do
            FastWrite(X+I-1, Y, S[I], FGround, BGround);
         end
      else
         begin
         LastKey := TheKey;
         SetString;
         Exit;
         end;

   RightArrow :
      if (P <= LastPos) and
         (P <= MaxLength) then
         begin
         P := P + 1;
         for I := 1 to MaxLength do
            FastWrite(X+I-1, Y, S[I], FGround, BGround);
         end
      else
         begin
         LastKey := TheKey;
         SetString;
         Exit;
         end;

   DeleteKey :
      begin
      if P < LastPos then
         begin
```

```
               LastPos := LastPos - 1;
               Delete(S, P, 1);
               S := S + ' ';
               for I := 1 to MaxLength do
                  FastWrite(X+I-1, Y, S[I], FGround, BGround);
               end
            else
               begin
               LastKey := TheKey;
               SetString;
               DoConcat := TRUE;
               Exit;
               end;
            end;

         InsertKey :
            if InsertOn then
               InsertOn := False
            else
               InsertOn := True;

         else
            if not (TheKey in [CarriageReturn, UpArrow, DownArrow,
                               PgDn, PgUp, NullKey, Esc, Tab,
                               F1, F2, F3, F4, F5, F6, F7, F8, F9, F10]) then
      end;

   until (TheKey in [CarriageReturn, UpArrow, DownArrow,
                     PgDn, PgUp, NullKey, Esc, Tab,
                     F1, F2, F3, F4, F5, F6, F7, F8, F9, F10]);

   LastKey := TheKey;
   SetString;
   end;

{   This procedure has a keyboard character input and matches this input
    with a constant name easily usable by a programmer                    }
procedure InKey(var IsSpecial   :  boolean;
                var SpecialKey  :  KeyType;
                var Ch          : char);

   begin
   IsSpecial := False;
   Ch := ReadKey;
   if (Ch = #0) then
      begin
      IsSpecial := True;
      Ch := ReadKey;
      end;

   if IsSpecial then
      case Ord(Ch) of
         15:  SpecialKey := ShiftTab;
         72:  SpecialKey := UpArrow;
         80:  SpecialKey := DownArrow;
         75:  SpecialKey := LeftArrow;
         77:  SpecialKey := RightArrow;
         73:  SpecialKey := PgUp;
         81:  SpecialKey := PgDn;
         71:  SpecialKey := HomeKey;
         79:  SpecialKey := EndKey;
         83:  SpecialKey := DeleteKey;
         82:  SpecialKey := InsertKey;
         59:  SpecialKey := F1;
         60:  SpecialKey := F2;
         61:  SpecialKey := F3;
```

```
            62:  SpecialKey := F4;
            63:  SpecialKey := F5;
            64:  SpecialKey := F6;
            65:  SpecialKey := F7;
            66:  SpecialKey := F8;
            67:  SpecialKey := F9;
            68:  SpecialKey := F10;
         end
      else
         case Ord(Ch) of
             8:  SpecialKey := Bksp;
             9:  SpecialKey := Tab;
            13:  SpecialKey := CarriageReturn;
            27:  SpecialKey := Esc;
            32:  SpecialKey := Space;
            45..46,
            48..57:  SpecialKey := NumberKey;
            else SpecialKey := TextKey;
         end;
      end;

begin
   VS := VidSeg;
end.
```

Appendix D

Screen Shot Examples of Some of the Functions

NOTE: In the following section, all examples use the following formatting codes.
BOLD = Current Highlighted Word or Phrase
ITALIC = A Highlightable Word or Phrase
_ = Cursor Position

Moving between Highlighted Words

F1 - Help | Level 1 -- Vocabulary | F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne *Erinnerung* ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik Deutschland die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten **Wertewandel** zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie *lobenswert*. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über *mangelnden Zuspruch* nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

change in values

Step 1. Before Pressing the <TAB> key

F1 - Help Level 1 -- Vocabulary F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne *Erinnerung* ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik Deutschland die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten *Wertewandel* zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie **lobenswert**. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über *mangelnden Zuspruch* nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

praiseworthy

Step 2. After Pressing the <TAB> key

Adding a Highlight and a Definition

F1 - Help | Level 1 -- Vocabulary | F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne **Erinnerung** ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik Deutschland die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten *Wertewandel* zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über *mangelnden Zuspruch* nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

memory

Step 1. Place cursor on First Character to Highlight

F1 - Help Level 1 -- Vocabulary F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne *Erinnerung* ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik Deutschland die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten *Wertewandel* zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über *mangelnden Zuspruch* nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

Step 2. Press <F4> to Begin Highlighting

F1 - Help Level 1 -- Vocabulary F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne *Erinnerung* ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik **Deutschland** die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten *Wertewandel* zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über *mangelnden Zuspruch* nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

Step 3. Highlight the Area Desired
Press <F4> to Complete Highlighting

F1 - Help | Level 1 -- Vocabulary | F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne *Erinnerung* ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik **Deutschland** die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten *Wertewandel* zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie *lobenswert*. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über *mangelnden Zuspruch* nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Editing a New Highlighted Word(s) Definition

_

Step 4. Type in the Definition

F1 - Help Level 1 -- Vocabulary F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne *Erinnerung* ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik **Deutschland** die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten *Wertewandel* zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie *lobenswert.* Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über *mangelnden Zuspruch* nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

Germany

Step 5. Completed Adding a Highlight and Definition

Adding a Multi-part Highlight and a Definition

F1 - Help | Level 1 -- Vocabulary | F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne **Erinnerung** ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik Deutschland die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten *Wertewandel* zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über *mangelnden Zuspruch* nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

memory

Step 1. Place cursor on First Character to Highlight

F1 - Help Level 1 -- Vocabulary F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne *Erinnerung* ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik Deutschland die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten *Wertewandel* zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über *mangelnden Zuspruch* nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

Step 2. Press <F4> to Begin Highlighting

F1 - Help Level 1 -- Vocabulary F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne *Erinnerung* ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik **Deutschland** die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten *Wertewandel* zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über *mangelnden Zuspruch* nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

Step 3. Highlight the Area Desired
Press <Insert> to Complete Highlighting of this part

F1 - Help Level 1 -- Vocabulary F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne *Erinnerung* ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik **Deutschland** die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten *Wertewandel* zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über *mangelnden Zuspruch* nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

Step 4. Reposition Cursor, then Press <Insert> to Begin Highlighting

F1 - Help Level 1 -- Vocabulary F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne *Erinnerung* ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik **Deutschland** die größte Schwerhörigkeit verzeichne zwischen den **Generationen**, das geringste Selbstbewußtsein der Menschen, den gründlichsten *Wertewandel* zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über *mangelnden Zuspruch* nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

Step 5. Highlight the Area Desired
Press <F4> to Complete Highlighting

F1 - Help | Level 1 -- Vocabulary | F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne *Erinnerung* ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik **Deutschland** die größte Schwerhörigkeit verzeichne zwischen den **Generationen**, das geringste Selbstbewußtsein der Menschen, den gründlichsten *Wertewandel* zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie *lobenswert*. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über *mangelnden Zuspruch* nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Editing a New Highlighted Word(s) Definition

_

Step 6. Type in the Definition

F1 - Help Level 1 -- Vocabulary F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne *Erinnerung* ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik **Deutschland** die größte Schwerhörigkeit verzeichne zwischen den **Generationen**, das geringste Selbstbewußtsein der Menschen, den gründlichsten *Wertewandel* zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie *lobenswert.* Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über *mangelnden Zuspruch* nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

Germany Generation

Step 7. Completed Adding a Multi-part Highlight and Definition

Creating a New Multiple Choice Question

F1 - Help | Level 1 -- Vocabulary | F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne *Erinnerung* ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik **Deutschland** die größte Schwerhörigkeit verzeichne zwischen den **Generationen**, das geringste Selbstbewußtsein der Menschen, den gründlichsten *Wertewandel* zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie *lobenswert.* Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über *mangelnden Zuspruch* nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

Germany Generation

Step 1. Text Prior to Pressing <F9>

F1 - Help | Level 1 -- Vocabulary | F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne Erinnerung ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik Deutschland die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten Wertewandel zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über mangelnden Zuspruch nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Editing a New Question

_

Step 2. Type in the Question

F1 - Help | Level 1 -- Vocabulary | F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne Erinnerung ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik Deutschland die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten Wertewandel zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über mangelnden Zuspruch nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

1. What is the theme of the article?
   a) The importance of history as national memory;
   b) The centrality of the Nazi period to German history;
   c) The responsibility of Germany for European stability.

Answer Type? (M)ultiple Choice, (Q)uestion/Answer, In (T)ext m

Step 3. Finish Editing the Question
Press <F5> to Determine Answer Type

F1 - Help | Level 1 -- Vocabulary | F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne Erinnerung ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik Deutschland die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten Wertewandel zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über mangelnden Zuspruch nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

1. What is the theme of the article?
   a) The importance of history as national memory;
   b) The centrality of the Nazi period to German history;
   c) The responsibility of Germany for European stability.

Step 4. Position the Cursor at the Beginning of the Answer
Press <F4> to Begin Highlighting the Answer

F1 - Help Level 1 -- Vocabulary F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne Erinnerung ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik Deutschland die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten Wertewandel zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über mangelnden Zuspruch nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

1. What is the theme of the article?
   a) **The importance of history as national memory;**
   b) The centrality of the Nazi period to German history;
   c) The responsibility of Germany for European stability.

Step 5. Continue Highlighting the Answer
Press <F4> to Finish Highlighting the Answer

Creating a New Question and Answer Question

F1 - Help | Level 1 -- Vocabulary | F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne *Erinnerung* ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik **Deutschland** die größte Schwerhörigkeit verzeichne zwischen den **Generationen**, das geringste Selbstbewußtsein der Menschen, den gründlichsten *Wertewandel* zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie *lobenswert.* Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über *mangelnden Zuspruch* nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

Germany Generation

Step 1. Text Prior to Pressing <F9>

F1 - Help | Level 1 -- Vocabulary | F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne Erinnerung ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik Deutschland die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten Wertewandel zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über mangelnden Zuspruch nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Editing a New Question

_

Step 2. Type in the Question

F1 - Help Level 1 -- Vocabulary F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne Erinnerung ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik Deutschland die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten Wertewandel zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über mangelnden Zuspruch nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

True or false? If false, what is true instead?

1. Museums are thriving.

Answer Type? (M)ultiple Choice, (Q)uestion/Answer, In (T)ext q

Step 3. Finish Editing the Question
Press <F5> to Determine Answer Type

F1 - Help Level 1 -- Vocabulary F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne Erinnerung ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik Deutschland die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten Wertewandel zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über mangelnden Zuspruch nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

True or false? If false, what is true instead?

1. Museums are thriving. _

Step 4. Position Cursor Where the Answer should Go
Type in the Answer

F1 - Help Level 1 -- Vocabulary F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne Erinnerung ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik Deutschland die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten Wertewandel zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über mangelnden Zuspruch nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

True or false? If false, what is true instead?

1. Museums are thriving. **T**

Step 5. Press <F5> to Finish Typing in the Answer

Creating a New Answer From Text Question

F1 - Help | Level 1 -- Vocabulary | F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne *Erinnerung* ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik **Deutschland** die größte Schwerhörigkeit verzeichne zwischen den **Generationen**, das geringste Selbstbewußtsein der Menschen, den gründlichsten *Wertewandel* zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie *lobenswert.* Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über *mangelnden Zuspruch* nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

Germany Generation

Step 1. Text Prior to Pressing <F9>

F1 - Help | Level 1 -- Vocabulary | F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne Erinnerung ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik Deutschland die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten Wertewandel zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über mangelnden Zuspruch nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Editing a New Question

_

Step 2. Type in the Question

F1 - Help Level 1 -- Vocabulary F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne Erinnerung ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik Deutschland die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten Wertewandel zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über mangelnden Zuspruch nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

1. Where is "across the country"?

Answer Type? (M)ultiple Choice, (Q)uestion/Answer, In (T)ext t

Step 3. Finish Editing the Question
Press <F5> to Determine Answer Type

F1 - Help | Level 1 -- Vocabulary | F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne Erinnerung ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik Deutschland die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten Wertewandel zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über mangelnden Zuspruch nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

1. Where is "across the country"?

Step 4. Position the Cursor at the Beginning of the Word in the Text
Press <F4> to Begin Highlighting the Answer

F1 - Help | Level 1 -- Vocabulary | F10 - Exit

Michael Stürmer

Geschichte in geschichtslosem Land

In einem Land ohne Erinnerung ist alles möglich. Die Meinungsforschung warnt, daß unter allen Industrieländern die Bundesrepublik Deutschland die größte Schwerhörigkeit verzeichne zwischen den Generationen, das geringste Selbstbewußtsein der Menschen, den gründlichsten Wertewandel zwischen ihnen. Wie werden die Deutschen morgen ihr Land, den Westen, sich selbst sehen? Es bleibt anzunehmen, daß die Kontinuität überwiegt. Aber sicher ist es nicht.

Landauf, landab registriert man die Wiederentdeckung der Geschichte und findet sie lobenswert. Museen sind in Blüte, Trödelmärkte leben von der Nostalgie nach alten Zeiten. Historische Ausstellungen haben über mangelnden Zuspruch nicht zu klagen, und geschichtliche Literatur, vor zwanzig Jahren peripher, wird wieder geschrieben und gelesen.

Es gibt zwei Deutungen dieser Suche nach der verlorenen

1. Where is "across the country"?

Step 4. Continue Highlighting
Press <F4> to Finish Highlighting the Answer