3 0 8 - 6 8 1 B

A P P L I C A T I O N P R O J E C T II

V O I C E / D I G I T A L A N D D I G I T A L / V O I C E C O N V E R S I O N O N A M I C R O C O M P U T E R (A P P L E II)

J O K T J I E Y A P

S E P T E M B E R , 1 9 8 7

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Nowadays, speech synthesis has shown up in personal computers, but the synthesis schemes in common use have limitations (in expense, vocabulary, or intelligibility) which have restricted their use. However, any personal computer which is equipped with an analog-to-digital (A/D) converter and a digital-to-analog (D/A) converter would theoretically be capable of both speech synthesis and recognition, if only it had the necessary software.

A practical, reliable and affordable method for computers to recognize and synthesis speech would yield at least two benefits to computer users:

(1) Maximum user friendliness - Speech is our most natural mode of communication.

(2) New applications - Speech interaction does not interfere with most human activities, especially those that require the use of the hands; therefore, speech interaction could allow computers to be used for tasks that are now impractical.

In this Voice/Digital and Digital/Voice Conversion on a Microcomputer project, the main objective is a set of routines in the Apple II UCSD Pascal System which digitize the sound input of spoken English words, store them in memory speech buffer (RAM or diskette) and then play them back on request. Therefore, in addition to the routines which perform the above tasks, a simple data base must be built to keep track of the digitized data and character strings of words. The basic operations of data base must be provided, namely: create dictionary, erase entire dictionary, entry retrieval keyed with word string, add entries, update entries, delete entries and periodical maintainance operations.

The name **dictionary** is used instead of data base in the documentation. Although the dictionary is the bookkeeper of sound and text, it does not have any knowledge of the encoding to the speech buffer and the decoding from the speech buffer. It just manages a warehouse of sounds and word strings on the diskette.

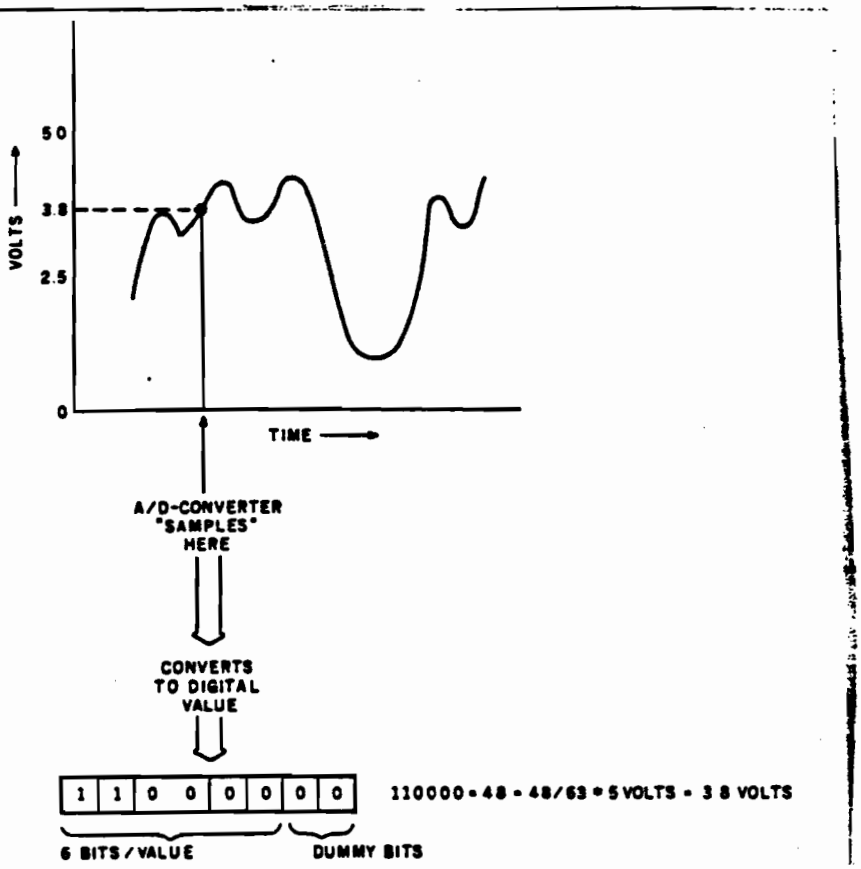


Figure 1: *An ADC converts an electrical analog, such as voltage, to a binary value.*

ADC: A/D Converter

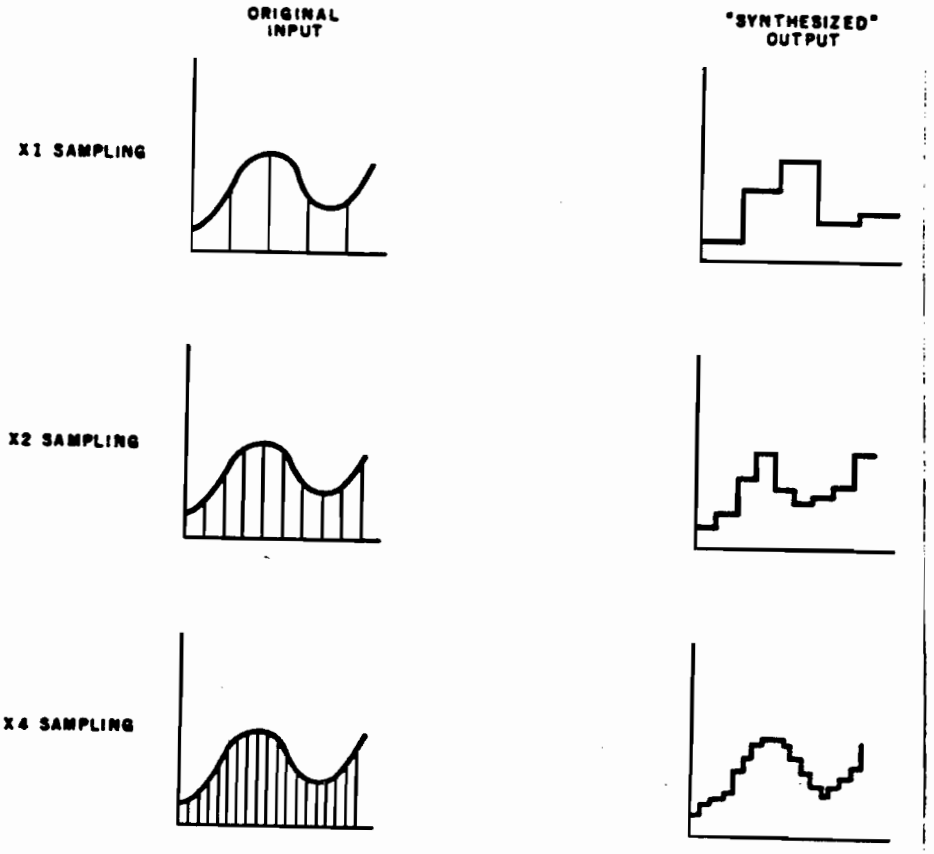


Figure 2: *The sampling rate and number of bits in the ADC determine how closely the input signal can be reproduced.*

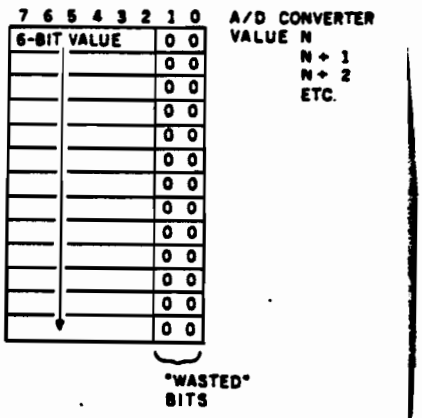


Figure 3: *Although 25 percent of the storage space is wasted in storing 6-bit ADC values in 8-bit bytes, it is efficient in terms of storage speed.*

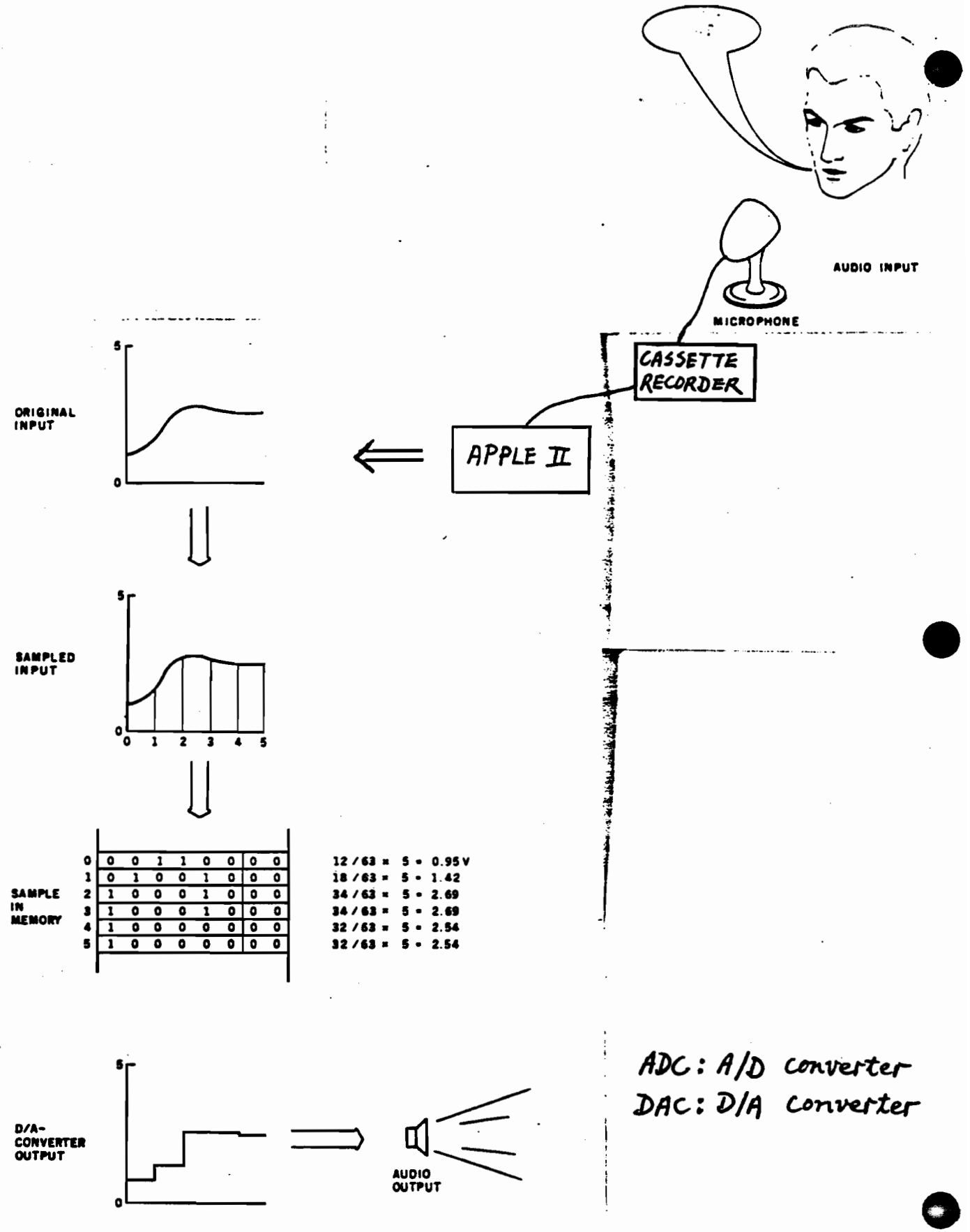


Figure 4: *Brute-force voice synthesis samples input to digitize it, stores the ADC values in memory, and then outputs the values from memory to a DAC.*

CHAPTER 2

THEORY AND DESIGN

2.1. BASIC THEORY

2.1.1. ANALOG TO DIGITAL AND REVERSE CONVERSIONS

Usually, an Analog_to_Digital converter (A/D) is used to convert the analog voice input signal to digital value form as illustrated in FIGURE 1 [REF 3]. The larger the number of bits in the sample, the finer the resolution in the digital representation of the analog value. If the A/D converter gives six bits of data, each digital value will be within 1/64 of the analog input value. A five bits A/D converter will give value within 1/32 of the analog input value, and so on. When replaying the digital form of the input, the output waveform will approximate the original waveform by a series of square steps. The higher the sampling rate and the resolution of the A/D converter, the more the output will resemble the original, as shown in FIGURE 2 [REF 3].

When a six bits A/D converter is used, for convenience and speed reasons, the six bits value is put in each byte and ignore the two unused bits, as shown in FIGURE 3 [REF 3]. With the sampling rate of 7000 Hz, one second of recorded sound will fill 7000 bytes of memory.

To play back the digitized sound, a Digital_to_Analog converter (D/A) is used. If the data was captured by a six bits A/D converter, then a six bits D/A converter is needed. It takes in each digital value and produces an output voltage level proportional to that value. A sequence of all these voltage levels will simulate an analog waveform.

The simple process of voice capture and synthesis as illustrated in FIGURE 4 [REF 3] and described above is a brute force method. It takes an analog voltage as input form from the audio source, samples it 7000 times or more per second with an A/D converter, stores the digitized A/D output values in the computer memory, and then plays back the values from memory with a D/A converter.

The process of using a A/D converter to record snatches of sound, initially held in a sound buffer, as a sequence of numbers is called speech encoding. The opposite process of converting a sequence of numbers in a sound buffer into audible sounds using a D/A converted is called speech decoding.

2.1.2. SAMPLING RATE

Human speech is much like music played from a complex instrument. It is composed of many different **notes** played simultaneously, interacting in complex ways. The voice is very versatile, it can be as percussive as a snare drum at a moment or as melodious as a flute just a fraction of a second later.

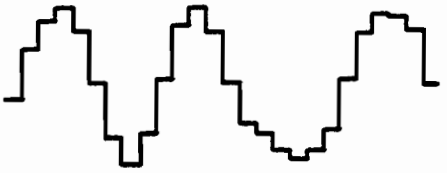
The frequencies of human speech can range from 20,000 Hz (cycles/sec) down to 10 Hz. The signal strength range is in the ratio of over 16,000 to 1, respectively, between a shout and a quiet whisper. The sampling rate must meet the Nyquist criteria [REF 5]. That is, the rate of sampling must be at least twice the frequency of the desired highest voice harmonic. To detect every nuance of all the sound in human speech, we would have to measure the sound signal over 40,000 times per second and store each measurement, using many bits of Apple II RAM. Exactly capturing one second of such speech would require 40K of RAM, this is more than the Apple II has. Even a floppy diskette would hold only four to five seconds of sound, and the data transmission rate could not keep up.

For reproduced speech sound to be acceptable and understandable, fortunately, it is not necessary to do it with that much precision. The telephone system only transmits sounds from 30 Hz to 3500 Hz, with volume ratios of less than 1000 to 1, and the clarity of voice suffers surprisingly little [REF 8]. It is true that people's voices do not sound the same in person as on the phone, but it is usually easy to recognize a familiar voice over the phone and the speech is easily understood. What constitutes acceptable speech quality is a subjective judgement on the part of the listener.

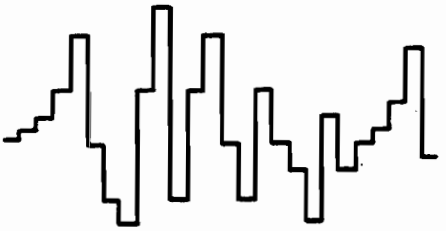
From the discussion in the above paragraphs, a conclusion can be drawn that in order to reproduce acceptable voice, capability of playing back frequencies up to 3500 Hz is needed and sampling rate must be at least twice the maximum frequency to be recorded. Hence voice must be

(a) *Analog speech waveform fed as input to the analog-to-digital converter.*

(b) *Speech waveform digitized by simple pulse-code modulation.*



(c) *Speech waveform digitized by differential pulse-code modulation.*



(d) *Speech waveform digitized by adaptive differential pulse-code modulation.*

Figure 5

recorded at rates of 7000 Hz or better. In other words, the voice input must be converted to digital form at a rate of 7000 samples per second or better.

At the higher sampling rate of an expensive A/D converter, the computer memory required to store the digital data exceeds the memory of the Apple II computer after only a few seconds of actual operation.

2.1.3. SPEECH ENCODING METHODS

The methos of encoding speech fall into two broad catagories: time domain and frequency domain approach.

Time domain encoding approach seeks to measure and record the amplitudes of the sound waveforms, which vary in time, and to reconstruct the speech waveforms from their recorded history. All the three most popular techniques of time domain encoding measure the exact amplitude of the speech waveform frequently enough so that when it is played back, a reasonable facsimile of the original wave results:

(1) PCM - Pulse Code Modulation. This requires no special knowledge about speech signal, except the bandwidth. It uses the most memory for a given amount of signal, but it is the simplest. In fact, virtually no computing at all is required beyond the storage and retrieval of the data.

(2) DPCM - Differential Pulse Code Modulation. This is a variation of PCM that utilizes the fact that speech signals are not random but are closely related to sinusoidal functions. Thus, at each successive sample time the signal value is not random. It is likely to be relatively close to its preceding value. Therefore, DPCM stores a value indicating the change from the last recorded sample, rather than a value representing the sample itself, it has the advantage in efficiency of storage.

(3) ADPCM - Adaptive Differential Pulse Code Modulation. This is a further development that cures the Slope Overloading defect of DPCM. Optimized for speech storage, ADPCM offers better compliance with the input waveform and better intelligibility of the reproduced voice signal at lower data rate [REF 4].

A speech sample before encoding is shown in FIGURE 5a [REF 4]. The FIGURE 5b, 5c and 5d [REF 4] show the speech sample after

encoding, respectively, using PCM, DPCM and ADPCM techniques.

The comparison of FIGURE 5c and 5d shows larger errors for ADPCM encoding method. This is not indicative of actual results achievable in a system set up specifically for ADCPM method. Usually, the sampling rate is two to four times what PCM or DCPM encodings might use, but the storage required is still only 25 to 50 percent more with comparable sampling errors. One further problem: ADPCM is characterized by high frequency error noise which must be filtered out to produce acceptable quality speech reproduction [REF 4].

Frequency domain encoding approach seeks to measure the frequencies present in a voice waveform and how they vary through time in distribution and in amplitude. Typically, if three or four dominant pitches are recorded, acceptable speech can be reproduced. The most popular technique of of this approach is Linear Predictive Coding (LPC), which is used in the Texas Instruments Speak & Spell toy.

LPC is the most compact encoding. Only 300 bytes are required to store one second of speech. However, the price in computational complexity for LPC is prohibitive in small scale use. Converting a few seconds of PCM speech to LPC or the reverse conversion can take several minutes on a large computer without the aid of a dedicated VLSI circuit [REF 4].

2.1.4. A/D AND D/A CONVERTERS OF APPLE II

Apple II computer has a cassette input port, which can be utilized to digitize the voice input which comes from a standard cassette recorder. The principle is only monitoring the zero crossing of any analog signal, including voice. We make a recording of the signal on a cassette tape, then plug the recorder into the Apple cassette input port and play the tape back while monitoring memory location C060 (hexadecimal), bit seven. Above a certain level of input signal, the cassette input port will read high; below this level it will read low. The level of this transition is called the threshold.

Each time the signal crosses the threshold, the state of the cassette input bit changes. A history of these changes contains all the frequency characteristics of the original analog input, assuming two thresholds per cycle.

There is a good reason that the indirect voice input from a cassette recorder is prefered over the direct microphone input. An ideal cassette recorder should have a volume and a tone control so that the user can input the correct amplitude and control the frequency range. In this way, the given voice can be reproduced with minimal noise and the mid-point of the input signal's peak to peak amplitude can be set at the point where the Apple threshold occurs. For each individual, the quality of speech recorded will vary a lot with changes in these parameters. The only way to find out the correct setting is by experimentation.

Two things should be noted:

(1) Since this method of data gathering really amounts to a one bit A/D conversion, amplitude information is not present and the stored voice will be reproduced at a constant volume depending on the hardware used for the actual playback.

(2) The sampling rate must meet the Nyquist criterion. That is, the rate at which the cassette input port is read must be at least twice the frequency of the highest voice harmonic that we wish to store.

The on board speaker of Apple II computer can be toggled by a read or write to memory location C030 (hexadecimal). A **click** will be produced on the output speaker every time memory location C030 is toggled. A large amount of clicks can reconstruct the human voice, but the reproduced voice will not have any tones or amplitudes related to the original voice input.

It may be difficult to image how these clicks can reconstruct the human voice. Try not to think about the individual click, but think instead of a series of clicks being output at a varying frequency which is a function of the original voice input. This varying frequency is an FM (frequency modulated) reproduction of that original input signal. The astonishing thing is that this crude method works fairly well. It is actually a special case of PCM encoding approach.

2.1.5. PACKING AND UNPACKING OF A/D DATA

By using the Pulse Code Modulation (PCM) technique, if each eight bits of A/D data is stored into each memory byte location, the whole RAM of Apple II can only handle a few seconds of digitized data. Therefore, the DPCM encoding approach is chosen by utilizing the method of

packing and unpacking A/D data described in this section.

The data which is stored does not always change from sample to sample. This will always happen when low frequency signals are being input and also during the times when no input is being sensed at the cassette input port. Therefore data can be stored such that the first bit (leftmost bit) of each memory location (byte) represents the state of the A/D input and the next seven bits represents a counter indicating the number of samples collected while that input remained unchanged. If the counter overflowed the allocated seven bits, the same A/D state bit would be stored in the next memory location with a new counter value. This solution will fill up RAM of Apple II with one to three minutes of reasonably understandable human voice.

In the processes of storing and playing back the voice, the time is a function of voice pitch and the setting of the input on the tape recorder. Certain voice sound such as **ssss** and **sh** tend to cause a great deal of change in the A/D data relative to the sampling rate, with the result of less voice recording time for a given amount of computer memory.

In the unpacking of A/D data byte, when the logical AND operator is applied to the data byte with X'80' value as the second operand, the result is the state of the original A/D input. On the other hand, the counter value for the total number of samples is obtained by the same logical AND operator, but with X'7F' value as the second operator.

The software routine which does this unpacking will decrement the counter by one in each loop of program flow. The output speaker will be toggled each time that the data counter reaches zero and the A/D bit changes state. This amounts to producing a **click** on the output speaker each time the original input voice signal had a zero crossing.

2.1.6. NOISE DURING PLAYBACK

Although bad quality of the onboard speaker contributes noise during the analog signal reproduction of voice, the method of sampling and reproducing the data is the main source of noise. It is important to sample the A/D converter at a constant rate and to make the corresponding D/A conversion at the same uniform rate. Any difference in these rates will cause a high noise level which will have to be filtered [REF 6].

The assembly language routines which are written to sample or reproduce the input data both have diffrent logical paths to follow. Take the routine of input data sampling (DIGITAL). Its logical paths test whether the counter is being incremented or whether the counter plus the data bit are being stored, and whether the least or most significant bit of the storage location has to be incremented. Each case requires a different number of machine cycles to complete and thus affects the time required to go back to read the cassette input port.

The solution is to make each logical path use the same number of machine cycles, by utilizing various delay cycles (which do nothing) on all logical paths except the slowest path. This leads to a lower sample rate, thus reducing the bandwidth of the input signal making voice recognition more difficult.

There are two possible methods to reduce this bad effect :

(1) Digital filtering techniques.

(2) Hardware filters of the cassette recorder.

The first technique is applied to stored data to remove the noise. It is rejected in this project, because it involves the designing of a bandpass filter of high complexity. This type of filter requires the use of complex multiplying coefficients and is not practical for real_time Apple II microcomputer operations on large amount of data.

In the second technique, the reproduced analog data is sent to the cassette recorder to be either recorded on tape or output in the PA (public address) mode. To achieve this purpose, the memory location C020 (cassette output) is toggled instead of C030 (Apple Speaker). The cassette recorder tone control is used to filter out unwanted noise. The result is quite satisfactory.

2.2. DESIGN INTENT

The dictionary is contained in a diskette with defined volumename STORE:. It consists of two types of data files, namely, index data file and binary data file. Any word which is stored in the dictionary has two data records, one record in each type of data file.

Each data record in the binary data file is an array of byte strings and it is called binary sound data record. The array's size is five and the capacity of each byte string is 255. The content of this record is the binary data of the digitized sound of the associated word.

In the index data file, each data record consists of four elements, which are:

(1) The character string of a word.

(2) The number of sound units. This tells the total number of elements (out of a maximum of five) of each binary data record which holds the digitized sound data.

(3) A value which is used to access the binary data record from the associated binary data file directly. Actually, this is the record number of the binary data record in the binary data file.

(4) Status element of boolean value. This is always TRUE for a word which exists in the dictionary. After a word has been deleted from the dictionary, the value is FALSE.

All the words in the dictionary are separated into three sets and each set of words is associated with a distinct set of starting characters. This arrangement makes the sequential search of words in the dictionary easier by reducing the maximum number of words to be searched to one third of all the words in the dictionary. Therefore, six data files are needed, three each for the index and binary data files.

In the sample dictionary, all the words are formed by upper case alphabet letter. The three sets of starting characters are: ['A'..'H'], ['I'..'P'] and ['Q'..'Z']. The index records in each index data file are keyed by the word string element in alphabetical ascending order. For the first, second and third set of the above starting characters, respectively, the names of the associated pairs of index and binary data files are:

(1) INDEX1.DATA and BINARY1.DATA

(2) INDEX2.DATA and BINARY2.DATA

(3) INDEX3.DATA and BINARY3.DATA

For each pair of associated index and binary data files in a new dictionary, the record numbers of a pair of index and binary data records for a word entry in their respectively data files are the same, but after one or more additions of entries in the dictionary, the similarity in the record numbers is not valid anymore. The addition of index records is

performed as insertion in various part of the index data files, in order to keep the word string elements in alphabetical ascending order and preserve the validity of sequential search on the index data records. Unlike index data records, the addition of binary data records can be placed at the end of the associated binary data files. The reason is that each index data record always contains the record number of its associated binary data record.

For the character string of a word, after its existence in the dictionary has been verified, the process of accessing the binary data record in order to reproduce the sound of this word requires the following three information. They are all elements of a record type called ELEM:

(1) The number of sound units.

(2) The record number of the binary data record.

(3) An index points to the exact binary data file which contains the binary data record.

The values of the first and second elements of an ELEM type record are obtained from the second and third elements of the associated index data record respectively. The third element is determined by the starting character of word string, the value is either one, two or three depending whether it is an element of the first, second or third set of starting characters.

The documentation on the interface part (GLOBAL unit) gives more insight on the data structure of this project.

CHAPTER 3

USER MANUAL

3.1 PROCEDURES OF THIS PROJECT

All the Pascal language procedures are grouped in four UNITs : UTILITY, MODULE1, MODULE2 and MODULE3.

In this section, the Pascal procedures which are available to the user who USES the units are listed. For more description of each procedure, please refer to section 3.6.

There are four procedures inside MODULE1 unit :

(1) BLD_DIRECTORY - builds all index data files.
(2) PRT_ENTRIES - displays the content of index data records in all the index data files. An option is available to display only one specified index data file.
(3) BLD_VOICE - builds all binary data files.
(4) CLR_DIRECTORY - deletes all the index and binary data files in the storage diskette. In other words, the whole dictionary is cleared.

There are seven procedures inside MODULE2 unit :

(1) ADD_XENTRIES - builds temporary index data files to store the new addition of index data records. This is the first part of adding new index data records into the dictionary.
(2) CMB_XENTRIES - combines (sort merges) each temporary index data file to its related permanent index data file. This is the second and last part of adding new index data records into the dictionary.
(3) ADD_BENTRIES - builds temporary binary data files to store the new addition of binary data records. This is the first part of adding new binary data records into the dictionary.
(4) CMB_BENTRIES - combines (sort merges) each temporary binary data file to its related permanent binary data file. This is the second and last part of adding new binary data records into the dictionary.
(5) DO_DELETE - deletes entries from the dictionary in the storage diskette. Each entry is a pair of index and binary data records, respectively, from the related index and binary data files. The entries are not removed phy_ sically, only the STATUS element of each entry is set to FALSE value.
(6) DO_CLNUP - periodical cleanup of the dictionary, all the deleted entries with FALSE value in the STATUS elements are removed permanently.
(7) IMPROVE_SOUND - improve or update the binary sound data of existing words in the dictionary.

The MODULE3 unit has three procedures :

(1) SPEAK - speaks a word, its binary sound data has already been placed in the global VOICE buffer. The sound ouput is at the connected speaker of Apple II or casette recorder. The choices of speaker and analog delay constant have already been done.

(2) SPEAK_WORD - speaks a specified word which is passed as parameter. The sound output speaker and analog delay constant should have been chosen.

(3) LISTEN - listens to a word and get the binary sound data from the con_ nected casette recorder. The global VOICE buffer is used to store the result of binary sound data.

The UTILITY unit has ten procedures :

(1) DO6_RESET - opens all six index and binary data files.

(2) DO6_CLOSE - closes all six index and binary data files.

(3) DISKETTE_ONLINE - verifies and requests that the storage diskette online by creating or opening a specified data file.

(4) CNT1_ELEM - counts the total number of data records in a specified index data file.

(5) CNT2_ELEM - counts the total number of data records in a specified binary data file.

(6) BLDIDX - builds all the index data records of a specified index data file.

(7) GETVOICE - builds all the binary data records of a specified binary data file.

) GET_WORDUNIT - gets a word string and its number of sound units from the user interactively.

(9) WORD_VERIFY - verifies that a specified word exists in the dictionary. When the existence has been confirmed, the information to access the bi_ nary data record is returned. This procedure can be used to update the number of sound units of a word when the special option is chosen.

(10) FILE_SORT - processes file sort merge on two index data files. Sorting is on the alphabetical ascending order of the word element in each index data record.

The three Assembly language routines (external procedures) which process the sound data directly reside in the system library called SYSTEM.LIBRARY. In order to call any one of them from a host program, the related declaration of the following three lines must appear right after the end of VAR declara_ tion section :

```
PROCEDURE DIGITAL ( VAR BDATA:SOUND; TEMPO, UNITT:INTEGER ) ; EXTERNAL ;
PROCEDURE ANALOG1 ( VAR BDATA:SOUND; TEMPO, UNITT:INTEGER ) ; EXTERNAL ;
PROCEDURE ANALOG2 ( VAR BDATA:SOUND; TEMPO, UNITT:INTEGER ) ; EXTERNAL ;
```

For the example of preceding declaration, please check the listing of

MODULE3 unit in section 3.6.

It is not necessary to use the three external procedures mentioned above in a host program, the equivalent Pascal language procedures are available in MODULE3 unit :

(1) Procedure LISTEN - for the DIGITAL external procedure.
(2) Procedure SPEAK - for the ANALOG1 and ANALOG2 external procedures.

3.2 INTERFACE PART OF THE PROJECT

In order to make INTERFACE part of this Voice/Digital and Digital/Voice Conversion on a Microcomputer easily used, it is defined as a UNIT of Apple Pascal System and called GLOBAL. All the global constants, types and variables are defined in here, the default values of some global variables are assigned. The CODE file of this GLOBAL unit is placed in the System Library of routines called SYSTEM.LIBRARY, so any Pascal host program which has a declaration using this GLOBAL unit can access and manipulate the content of this INTERFACE part.

Placing the interface part in one unit lets the user changes the values of global constants, data types and variables without having the require_ ment to modify any unit of V/D and D/V Conversion on a Microcomputer. After the modification and successful compilation of this GLOBAL unit, the user only needs to link the codes files of all units with the new code file of this GLOBAL unit.

Each one of the four UNITs (UTILITY, MODULE1, MODULE2 and MODULE3) of this project has a declaration using the GLOBAL unit at the beginning of the program.

```
CONSTANT

  MAXFILE = 3 ;
  MAXUNIT = 5 ;
  UNITSIZE = 255 ;
  BUFFUNIT = 20 ;
  MAXCHAR = 20 ;
  MAXWORD = 25 ;
  TOTALWORD = 100 ;
  FILENMLEN = 40 ;
  VOLNMLEN = 8 ;
```

MAXFILE - maximum number of index data files or binary data files to store all the available words of dictionary in this project.

MAXUNIT - maximum number of sound units of each word.

UNITSIZE - the total number of bytes for each sound unit.

BUFFUNIT - the total number of sound units of a buffer which is used in testing the sound input and output of several words.

MAXCHAR - maximum number of characters in each word.

MAXWORD - maximum number of words in each sentence.

TOTALWORD - the capacity of each index or binary data file.

FILENMLEN - the maximum length of a filename character string.

VOLNMLEN - the maximum length of a volumename character string, including the colon character.

```
TYPE

  WORDRANGE = 1..MAXWORD ;
  TWORDRANGE = 0..TOTALWORD ;
  FILERANGE = 1..MAXFILE ;
  UNITRANGE = 1..MAXUNIT ;
  SPEAKEROF = (CASETTE,APPLE) ;
  WORD = STRING[MAXCHAR] ;
  FILENAME = STRING[FILENMLEN] ;
  VOLNAME = STRING[VOLNMLEN] ;
  IDXELEM = PACKED RECORD
                      STRG : WORD ;
                      UNITT : UNITRANGE ;
                      IDXX : TWORDRANGE ;
                      STATUS : BOOLEAN
                    END ;
  ELEM = PACKED RECORD
                   WUNIT : UNITRANGE ;
                   WIDX : TWORDRANGE ;
                   WSET : FILERANGE
                 END ;
  SOUND = ARRAY[UNITRANGE] OF STRING[UNITSIZE] ;
  SENTENCE = ARRAY[WORDRANGE] OF WORD ;
  ELEMARRAY = ARRAY[WORDRANGE] OF ELEM ;
  INDEXFILE = FILE OF IDXELEM ;
  BINARYFILE = FILE OF SOUND ;
  STRGFILE = FILE OF STRING ;
```

WORDRANGE - the range of possible number of words in a sentence.

TWORDRANGE - the range of possible record number of a binary data file or the range of possible number of index data records in an index data file.

FILERANGE - the range of total number of index data files or binary data files in the storage diskette.

UNITRANGE - the range of possible number of sound units in a word.

SPEAKEROF - the possible speaker destination for the output sound.

WORD - defines the character string size of a word.

FILENAME - defines the character string size of a filename.

VOLNAME - defines the character string size of a volumename for the storage diskette.

IDXELEM - this is the record type of each index data record in the index data file. The first element is the character string of a word, the second element is the number of sound units of this word. The binary sound data is in one of the three binary data files, the third element is the index number (record number) of this binary data record in the associated binary data file. The existing status of this word in the dictionary is in the STATUS (last) element of this structured record type.

ELEM - it defines the record type of each element in the ELEMARRAY array type. The first element of the record is the number of sound units of a word and the second element is the index number (record number) of the binary data record in one of the three binary data files. The WSET (last) element tells which one of the three binary data files has the binary data record.

SOUND - it defines the maximum size of binary sound data that a word can have in the dictionary.

SENTENCE - defines the maximum number of words in a sentence.

ELEMARRAY - each array of this data type associates with a sentence. Each element of this array corresponds to a word from the sentence, it contains all the required information in order to access the binary data record from a binary data file.

INDEXFILE - file type of index data file.

BINARYFILE - file type of binary data file.

STRGFILE - file type of character string data file.

VARIABLE

```
  IFILE1, IFILE2, IFILE3 : INDEXFILE ;
```

```
  BFILE1, BFILE2, BFILE3 : BINARYFILE ;
  YFILE : STRGFILE ;
  VOICE : SOUND ;
  SPKER : SPEAKEROF ;
  DTEMPO, ATEMPO : INTEGER ;
  I1FNAME, I2FNAME, I3FNAME : FILENAME ;
  B1FNAME, B2FNAME, B3FNAME : FILENAME ;
  YFNAME : FILENAME ;
  SET1CHR, SET2CHR, SET3CHR, SET4CHR : SET OF CHAR ;
  SET1STCHR, SETCHRS : SET OF CHAR ;
  VOLUMENAME : VOLNAME ;
  CHRS1, CHRS2, CHRS3, CHRS4, CHRSALL : FILENAME ;
  VOICEBUFF : ARRAY[1..BUFFUNIT] OF STRING[UNITSIZE] ;
  LENWORDS : ARRAY[1..BUFFUNIT] OF UNITRANGE ;
  TOTWORDS : INTEGER ;
```

IFILE1, IFILE2, IFILE3 - file window variables for all three index data files.

BFILE1, BFILE2, BFILE3 - file window variables for all three binary data files.

YFILE - file window variable for a data file of character string records.

VOICE - buffer of binary sound data of a word. It is used for the purpose of either speaking or listening a word.

SPKER - it contains the choice of voice output on the speaker of Apple II or a casette recorder. The default value is Apple II's speaker.

DTEMPO - it contains the delay constant in the process of digitizing the input sound to get the binary sound data. The default value is one.

ATEMPO - it contains the delay constant in the process of analoging the binary sound data into the output sound frequency. The default value is four.

I1FNAME, I2FNAME, I3FNAME - filename variables of all three index data files.

B1FNAME, B2FNAME, B3FNAME - filename variables of all three binary data files.

YFNAME - filename variable of a character string data file.

SET1CHR, SET2CHR, SET3CHR - sets of valid starting characters, respectively, for the words in the first, second and third index data files.

SET4CHR - a set of digit characters for the valid total numbers of sound units of a word.

SET1STCHR - a set of all the valid starting characters of the words in the dictionary.

SETCHRS - it contains the current set of starting characters.

VOLUMENAME - it has the volumename character string of the data storage dis_ kette, the default value is STORE: .

CHRS1, CHRS2, CHRS3, CHRS4, CHRSALL - valid starting characters message strings, respectively, for SET1CHR, SET2CHR, SET3CHR, SET4CHR and SET1STCHR sets of characters.

VOICEBUFF - a big buffer to store the binary sound data of several words, for the testing purpose of either sound input or output.

LENWORDS - each element of this array contains the number of sound units of the associated word's binary sound data in the VOICEBUFF buffer.

TOTWORDS - total number of words which have the binary sound data in the voiceBuff buffer.

3.3 PROCESSING

3.3.1 AVAILABLE OPERATIONS AND RELATED UNITS

In order to do any one of the available operations, the calling Pascal language host program must have the following statement before the LABEL or CONSTANT definition section to indicate which UNITs of the five are used :

USES GLOBAL, [UTILITY], MODULEx [, MODULEy, MODULEz] ;

the ending letter 'x' of MODULEx is either digit characters 1, 2 or 3. It is similar for ending letters 'y' and 'z' in MODULEy and MODULEz respectively. The UTILITY unit is used only when at least one of its procedures is needed by the host program in the processing.

There are three operations involving speaking and listening which are grouped in MODULE3 unit, therefore, for one or more of these operations, the host program must have the following statement :

USES GLOBAL, UTILITY, MODULE3 ;

The three operations are :

(1) Basic speak - call procedure SPEAK
(2) Speak a specified word - call procedure SPEAK_WORD
(3) Basic listen - call procedure LISTEN

The UTILITY unit is used because its DO6_RESET and DO6_CLOSE procedures are called by the host program to open and close all index and binary data files. If the host program opens and closes each data file directly by calling the build in file I/O procedures RESET and CLOSE respectively, the UTILITY unit is not needed.

The operations which work on the database of words or dictionary are :

(1) Create a new dictionary - call procedures BLD_DIRECTORY and BLD_VOICE.
    USES GLOBAL, MODULE1 ;
(2) Purge the whole dictionary - call procedure CLR_DICTIONARY.
    USES GLOBAL, MODULE1 ;
(3) Display content of all index data files (index data records) - call procedure PRT_ENTRIES.
    USES GLOBAL, MODULE1 ;
(4) Add new entries to the dictionary - call procedures ADD_XENTRIES, ADD_BENTRIES, CMB_XENTRIES and CMB_BENTRIES.
    USES GLOBAL, MODULE2 ;
(5) Delete entries from the dictionary - call procedure DO_DELETE and DO_CLNUP.
    USES GLOBAL, MODULE2 ;
    For a speedy deletion only procedure DO_DELETE is required. The physical removal of the deleted records can be carried out later with the periodical maintainance procedure DO_CLNUP.
(6) Periodical maintainance on the dictionary - call procedure DO_CLNUP.
    USES GLOBAL, MODULE2 ;

It does the physical removal of deleted records (index and binary data records) from the dictionary. These data records have been de_ leted previously with the SPEEDY DELETION option.

(7) Update binary data records - call procedure IMPROVE_SOUND.
USES GLOBAL, MODULE1 ;

For any one of the above seven operations, one or more procedures of UTILITY unit might be called but the related unit of the operation already has an USES statement with UTILITY unit, therefore the host pro_ gram does not need to USES the UTILITY unit. it is assumed that the host program does not call any procedure of UTILITY unit, otherwise the USES statement must include the UTILITY unit.

There are two sample programs in the following pages, the first sample shows the operations of MODULE1 and MODULE2 units and the second sample shows the operations of MODULE3 unit.

3.2 PROGRAMMING EXAMPLES OF ALL AVAILABLE OPERATIONS

```
(* ------------------------------------------------------------------ *)
   This is the example of using SPEAK, SPEAK_WORD and LISTEN procedures *)
(* of MODULE3 module.                                                 *)
(*                                                                    *)
(* Several procedures are built, each of them either calls SPEAK,     *)
(* SPEAK_WORD or LISTEN to complete the task, some procedures of      *)
(* UTILITY module are called during the processing.                   *)
(*                                                                    *)
(* The main program must call DO6_RESET and DO6_CLOSE at the beginning *)
(* and at the end of the program. These two procedures are from the   *)
(* UTILITY module, opening and closing all six index and binary data  *)
(* files respectively.                                                *)
(*                                                                    *)
(* This example USES three modules :                                  *)
(*   GLOBAL - must be USED by all application program                 *)
(*   MODULE3 - this is the example of using it                        *)
(*   UTILITY - several procedures of this module are called           *)
(*                                                                    *)
(* The compiler option $S++ is used in this example, it slows down the *)
(* speed of compilation but more memory space is available.           *)
(*                                                                    *)
(* The following global variables from the GLOBAL module are used in  *)
(* this example :                                                     *)
(*   VOICE - storing buffer of the digitized voice data of a word     *)
(*   ATEMPO - analoging delay constant, used in SPEAKing procedure    *)
(*   DTEMPO - digitizing delay constant, used in LISTENing procedure  *)
     SPKER - for chosing the speaker of Apple II or cassette recorder *)
             in speaking (playback the voice of) words                *)
(* ------------------------------------------------------------------ *)

PROGRAM USER_MANUAL_MODULE3 (INPUT, OUTPUT) ;

(*$S++*)              (* compiler option *)

USES GLOBAL, UTILITY, MODULE3 ;          (* this example only uses *)
                                         (* these three modules    *)

VAR WRD : WORD ;                    (* variables of the main program *)
    STRG : STRING ;
    FN : FILENAME ;
    CONFIRM : BOOLEAN ;

(* ------------------------------------------------------------------ *)
(* This procedure speaks a word. The character string of the word to be *)
(* spoken is passed in the parameter WRD.                             *)
(*                                                                    *)
(* Input :                                                            *)
(*   WRD - parameter, word to be spoken.                              *)
(*                                                                    *)
(* Output :                                                           *)
(*   Sound of the word which is output on the speaker of Apple II or  *)
(*   attached casette recorder depending on the value of the global   *)
     variable SPKER.                                                  *)
(*                                                                    *)
(* The following procedures are called :                              *)
```

```
(*   WORD_VERIFY - module UTILITY, verifying whether the passing para_  *)
(*     meter WRD is in the dictionary. When it exists, return all       *)
(*     required information in FNDREC parameter in order to access the  *)
(*     binary sound data from the related binary data file.             *)
(*   SPEAK_WORD - module MODULE3, speaks the word in WRD                *)
(*   LENGTH - string built in function, get the length of the character *)
(*            string                                                    *)
(*                                                                      *)
(* The following global variables of the GLOBAL unit are used :         *)
(*   None                                                               *)
(* -------------------------------------------------------------------- *)

PROCEDURE WORD_SPEAK ( WRD : WORD ) ;

VAR FOUND, CHANGE : BOOLEAN ;
    L : INTEGER ;
    FNDREC : ELEM ;

BEGIN

  L := LENGTH( WRD ) ;       (* process only when the word is not NULL *)

  IF ( L > 0 ) THEN
  BEGIN

    CHANGE := FALSE ;
    FOUND := FALSE ;         (* regular verification of word *)

    WORD_VERIFY(CHANGE, FOUND, WRD, FNDREC) ;       (* verify existence  *)
                                                    (* of word & get all *)
                                                    (* needed information*)
    IF ( FOUND ) THEN
      SPEAK_WORD ( FNDREC )                         (* exist, speak it *)
    ELSE
      WRITELN('  WORD ':7,WRD:L,' IS NOT IN THE DICTIONARY')

  END

END ;


(* -------------------------------------------------------------------- *)
(* This procedure speaks a sentence. The character string of the sen_   *)
(* tence to be spoken is passed in the parameter STRG.                  *)
(*                                                                      *)
(* Input :                                                              *)
(*   STRG - parameter, sentence to be spoken.                           *)
(*                                                                      *)
(* Output :                                                             *)
(*   Sound of the sentence which is output on the speaker of Apple II   *)
(*   or attached casette recorder depending on the value of the global  *)
(*   variable SPKER.                                                    *)
(*                                                                      *)
(* The following procedures are called :                                *)
(*   ANALYZE_SENTENCE - assume this procedure exists, it gets all the   *)
```

```
(*     words of a sentence and places them in a word buffer WBUFF.      *)
(*   WORD_VERIFY - module UTILITY, verifying whether the passing para_  *)
(*     meter WRD is in the dictionary. When it exists, return all       *)
(*     required information in FNDREC parameter in order to access the  *)
(*     binary sound data from the related binary data file.             *)
(*   SPEAK_WORD - module MODULE3, speaks the word in WRD                *)
(*   LENGTH - string built in function, get the length of the character *)
(*            string                                                    *)
(*                                                                      *)
(* The following global variables of the GLOBAL unit are used :         *)
(*   None                                                               *)
(* -------------------------------------------------------------------- *)

PROCEDURE SPEAK_STRING ( STRG : STRING ) ;

VAR WBUFF : SENTENCE ;
    IDX, I, L, TOT : INTEGER ;
    WRD : WORD ;
    EOS, CHANGE, FOUND : BOOLEAN ;
    FNDREC : ELEM ;
    EARRAY : ELEMARRAY ;

BEGIN

  TOT := 0 ;

  ANALYZE_SENTENCE(STRG,WBUFF,IDX) ;          (* get all the words *)
                                              (* of the sentence   *)
  CHANGE := FALSE ;          (* regular verification of word *)

  (* Loop, verifies all the words in WBUFF buffer. Processes only one  *)
  (* word in each pass of the loop. The IDX variable is the total num_ *)
  (* ber of words in the sentence (WBUFF buffer).                      *)

  FOR I := 1 TO IDX DO
  BEGIN

    WRD := WBUFF[I] ;
    FOUND := FALSE ;

    WORD_VERIFY( CHANGE, FOUND, WRD, FNDREC ) ;     (* verify each word *)

    IF ( FOUND ) THEN              (* only keep & process those words *)
    BEGIN                          (* which exist in the dictionary.  *)
      TOT := TOT + 1 ;             (* TOT is the total number of words*)
      EARRAY[TOT] := FNDREC        (* which exists in the dictionary. *)
    END
    ELSE
    BEGIN
      L := LENGTH(WRD) ;
      WRITELN('> WORD ':7,WRD:L,' IS NOT IN THE DICTIONARY')
    END

  END ;
```

```
  IF ( TOT > 0 ) THEN          (* speak only when the sentence has *)
  BEGIN                        (* one or more valid words.         *)
    FOR I := 1 TO TOT DO
    BEGIN                            (* speak each word separately *)
      FNDREC := EARRAY[I] ;
      SPEAK_WORD( FNDREC )           (* speak the word *)
    END
  END

END ;


(* ------------------------------------------------------------------ *)
(* This procedure speaks all the sentences of a text data file. The   *)
(* name of the text file is passed in the parameter FN. The parameter *)
(* must be declared as VARiable, otherwise the file window pointer    *)
(* of this file can not be advanced by the GET procedure.             *)
(*                                                                    *)
(* A volume name should accompany the filename, otherwise the boot    *)
(* diskette is assumed to contain the text data file.                 *)
(*                                                                    *)
(* Input :                                                            *)
(*   FN - parameter, data file of sentences to be spoken.             *)
(*                                                                    *)
(* Output :                                                           *)
(*   Sound of the sentences which are output on the speaker of Apple II *)
(*   or attached casette recorder depending on the value of the global *)
(*   variable SPKER.                                                  *)
(*                                                                    *)
(* The following procedures are called :                              *)
(*   CLOSE - build in file I/O procedures, open an existing data file. *)
(*   SPEAK_STRING - defined procedure in this example program, speaks a *)
(*                  sentence.                                         *)
(*   GET - build in file I/O procedure, get next element from the file *)
(*   CLOSE - build in file I/O procedures, close the file             *)
(*                                                                    *)
(* The following global variables of the GLOBAL unit are used :       *)
(*   YFILE - file window variable of a data file of character strings. *)
(* ------------------------------------------------------------------ *)

PROCEDURE SPEAK_FILE ( VAR FN : FILENAME ) ;

VAR L, IDX : INTEGER ;
    STRG : STRING ;
    EOS : BOOLEAN ;

BEGIN

  RESET( FN, YFILE ) ;          (* open the text file, assume the file  *)
                                (* exists and no error in the file I/O. *)

  WHILE( NOT(EOF(YFILE)) ) DO          (* process one sentence of the   *)
  BEGIN                                (* file in each pass of the loop *)

    STRG := YFILE^ ;                   (* get the sentence from the *)
```

```
                                        (* file window pointer       *)
    SPEAK_STRING(STRG, IDX, EOS) ;                  (* speak the sentence *)
    GET(YFILE)                          (* advance the file window pointer *)
  END ;
  CLOSE(YFILE)          (* close the text data file *)
END ;
```

```
(* ------------------------------------------------------------------ *)
(* This procedure stores the digitized voice data of one or more words *)
(* into a buffer. The voice of each full word is input separately. Each *)
(* full word may consist of several words, as long as the total unit of *)
(* sound is not greater than MAXUNIT (5). The capacity of this buffer  *)
(* VOICEBUFF is 20 (BUFFUNIT) units of sound.                          *)
(*                                                                     *)
(* The delay constant of digitizing (DTEMPO) can be chosen and input by *)
(* the user interactively. Therefore, this procedure is useful for the *)
(* user to find out the best value of delay constant in the digitizing *)
(* process of his/her own voice.                                       *)
(*                                                                     *)
(* Input :                                                             *)
(*   Voice of one or more words.                                       *)
(*                                                                     *)
(* Output :                                                            *)
(*   Digitized binary voice/sound data in the VOICE global buffer.     *)
(*                                                                     *)
(* The following procedure is called :                                 *)
(*   LISTEN - MODULE3 unit, digitize the input voice and place the digi_*)
(*            tized binary voice data into global buffer VOICE.        *)
(*                                                                     *)
(* The following global variables of the GLOBAL unit are used :        *)
(*   TOTWORDS - total number of words in the VOICEBUFF buffer.         *)
(*   DTEMPO - delay constant in digitizing process.                    *)
(*   MAXUNIT - maximum number of sound units of each word.             *)
(*   BUFFUNIT - maximum capacity of VOICEBUFF buffer.                  *)
(*   LENWORDS - contains the number of sound units of each word in the *)
(*              VOICEBUFF global buffer.                               *)
(*   VOICE - buffer of binary sound data of a word.                    *)
(*   VOICEBUFF - buffer of binary sound data of several words.         *)
(* ------------------------------------------------------------------ *)

PROCEDURE LISTEN_BUFFER ;

VAR I,PTR,VAL : INTEGER ;
    EOLB : BOOLEAN ;
    INPKEY : CHAR ;
    NUNIT : UNITRANGE ;

BEGIN

  PTR := 0 ;
  TOTWORDS := 0 ;
  EOLB := FALSE ;
  PAGE(OUTPUT) ;
  WRITELN(' STORE VOICE OF WORDS IN A BUFFER ') ;
  WRITELN('----------------------------------') ;

  WRITELN ;    (* get the delay constant of digitizing interactively *)
  WRITELN('> INPUT DELAY CONSTANT OF DIGITIZING : ') ;
  READLN(DTEMPO) ;

  REPEAT           (* process each full word separately *)
```

```
    WRITELN ;       (* get the total units of sound interactively *)
    WRITELN('> INPUT TOTAL SOUND UNITS OF WORD [1,..,5] ') ;
    WRITE('  INPUT 0 TO QUIT, UNIT : ') ;
    READLN(VAL) ;

    IF ( (VAL>0) AND (VAL<=MAXUNIT) AND     (* process additional word *)
         ((PTR+VAL)<=BUFFUNIT) ) THEN       (* only when the capacity  *)
    BEGIN                                   (* of the buffer is not    *)
                                            (* exceeded                *)
      TOTWORDS := TOTWORDS + 1 ;
      LENWORDS[TOTWORDS] := VAL ;
      NUNIT := VAL ;
      WRITELN ;
      WRITELN('> HIT RETURN KEY WHEN READY FOR SOUND INPUT') ;
                            (* digitizing the input sound,          *)
      LISTEN ( NUNIT ) ;    (* the digitized voice data is returned *)
                            (* in the global buffer VOICE           *)
      READLN(INPKEY) ;
      FOR I := 1 TO NUNIT DO                (* transfer data of each  *)
        VOICEBUFF[PTR+I] := VOICE[I] ;      (* word into the destina_ *)
      PTR := PTR + NUNIT                    (* tion buffer VOICEBUFF  *)

    END

    ELSE IF ( VAL = 0 )             (* normal ending of input sound *)
           THEN EOLB := TRUE

    ELSE IF ( (PTR+NUNIT) > BUFFUNIT )      (* exceed buffer capacity, *)
    THEN                                    (* end of input with a     *)
    BEGIN                                   (* message                 *)
      EOLB := TRUE ;
      WRITELN ;
      WRITELN('> BUFFER HAS < ',NUNIT:1,' UNITS VACANT')
    END

  UNTIL EOLB

END ;


(* ------------------------------------------------------------------ *)
(* This procedure speaks the digitized voice data of one or more words  *)
(* from buffer VOICEBUFF. The capacity of this buffer is 20 (BUFFUNIT)  *)
(* units of sound.                                                      *)
(*                                                                      *)
(* The delay constant of analoging (ATEMPO) can be chosen and input by  *)
(* the user interactively. Therefore, this procedure is useful for the  *)
(* user to find out the best value of delay constant in the playback    *)
(* process of his/her own digitized voice data.                         *)
(*                                                                      *)
(* This procedure is used after the preceding LISTEN_BUFFER procedure   *)
(* has been called. The combination of these two procedures gives the   *)
   user his/her own best & unique pair of digitizing and analoging      *)
(* delay constants. The reason for this unique pair of constants is     *)
(* that each person has distinct voice.                                 *)
```

```
(*                                                                    *)
(* Input :                                                            *)
(*   CONFIRM - parameter, wait for the user to press the RETURN key to *)
(*             indicate that he/she is ready to listen.               *)
(*   VOICEBUFF - buffer of binary sound data of several words.        *)
(*                                                                    *)
(* Output :                                                           *)
(*   Sound of the words in VOICEBUFF buffer which are output on the   *)
(*   speaker of Apple II or attached casette recorder depending on the *)
(*   value of global variable SPKER.                                  *)
(*                                                                    *)
(* The following procedure is called :                                *)
(*   SPEAK - MODULE3 unit, analog (playback) the digitized binary voice *)
(*           data which resides in the global buffer VOICE            *)
(*                                                                    *)
(* The following global variables of the GLOBAL unit are used :       *)
(*   TOTWORDS - total number of words in the VOICEBUFF buffer.        *)
(*   ATEMPO - delay constant in digitizing process.                   *)
(*   LENWORDS - contains the number of sound units of each word in the *)
(*              VOICEBUFF global buffer.                              *)
(*   VOICE - buffer of binary sound data of a word.                   *)
(*   VOICEBUFF - buffer of binary sound data of several words.        *)
(* ------------------------------------------------------------------ *)

PROCEDURE SPEAK_BUFFER ( CONFIRM : BOOLEAN ) ;

VAR I, J, PTR, STRTPTR, ENDPTR : INTEGER ;
    INPKEY : CHAR ;
    NUNIT : UNITRANGE ;

BEGIN

  WRITELN ;    (* get the delay constant of analoging interactively *)
  WRITELN('> INPUT DELAY FACTOR OF ANALOGING : ') ;
  READLN(ATEMPO) ;

  IF ( CONFIRM ) THEN       (* confirm that user is ready to listen *)
  BEGIN
    PAGE(OUTPUT) ;
    WRITELN('> HIT RETURN KEY WHEN READY') ;
    READLN(INPKEY)
  END ;

  STRTPTR := 1 ;                (* speak each full word separately *)
  FOR I := 1 TO TOTWORDS DO
  BEGIN

    NUNIT := LENWORDS[I] ;      (* total unit of sound of each full word *)
    ENDPTR := STRTPTR + NUNIT - 1 ;
    J := 0 ;

    FOR PTR := STRTPTR TO ENDPTR DO     (* transfer digitized data of *)
    BEGIN                               (* each word into the global  *)
      J := J + 1 ;                      (* buffer VOICE               *)
      VOICE[J] := VOICEBUFF[PTR]
```

```
    END ;

    SPEAK ( NUNIT ) ;          (* speak the full word *)

    STRTPTR := ENDPTR + 1

  END

END ;

(* The main program of USER_MANUAL_MODULE3 sample program starts here  *)

BEGIN

  DO6_RESET ;  (* open all index & binary data files of the dictionary *)

  SPKER := APPLE ;     (* utilize the Apple II speaker for voice output *)

  (* Speak the word 'MILLION' ten times, each time varies the value of *)
  (* the analog delay constant ATEMPO in the sequence of 1,..,10. Try  *)
  (* to decide which value of ATEMPO gives the best quality of play_   *)
  (* back voice.                                                       *)

  WRD := 'MILLION' ;
  FOR ATEMPO := 1 TO 10 DO
    WORD_SPEAK ( WRD ) ;

  (* Speak the following sentence in STRG variable ten times, each     *)
  (* time varies the value of the analog delay constant ATEMPO in the  *)
  (* sequence of 1,..,10. Try to decide the best value of ATEMPO.      *)
  (*                                                                   *)
  (* The combination result of this test and the preceding test gives  *)
  (* a good estimation of the most suitable value of ATEMPO, to be     *)
  (* used in speaking any word of the existing dictionary from the     *)
  (* Apple II speaker. The reason for these two tests is that diffe_   *)
  (* rent Apple II (especially clone) micro computers might not have   *)
  (* exactly similar speaker.                                          *)

  STRG := 'TUESDAY, THE FIFTH DAY OF MAY, YEAR OF ONE THOUSAND NINE
           HUNDRED AND EIGHTY SEVEN' ;
  FOR ATEMPO := 1 TO 10 DO
    SPEAK_STRING ( STRG ) ;

  (* Speak all the sentences of a text file. Use the speaker of a con_ *)
  (* nected casette recorder for voice output, employ a fixed value    *)
  (* for the analoging delay constant ATEMPO.                          *)

  SPKER := CASETTE ;           (* choose speaker of casette recorder *)
  ATEMPO := 4 ;                (* fixed value for the delay constant *)
  FN := '#5:TEST.TEXT' ;       (* text file from the #5: disk drive  *)

  SPEAK_FILE ( FN ) ;          (* speak it *)

  (* Digitize a set of several speaking words of an individual user    *)
```

```
  (* five times, similar or different set of words can be used in each *)
  (* time but different value of DTEMPO delay constant is chosen.      *)
  (*                                                                   *)
  (* After each digitizing process, the digitized voice data in the    *)
  (* VOICEBUFF buffer is spoken (played back) five times, using a      *)
  (* different value of ATEMPO delay constant in each speaking process.*)
  (*                                                                   *)
  (* In both processes, the choice of delay constants ATEMPO and       *)
  (* DTEMPO is done interactively.                                     *)

  CONFIRM := TRUE ;          (* need confirmation before speaking *)
  FOR I := 1 TO 5 DO
  BEGIN

    LISTEN_BUFFER ;          (* listen, get the digitized voice data *)

    FOR J := 1 TO 5 DO
      SPEAK_BUFFER ( CONFIRM ) ;     (* speak it, still using the   *)
                                     (* speaker of casette recorder *)
  END ;

  DO6_CLOSE          (* ending, close all data files of the dictionary *)

END.
```

```
  ------------------------------------------------------------------ *)
  This is the example of utilizing MODULE1 and MODULE2 units to       *)
(* create, update and delete the dictionary in the storage diskette.  *)
(*                                                                    *)
(* This example USES four units :                                     *)
(*   GLOBAL - must be used by all application host program.           *)
(*   MODULE1 - this is the example program of using it.               *)
(*   MODULE2 - this is the example program of using it.               *)
(*   UTILITY - several procedures of this unit are called.            *)
(*                                                                    *)
(* The compiler option $S++ is used in this example, it slows down the *)
(* speed of compilation but more memory space is available.           *)
(*                                                                    *)
(* The following global variables of the GLOBAL unit are used :       *)
(*   VOLUMENAME, DTEMPO, ATEMPO, SPKER,                               *)
(*   I1FNAME, I2FNAME, I3FNAME, B1FNAME, B2FNAME, B3FNAME,            *)
(*   IFILE1, BFILE1.                                                  *)
(* ------------------------------------------------------------------ *)

PROGRAM USER_MANUAL_MODULE1_MODULE2 ( INPUT, OUTPUT ) ;

(*$S++*)                (* compiler option *)

USES GLOBAL, UTILITY, MODULE1, MODULE2 ;          (* declare the units *)
                                                  (* used in here.     *)
  R CHOICE, CNT1, CNT2 : INTEGER ;

BEGIN

  (* ---------------------------------------------------------------- *)
  (* Build the dictionary in the default storage diskette STORE: .    *)
  (* ---------------------------------------------------------------- *)

  (* Part 1 - Build the index data files. Directory of all the words  *)
  (*          and indices. The creation and closing of data files are *)
  (*          done in the following called procedure.                 *)

  BLD_DIRECTORY ;

  (* Part 2 - Build the binary data files, voice input of all words.  *)
  (*          The creation and closing of data files are done in the  *)
  (*          following called procedure.                             *)
  (*          Use the default value 1 of the delay constant in the    *)
  (*          voice digitizing process.                               *)

  BLD_VOICE ;


  (* ---------------------------------------------------------------- *)
  (* Update on the dictionary. The default storage diskette STORE: has *)
  (* all the entries (index and binary data files).                   *)
  (* ---------------------------------------------------------------- *)

  (* Insertion of entries.                                            *)
```

```
ADD_XENTRIES ;                    (* insertion of index data records *)

ADD_BENTRIES ;                  (* insertion of binary data records,    *)
                                (* use default value 1 of delay constant*)
CMB_XENTRIES ;                  (* combine new & old index data records *)

CMB_BENTRIES ;                (* combine new & old binary data records *)

(* Deletion of entries.                                                 *)

DO_DELETE ;                        (* speedy deletion *)

DO_CLNUP ;                  (* clean up the deleted records physically *)
                            (* and permanently from the dictionary     *)

(* Improve the sound of entries.                                        *)

DTEMPO := 2 ;                     (* use new value of delay cocnstant *)
DO6_RESET ;                       (* open all data files *)

IMPROVE_SOUND;

DO6_CLOSE ;                       (* close all data files *)

(* Print the directory of the dictionary, content of all index data  *)
(* records from the theree index data files.                         *)

CHOICE := 4 ;
PRT_ENTRIES ( CHOICE ) ;


(* ------------------------------------------------------------------ *)
(* Deleting the whole dictionary in the default storage diskette      *)
(* STORE: .                                                           *)
(* ------------------------------------------------------------------ *)

CLR_DICTIONARY  ;


(* ------------------------------------------------------------------ *)
(* Modify the default values of some global variables.                *)
(*                                                                    *)
(* The change of values only applies to the current processing of     *)
(* all procedures in the project. After the Apple II UCSD Pascal      *)
(* system is rebooted, it is back to the default values.              *)
(*                                                                    *)
(* If change of default values must be used in every processing, it   *)
(* is a good practice to define a special procedure which USES GLOBAL*)
(* unit. All the change of default values is defined in this special  *)
(* procedure, the main program of application host program must call  *)
(* this special procedure in the beginning.                           *)
(* ------------------------------------------------------------------ *)

VOLUMENAME := '#5:' ;                (* treate any diskette in drive *)
                                     (* #5: as the storage diskette. *)
```

```
  ATEMPO := 5 ;              (* change the delay constants for    *)
  DTEMPO := 5 ;              (* analoging & digitizing processes. *)

  SPKER := CASETTE ;              (* use the speaker of attached casette *)
                                  (* recorder for voice output.          *)

  (* Change the filenames of all index and binary data files.           *)

  I1FNAME := CONCAT(VOLUMENAME,'IDX1.DATA') ;
  I2FNAME := CONCAT(VOLUMENAME,'IDX2.DATA') ;
  I3FNAME := CONCAT(VOLUMENAME,'IDX3.DATA') ;
  B1FNAME := CONCAT(VOLUMENAME,'BIN1.DATA') ;
  B2FNAME := CONCAT(VOLUMENAME,'BIN2.DATA') ;
  B3FNAME := CONCAT(VOLUMENAME,'BIN3.DATA') ;

  DO6_RESET ;                     (* open all data files in the new *)
                                  (* storage diskette in drive #5: .*)

  CNT1_ELEM ( IFILE1, CNT1 ) ;       (* count the total # of index data  *)
                                     (* records in the new first index   *)
                                     (* data file.                       *)

  CNT2_ELEM ( BFILE1, CNT2 ) ;       (* count the total # of binary data *)
                                     (* records in the new first binary  *)
                                     (* data file.                       *)

  CHOICE := 3 ;                      (* print the content of all index *)
  PRT_ENTRIES ( CHOICE ) ;           (* data records in the new third  *)
                                     (* index data file.               *)

  DO6_CLOSE                          (* close all data files *)

END.
```

3.4 SYSTEM CONFIGURATION

3.4.1 FLOPPY DISKETTES SUPPLIED

Three floppy diskettes are supplied in this project. The volumename of these three diskettes are AUDIO:, UNIT: and STORE:.

There are five files inside the AUDIO: diskette, they are :
(1) SYSTEM.APPLE, SYSTEM.PASCAL, SYSTEM.MISCINFO, SYSTEM.CHARSET - these four system files are needed to boot the Apple UCSD Pascal System.
(2) SYSTEM.LIBRARY - it is the system library of this project.

There are seven library units inside the SYSTEM.LIBRARY library :
(1) PASCALIO - Apple Pascal System Linked Intrinsic routines for file I/O.
(2) GLOBAL - It is the interface part of this project. The user must 'USES' it in the host program in order to call any Pascal language procedure of this project.
(3) DIGITAL - It contains the three Assembly language routines for the tasks of voice digitizing and analoging. The routine names are DIGITAL, ANALOG1 and ANALOG2.
(4) UTILITY - code file of UTILITY unit.
(5) MODULE1 - code file of MODULE1 unit.
(6) MODULE2 - code file of MODULE2 unit.
(7) MODULE3 - code file of MODULE3 unit.

Inside the UNIT: diskette, the code and text files of all five units can be found. There are twelve files :
(1) DA.TEXT and DA.CODE - they are the files of DIGITAL, ANALOG1 and ANALOG2 external procedures.
(2) GLOBAL.TEXT and GLOBAL.CODE - files of GLOBAL unit.
(3) UTILITY.TEXT and UTILITY.CODE - files of UTILITY unit.
(4) MODULE1.TEXT and MODULE1.CODE - files of MODULE1 unit.
(5) MODULE2.TEXT and MODULE2.CODE - files of MODULE2 unit.
(6) MODULE3.TEXT and MODULE3.CODE - files of MODULE3 unit.

The STORE: diskette has all the data files. There are three index data files, three binary data files and one character string data file. In other words, the STORE: diskette is the dictionary of this project. The filenames are :
(1) INDEX1.DATA - it is the index data file for those words with starting characters in the range of A,..,H.
(2) INDEX2.DATA - it is the index data file for those words with starting characters in the range of I,..,P.
(3) INDEX3.DATA - it is the index data file for those words with starting characters in the range of Q,..,Z.
(4) BINARY1.DATA - it contains the binary sound data of all the words in INDEX1.DATA file.
(5) BINARY2.DATA - it contains the binary sound data of all the words in INDEX2.DATA file.

(6) BINARY3.DATA - it contains the binary sound data of all the words in INDEX3.DATA file.
(7) STRING.DATA - this is a data file of character string record for demonstration purpose, each string (sentence) has one or more words. Some of the words in a string record might not exist in the dictionary.

The supplied diskettes AUDIO: and STORE: must be available at the same time. It is not advisable to do program development on the STORE: diskette, disk space is available in the AUDIO: diskette for programming purpose.

3.4.2 HARDWARE CONFIGURATION

This project is intended to run on the Apple II and Apple II Plus computers with a CRT, although the development was carried out on an Apple II Plus computer.

The following extra items are needed :
(1) Apple Pascal System
(2) Apple Language System with the required language card
(3) 48K or more bytes of installed RAM
(4) Two or more Apple Disk II disk drives

The previous 3.4.1 subsection states that the supplied diskettes AUDIO: and STORE: must be available at the same time, therefore the above requirement of two or more disk drives is justified.

A cassette recorder with RCA plugs connected to the Apple II computer's cassette input and output ports is optional. A connected cassette recorder gives the following advantages :
(1) Input (listen) - digitizing input sound. This is better than the direct microphone input. The volume and tone control of the cassette recorder can be used to input the correct amplitude and control the frequency range. In this way, the given voice can be reproduced with minimal noise. At the same time, the best of the recorded voices can be chosen and replayed repeatedly.
(2) Output (speak) - analoging binary sound data. Voice is recorded on the cassete tape or output in the public address (PA) mode. The cassette recorder tone control is used to filter out unwanted noise.

5. LISTING OF WORDS IN THE SAMPLE DICTIONARY

There are 47 words in the sample dictionary. For the groups of words which have the starting character in the sets of ['A',..,'H'], ['I',..,'P'] and ['Q',..,'Z'], respectively, the total number of words are 15, 13 and 19. The following table is a listing of all the words with the units of sound of each word.

WORD	UNITS OF SOUND	WORD	UNITS OF SOUND	WORD	UNITS OF SOUND
APRIL	2	JANUARY	3	SATURDAY	3
AUGUST	4	JULY	2	SEPTEMBER	3
DECEMBER	4	JUNE	2	SEVEN	2
EIGHT	2	MARCH	2	SEVENTEEN	3
EIGHTEEN	2	MAY	1	SEVENTY	3
EIGHTY	2	MILLION	3	SIX	2
ELEVEN	3	MONDAY	2	SIXTEEN	3
FEBRUARY	3	NINE	1	SIXTY	3
FIFTEEN	2	NINETEEN	2	SUNDAY	2
FIFTY	2	NINETY	2	TEN	1
FIVE	2	NOVEMBER	3	THIRTEEN	3
FORTY	2	OCTOBER	3	THIRTY	2
FOUR	1	ONE	1	THOUSAND	3
FOURTEEN	2			THREE	2
HUNDRED	2			THURSDAY	3
				TUESDAY	4
				TWELVE	2
				TWENTY	2
				TWO	1

3.6 DESCRIPTION AND LISTING OF PROCEDURES

```
(* ------------------------------------------------------------------ *)
(* This is the GLOBAL unit of Voice/Digital and Digital/Voice Conver_  *)
(* sion on a Microcomputer project.                                    *)
(*                                                                     *)
(* All the constants, variable types and global variables are declared *)
(* in this unit and will be utilized by all the procedures of this     *)
(* project. Any Pascal language host program which USES this unit can  *)
(* access and manipulate the content of this GLOBAL unit.              *)
(*                                                                     *)
(* In the IMPLEMENTATION section of this unit, some global variables   *)
(* are initialized with defaul values.                                 *)
(*                                                                     *)
(* The compiler option $S++ is invoked here, more memory space is      *)
(* available for the compiling process but the speed of compilation is *)
(* decreased.                                                          *)
(* ------------------------------------------------------------------ *)

(*$S++*)                              (* compiler option *)

UNIT GLOBAL ;


INTERFACE

  CONST

    MAXUNIT = 5 ;
    MAXFILE = 3 ;
    MAXCHAR = 20 ;
    MAXWORD = 25 ;
    UNITSIZE = 255 ;
    TOTALWORD = 100 ;
    BUFFUNIT = 20 ;
    FILENMLEN = 40 ;
    VOLNMLEN = 8 ;

  TYPE

    WORDRANGE = 1..MAXWORD ;
    TWORDRANGE = 0..TOTALWORD ;
    FILERANGE = 1..MAXFILE ;
    UNITRANGE = 1..MAXUNIT ;
    SPEAKEROF = (CASETTE,APPLE) ;
    WORD = STRING[MAXCHAR] ;
    FILENAME = STRING[FILENMLEN] ;
    VOLNAME = STRING[VOLNMLEN] ;
    IDXELEM = PACKED RECORD
                        STRG : WORD ;
                        UNITT : UNITRANGE ;
                        IDXX : TWORDRANGE ;
                        STATUS : BOOLEAN
                      END ;
    ELEM = PACKED RECORD
                     WUNIT : UNITRANGE ;
                     WIDX : TWORDRANGE ;
```

```
                    WSET : FILERANGE
                  END ;
    SOUND = ARRAY[UNITRANGE] OF STRING[UNITSIZE] ;
    SENTENCE = ARRAY[WORDRANGE] OF WORD ;
    ELEMARRAY = ARRAY[WORDRANGE] OF ELEM ;
    INDEXFILE = FILE OF IDXELEM ;
    BINARYFILE = FILE OF SOUND ;
    STRGFILE = FILE OF STRING ;

  VAR

    IFILE1, IFILE2, IFILE3 : INDEXFILE ;
    BFILE1, BFILE2, BFILE3 : BINARYFILE ;
    YFILE : STRGFILE ;
    VOICE : SOUND ;
    SPKER : SPEAKEROF ;
    DTEMPO, ATEMPO : INTEGER ;
    I1FNAME, I2FNAME, I3FNAME : FILENAME ;
    B1FNAME, B2FNAME, B3FNAME : FILENAME ;
    YFNAME : FILENAME ;
    SET1CHR, SET2CHR, SET3CHR, SET4CHR : SET OF CHAR ;
    SET1STCHR, SETCHRS : SET OF CHAR ;
    VOLUMENAME : VOLNAME ;
    CHRS1, CHRS2, CHRS3, CHRS4, CHRSALL : FILENAME ;
    VOICEBUFF : ARRAY[1..BUFFUNIT] OF STRING[UNITSIZE] ;
    LENWORDS : ARRAY[1..BUFFUNIT] OF UNITRANGE ;
    TOTWORDS : INTEGER ;

    PROCEDURE DO_NONE ;                    (* dummy procedure declaration *)


IMPLEMENTATION                          (* at least one procedure must reside *)
                                        (* in this section, therefore the     *)
  PROCEDURE DO_NONE ;                   (* dummy procedure is used here       *)

  BEGIN

  END ;


BEGIN

  VOLUMENAME := 'STORE:' ;                          (* storage diskette *)

  DTEMPO := 1 ;                                     (* delay constants *)
  ATEMPO := 4 ;

  SPKER := APPLE ;                                  (* use Apple II speaker *)

  SET1STCHR := ['A'..'Z'] ;                          (* define sets of      *)
   ET1CHR := ['A'..'H'] ;                            (* starting characters *)
   ET2CHR := ['I'..'P'] ;
  SET3CHR := ['Q'..'Z'] ;
  SET4CHR := ['1'..'5'] ;
```

```
(* define filenames of all index and binary data files.                *)

  I1FNAME := CONCAT(VOLUMENAME,'INDEX1.DATA') ;
  I2FNAME := CONCAT(VOLUMENAME,'INDEX2.DATA') ;
  I3FNAME := CONCAT(VOLUMENAME,'INDEX3.DATA') ;
  B1FNAME := CONCAT(VOLUMENAME,'BINARY1.DATA') ;
  B2FNAME := CONCAT(VOLUMENAME,'BINARY2.DATA') ;
  B3FNAME := CONCAT(VOLUMENAME,'BINARY3.DATA') ;

  YFNAME := CONCAT(VOLUMENAME,'DEMO.DATA') ;          (* demo data file*)
                                                      (* of senetences *)
  CHRS1 := 'A..H' ;
  CHRS2 := 'I..P' ;
  CHRS3 := 'Q..Z' ;
  CHRS4 := '1..5' ;
  CHRSALL := 'A..Z'

END .
```

```
(* ------------------------------------------------------------------ *)
(* This is the UTILITY unit of Voice/Digital and Digital/Voice Conver_ *)
(* sion on a Microcomputer project.                                    *)
(*                                                                     *)
(* This unit contains all the Pascal language utility procedures, each *)
(* utility procedure may be called by one or more procedures of        *)
(* MODULE1, MODULE2 and MODULE3 units.                                 *)
(*                                                                     *)
(* Only the GLOBAL unit is used here.                                  *)
(*                                                                     *)
(* The compiler option $S++ is invoked here, more memory space is      *)
(* available for the compiling process but the speed of compilation is *)
(* decreased.                                                          *)
(* ------------------------------------------------------------------ *)

(*$S++*)                                  (* compiler option *)

UNIT UTILITY ;


INTERFACE

  USES GLOBAL ;

   (* declare all the procedures of this module here *)

  PROCEDURE DO6_RESET ;
  PROCEDURE DO6_CLOSE ;
  PROCEDURE DISKETTE_ONLINE(IDX:INTEGER) ;
  PROCEDURE CNT1_ELEM (VAR XFILE:INDEXFILE; VAR COUNT:INTEGER ) ;
  PROCEDURE CNT2_ELEM (VAR XFILE:BINARYFILE; VAR COUNT:INTEGER ) ;

  PROCEDURE BLDIDX ( VAR XFILE:INDEXFILE; CHRS:FILENAME;
                     STRTIDX:INTEGER ) ;
  PROCEDURE GETVOICE ( VAR XFILE:BINARYFILE; VAR YFILE:INDEXFILE;
                       CHRS:FILENAME) ;

  PROCEDURE GET_WORDUNIT ( VAR STRG:WORD; VAR UVAL:INTEGER ) ;
  PROCEDURE WORD_VERIFY ( VAR WCHANGE, WFOUND:BOOLEAN; WRD:WORD;
                          VAR FNDREC:ELEM ) ;
  PROCEDURE FILE_SORT ( VAR X1FILE, X2FILE, YFILE:INDEXFILE ) ;


IMPLEMENTATION

  (* declare all the assembly language routines to be called here *)

  PROCEDURE DIGITAL(VAR BDATA:SOUND; BTEMPO, BIDX:INTEGER); EXTERNAL ;
```

```
(* ------------------------------------------------------------------ *)
(* This procedure opens all existing index data files and binary data  *)
(* files. For each data file, an error message is displayed when there *)
(* is error in the opening process.                                    *)
(*                                                                     *)
(* The process of opening a specified existing data file is accom_     *)
(* plished by calling the build in RESET procedure.                    *)
(*                                                                     *)
(* Input :                                                             *)
(*   All index data files and binary data files.                       *)
(*                                                                     *)
(* Output :                                                            *)
(*   Opened index data files and binary data files.                    *)
(*                                                                     *)
(* The following procedures are called :                               *)
(*   DISKETTE_ONLINE - UTILITY unit. It checks and requests that the   *)
(*     storage diskette STORE: be on line. At the same time, the first *)
(*     index data file is opened.                                      *)
(*   RESET - build in file I/O procedure. It opens the specified       *)
(*           existing file.                                            *)
(*                                                                     *)
(* The following global variables from the GLOBAL unit are used in     *)
(* this procedure :                                                    *)
(*   IORESULT - UCSD Pascal system variable. It is the error code of   *)
(*              the latest I/O operation, the value is zero for a      *)
(*              success completion I/O operation.                      *)
(*   IFILE2 - file window variable of the second index data file.      *)
(*   I2FNAME - filename of the second index data file.                 *)
(*   IFILE3 - file window variable of the third index data file.       *)
(*   I3FNAME - filename of the third index data file.                  *)
(*   BFILE1 - file window variable of the first binary data file.      *)
(*   B1FNAME - filename of the first binary data file.                 *)
(*   BFILE2 - file window variable of the second binary data file.     *)
(*   B2FNAME - filename of the second binary data file.                *)
(*   BFILE3 - file window variable of the third binary data file.      *)
(*   B3FNAME - filename of the third binary data file.                 *)
(* ------------------------------------------------------------------ *)

PROCEDURE DO6_RESET ;

VAR IDX : INTEGER ;
    FN : FILENAME ;

BEGIN

  IDX := 3 ;                          (* verify that the storage diskette *)
  DISKETTE_ONLINE( IDX ) ;            (* is on line. The first index data *)
                                      (* file is opened in the process.   *)
 (*$I-*)

  RESET(IFILE2, I2FNAME) ;                (* open second index data file *)
  IF ( IORESULT <> 0 ) THEN
    WRITELN('> ERROR IN OPENING INDEX DATA FILE ',I2FNAME) ;

  RESET(IFILE3, I3FNAME) ;                (* open third index data file *)
```

```
   IF ( IORESULT <> 0 ) THEN
     WRITELN('> ERROR IN OPENING INDEX DATA FILE ',I3FNAME) ;

   RESET(BFILE1, B1FNAME) ;                (* open first binary data file *)
   IF ( IORESULT <> 0 ) THEN
     WRITELN('> ERROR IN OPENING INDEX DATA FILE ',B1FNAME) ;

   RESET(BFILE2, B2FNAME) ;                (* open second binary data file *)
   IF ( IORESULT <> 0 ) THEN
     WRITELN('> ERROR IN OPENING INDEX DATA FILE ',B2FNAME) ;

   RESET(BFILE3, B3FNAME) ;                (* open third binary data file *)
   IF ( IORESULT <> 0 ) THEN
     WRITELN('> ERROR IN OPENING INDEX DATA FILE ',B3FNAME)

   (*$I+*)

END ;
```

```
(* ------------------------------------------------------------------ *)
(* This procedure closes all existing opened index data files and     *)
(* binary data files. The process of closing a specified data file is *)
(* accomplished by calling the build in CLOSE procedure.              *)
(*                                                                    *)
(* Input :                                                            *)
(*   All opened index data files and binary data files.               *)
(*                                                                    *)
(* Output :                                                           *)
(*   Closed index data files and binary data files.                   *)
(*                                                                    *)
(* The following procedure is called :                                *)
(*   CLOSE - build in file I/O procedure. It closes the specified     *)
(*           existing opened file.                                    *)
(*                                                                    *)
(* The following global variables from the GLOBAL unit are used in    *)
(* this procedure :                                                   *)
(*   IFILE1 - file window variable of the first index data file.      *)
(*   IFILE2 - file window variable of the second index data file.     *)
(*   IFILE3 - file window variable of the third index data file.      *)
(*   BFILE1 - file window variable of the first binary data file.     *)
(*   BFILE2 - file window variable of the second binary data file.    *)
(*   BFILE3 - file window variable of the third binary data file.     *)
(* ------------------------------------------------------------------ *)

PROCEDURE DO6_CLOSE ;

BEGIN

(*$I-*)

  CLOSE(IFILE1) ;                  (* index data files processing *)
  CLOSE(IFILE2) ;
  CLOSE(IFILE3) ;

  CLOSE(BFILE1) ;                  (* binary data files processing *)
  CLOSE(BFILE2) ;
  CLOSE(BFILE3)

(*$I+*)

END ;
```

```
(* ------------------------------------------------------------- *)
(* This procedure verifies that the storage diskette is on line. When  *)
(* the verification fails, the user is requested to put the diskette   *)
(* on line. The verification is accomplished by creating or opening a  *)
(* data file, either the first index or binary data file. The filename *)
(* is defined with the volumename, therefore, when failure occurs in   *)
(* the verification, it usually means that the diskette is not on line.*)
(*                                                                     *)
(* The processing of this procedure is only ended when the correct     *)
(* storage diskette is on line. When the error and request message is  *)
(* still displayed but the storage diskette has already been inserted  *)
(* in the disk drive, check the following possibilities :              *)
(* (1) The global variable VOLUMENAME has been altered and the related *)
(*     storage diskette is not on line.                                *)
(* (2) Diskette I/O errors                                             *)
(* (3) Disk drive I/O errors                                           *)
(*                                                                     *)
(* Input :                                                             *)
(*   IDX - parameter. It tells the procedure whether to use index or   *)
(*         binary data file in order to accomplish the objective. The  *)
(*         data file may be created or opened. The possible values of  *)
(*         this parameter are :                                        *)
(*     (1) Creates new first index data file                           *)
(*     (2) Creates new first binary data file                          *)
(*     (3) Open existing first index data file                         *)
(*     (4) Open existing first binary data file                        *)
(*                                                                     *)
(* Output :                                                            *)
(*   One of the following four possibilities :                         *)
(*   (1) A new and empty first index data file                         *)
(*   (2) A new and empty first binary data file                        *)
(*   (3) An opened existing first index data file                      *)
(*   (4) An opened existing first binary data file                     *)
(*                                                                     *)
(* The following procedures are called :                               *)
(*   REWRITE - build in file I/O procedure. It creates a new and empty *)
(*             data file.                                              *)
(*   RESET - build in file I/O procedure. It opens an existing data    *)
(*           file.                                                     *)
(*                                                                     *)
(* The following global variables of the GLOBAL unit are used :        *)
(*   IORESULT - UCSD Pascal system variable. It has the result code of *)
(*              the latest I/O operation, the value is zero for a      *)
(*              success completion I/O operation.                      *)
(*   IFILE1 - file window variable of the first index data file.       *)
(*   I1FNAME - filename of the first index data file.                  *)
(*   BFILE1 - file window variable of the first binary data file.      *)
(*   B1FNAME - filename of the first binary data file.                 *)
(*   VOLUMENAME - volume name of the storage diskette, the default     *)
(*                name is STORE:                                       *)
(* ------------------------------------------------------------- *)

PROCEDURE DISKETTE_ONLINE ;

VAR RTN : CHAR ;
```

```
    IOR : INTEGER ;

BEGIN

(*$I-*)                          (* compiler option, disable I/O checking *)
                                           (* initialize local copy *)
  IOR := 1 ;                               (* of IORESULT variable  *)

  WHILE ( IOR <> 0 ) DO          (* end of loop processing when the sto_ *)
  BEGIN                          (* rage diskette is on line.            *)

    CASE IDX OF

      1: REWRITE(IFILE1,I1FNAME) ;          (* create first index file *)

      2: REWRITE(BFILE1,B1FNAME) ;          (* create first binary file *)

      3: RESET(IFILE1,I1FNAME) ;            (* open existing index file *)

      4: RESET(BFILE1,B1FNAME)              (* open existing binary file *)

    END ;

    IOR := IORESULT ;

    IF ( IOR <> 0 ) THEN                 (* unsuccessful preceding create *)
    BEGIN                                (* file or open file operation   *)
      WRITELN ;  WRITELN ;
      WRITE('> PUT ',VOLUMENAME,' DISKETTE IN DRIVE#5 THEN RETURN KEY') ;
      READLN(RTN)
    END

  END

(*$I+*)                  (* compiler option, resumes I/O checking *)

END ;
```

```
  ------------------------------------------------------------------ *)
(* This procedure counts the total number of index data records in an  *)
(* index data file. The data file has been opened by the calling       *)
(* procedure.                                                          *)
(*                                                                     *)
(* Input :                                                             *)
(*   XFILE - parameter, it is the file window variable of the index    *)
(*           data file.                                                *)
(*                                                                     *)
(* Output :                                                            *)
(*   COUNT - parameter, total number of index data records.            *)
(*                                                                     *)
(* The following procedures are called :                               *)
(*   EOF - build in file I/O function. It indicates whether the end of *)
(*         the specified data file has been reached.                   *)
(*   GET - build in file I/O procedure. It advances the file window    *)
(*         variable to the next record and moves the content of this   *)
(*         record into the file buffer variable.                       *)
(*                                                                     *)
(* The following global variables of the GLOBAL unit are used :        *)
(*   None                                                              *)
(* ------------------------------------------------------------------ *)

PROCEDURE CNT1_ELEM ;

  ;IN

  COUNT := 0 ;                      (* initializes the counter parameter *)

  WHILE ( NOT(EOF(XFILE)) ) DO        (* loop, counts one record *)
  BEGIN                               (* in each pass            *)

    COUNT := COUNT + 1 ;              (* increments the counter by 1 *)

    GET(XFILE)                        (* moves to the next data record *)

  END

END ;
```

```
(* ------------------------------------------------------------------ *)
(* This procedure counts the total number of binary data records in a  *)
(* binary data file. The data file has been opened by the calling      *)
(* procedure.                                                          *)
(*                                                                     *)
(* Input :                                                             *)
(*   XFILE - parameter, it is the file window variable of the binary   *)
(*           data file.                                                *)
(*                                                                     *)
(* Output :                                                            *)
(*   COUNT - parameter, total number of binary data records.           *)
(*                                                                     *)
(* The following procedures are called :                               *)
(*   EOF - build in file I/O function. It indicates whether the end of *)
(*         the specified data file has been reached.                   *)
(*   GET - build in file I/O procedure. It advances the file window    *)
(*         variable to the next record and moves the content of this   *)
(*         record into the file buffer variable.                       *)
(*                                                                     *)
(* The following global variables of the GLOBAL unit are used :        *)
(*   None                                                              *)
(* ------------------------------------------------------------------ *)

PROCEDURE CNT2_ELEM ;

IN

  COUNT := 0 ;                              (* initializes the counter *)

  WHILE ( NOT(EOF(XFILE)) ) DO              (* loop, counts one    *)
  BEGIN                                     (* record in each pass *)

    COUNT := COUNT + 1 ;                    (* increments counter by 1 *)

    GET(XFILE)                             (* moves to next data record *)

  END

END ;
```

```
(* ------------------------------------------------------------------ *)
(* This procedure builds the content (all index data records) of an   *)
(* index data file. The data file has been created and opened by the  *)
(* calling procedure.                                                 *)
(*                                                                    *)
(* Each index data record consists of :                               *)
(*   word element - character string of a word                        *)
(*   total number of sound units of the word element                  *)
(*   index element - index number (record number) of a digitized bina_*)
(*                   ry data record in the associated binary data file.*)
(*   status element - indicates whether this is an existing data re_  *)
(*     cord or deleted data record, respectively, with boolean TRUE   *)
(*     or FALSE value.                                                *)
(*                                                                    *)
(* The word elements of all the index records in this data file must  *)
(* use the specified set of starting letters.                         *)
(*                                                                    *)
(* Input :                                                            *)
(*   XFILE - parameter, file window variable of the index data file to*)
(*           be built which was just opened by the calling procedure. *)
(*   CHRS - parameter, message string of the current set of starting  *)
(*          letters.                                                  *)
(*   STRTIDX - starting index number for the first index record.      *)
(*                                                                    *)
(* Output :                                                           *)
(*   The content (all index records) of an index data file.           *)
(*                                                                    *)
(* The following procedures are called :                              *)
(*   GET_WORDUNIT - UTILITY unit. It obtains a word and its total num_*)
(*     ber of sound unit from the user interactively.                 *)
(*   LENGTH - build in string function. It returns the length of a cha_*)
(*     racter string.                                                 *)
(*   PUT - build in file I/O procedure. It advances the file window va_*)
(*     riable to the next record and moves the content of the file    *)
(*     buffer variable into this record.                              *)
(*                                                                    *)
(* The following global variables of the GLOBAL unit are used :       *)
(*   None                                                             *)
(* ------------------------------------------------------------------ *)

PROCEDURE BLDIDX ;

VAR IDX, UVAL, L : INTEGER ;
    WRD : WORD ;

BEGIN

  IDX := STRTIDX ;
  WRITELN('SECTION OF WORDS STARTING WITH ',CHRS:6) ;
  WRITELN('--------------------------------------') ;
   RITELN ;

  GET_WORDUNIT(WRD,UVAL) ;                  (* get a word & its number *)
                                            (* of sound units          *)
  L := LENGTH(WRD) ;              (* get the length of input word string *)
```

```
  WHILE ( L > 0 ) DO              (* loop, process one word in each pass, *)
  BEGIN                           (* end of loop when null word was input,*)
                                  (* null word has zero length.           *)
    XFILE^.STRG := WRD ;                (* assign word to file buffer var *)
    XFILE^.UNITT := UVAL ;              (* assign number of unit sound *)
    XFILE^.IDXX := IDX ;                (* assign record number *)
    XFILE^.STATUS := TRUE ;             (* assign existing record status *)

    PUT(XFILE) ;                        (* put all data into data record *)

    IDX := IDX + 1 ;                          (* next record/index number *)

    GET_WORDUNIT(WRD,UVAL) ;                  (* get next word etc *)
    L := LENGTH(WRD)

  END

END ;
```

```
(* ------------------------------------------------------------------ *)
(* This procedure builds the content (all binary data records) of an  *)
(* binary data file. The data file has been created and opened by the  *)
(* calling procedure.                                                 *)
(*                                                                    *)
(* The associated index data file of the current binary data file is  *)
(* used because the sequence of binary records input must be similar  *)
(* with the sequence of related index records input. Before each voice *)
(* input, get the current index data record, displaying the word and  *)
(* its number of sound units. In this way, the user is reminded to    *)
(* input the correct voice data for the current word.                 *)
(*                                                                    *)
(* Input :                                                            *)
(*   YFILE - parameter, file window variable of the binary data file  *)
(*           to be built. It was just opened by the calling procedure.*)
(*   XFILE - parameter, file window variable of the index data file   *)
(*         which is associated with the current binary data file to be*)
(*         built. It was opened by the calling procedure.             *)
(*   CHRS - parameter, message string of the current set of starting  *)
(*          letters.                                                  *)
(*                                                                    *)
(* Output :                                                           *)
(*   The content (all binary data records) of a binary data file.     *)
(*                                                                    *)
(* The following procedures are called :                              *)
(*   PUT - build in file I/O procedure. It advances the file window va_*)
(*     riable to the next record and moves the content of the file    *)
(*     buffer variable into this record.                              *)
(*   GET - build in file I/O procedure. It advances the file window va_*)
(*     riable to the next record and moves the content of this record *)
(*     into the file buffer variable.                                 *)
(*   EOF - build in file I/O procedure. It indicates whether the end  *)
(*     of a specified file has been reached.                          *)
(*   DIGITAL - assembly language routine. It is the digitizing process*)
(*             of the input voice.                                    *)
(*                                                                    *)
(* The following global variables of the GLOBAL unit are used :       *)
(*   VOICE - It is a buffer which contains the digitized binary data  *)
(*           of the input voice.                                      *)
(*   DTEMPO - delay constant of the digitizing process of the input   *)
(*            voice.                                                  *)
(* ------------------------------------------------------------------ *)

PROCEDURE GETVOICE ;

VAR KEYIN : CHAR ;
    UVAL : INTEGER ;

BEGIN

  WRITELN('SECTION OF WORDS STARTING WITH ',CHRS:6) ;
  WRITELN('-------------------------------------') ;
  WRITELN ;
                                        (* loop, processes one word in *)
  WHILE ( NOT(EOF(YFILE)) ) DO          (* each pass when index data   *)
```

```
                                        (* still has data records.      *)
  BEGIN

    (* Get current index data record, display the word and its number  *)
    (* of sound unit. In this way, the user is reminded to input the   *)
    (* proper voice data for the current word.                         *)

    UVAL := YFILE^.UNITT ;
    WRITELN('WORD: ',YFILE^.STRG,' UNIT: ',UVAL) ;
    WRITELN('> PRESS A KEY WHEN VOICE INPUT IS READY') ;
                                                  (* digitizing input *)
    DIGITAL(VOICE, DTEMPO, UVAL) ;                (* voice data       *)

    XFILE^ := VOICE ;          (* assign result of DIGITAL process  *)
    PUT(XFILE) ;               (* to the current binary data record *)

    GET(YFILE) ;               (* get next index record *)

    READ(KEYIN) ;              (* for the key pressed in DIGITAL process *)
    WRITELN

  END ;

  WRITELN ;
  WRITELN('> END OF SECTION ',CHRS:6) ;
  WRITELN

END ;
```

```
   ------------------------------------------------------------------ *)
(* This procedure prompts user to input a word and its total number of *)
(* sound units interactively. The starting letter of the input word    *)
(* must be a member of the specified set of letters.                   *)
(*                                                                     *)
(* When no more word input is required, a null word is input by just   *)
(* pressing the RETURN key. In this case, the returned number of sound *)
(* units parameter is assigned with zero value.                        *)
(*                                                                     *)
(* Input :                                                             *)
(*   From the user interactively a word and its number of sound units. *)
(*   The starting letter of the input word must be in the specified    *)
(*   set of letters.                                                   *)
(*                                                                     *)
(* Output :                                                            *)
(*   STRG - parameter, it contains the word.                           *)
(*   UVAL - parameter, the total number of sound units of the word.    *)
(*                                                                     *)
(* The following procedures are called :                               *)
(*   LENGTH - build in string function. It returns the lengthh of a    *)
(*            character string.                                        *)
(*   ORD - build in string function. It returns the ASCII value of a   *)
(*         character.                                                  *)
(*                                                                     *)
(* The following global variables of the GLOBAL unit are used :        *)
     SETCHRS - current set of starting letters for all the word ele_   *)
               ments of current index data file.                       *)
(*   SET4CHR - set of digital characters for all the possible values   *)
(*             of the number of sound unit.                            *)
(* ------------------------------------------------------------------ *)

PROCEDURE GET_WORDUNIT ;

VAR CHR:CHAR ;
    ENDINPUT:BOOLEAN ;

BEGIN

  ENDINPUT := FALSE ;
  WRITELN ;
  WRITELN('> PRESS RETURN KEY ONLY IF NO MORE WORD INPUT !') ;
  WRITELN('  ----------------------------------------------') ;

  (* This is a loop to get a word from the user interactively. The     *)
  (* loop is only ended when a valid word or null word has been input. *)

  REPEAT

    WRITELN ;
    WRITE('WORD: ') ;
    READLN(STRG) ;                    (* get the input word *)

    IF ( LENGTH(STRG)=0 )                      (* end of loop when a null *)
      THEN ENDINPUT := TRUE                    (* word has been input     *)
```

```
    ELSE
      BEGIN
        IF ( STRG[1] IN SETCHRS )          (* end of loop when the word *)
          THEN ENDINPUT := TRUE            (* has valid starting letter *)
          ELSE
            BEGIN                                  (* invalid word input *)
              WRITELN('> WORD MUST START WITH CHARACTER ',
                      'IN DEFINED CHARACTER SET') ;
              WRITELN('  PRESS RETURN KEY ONLY IF NO MORE ',
                      'WORD INPUT')
            END
      END

  UNTIL ENDINPUT ;

  IF ( LENGTH(STRG) > 0 ) THEN               (* no need to get the number *)
  BEGIN                                      (* of sound units when a null*)
                                             (* word has been input       *)
    WRITELN ;
    WRITELN('UNITS OF SOUND IS ',CHRS4,' ONLY !') ;
    WRITELN('-------------------------------------') ;
    WRITELN ;

    REPEAT                          (* loop for number of sound units input*)
                                    (* until valid value has been input    *)
      WRITE('UNIT OF SOUND: ') ;
      READLN(CHR) ;

      IF ( NOT(CHR IN SET4CHR) )          (* input is not a valid digit *)
        THEN WRITELN('> UNIT OF SOUND MUST BE IN DEFINED RANGE !')

    UNTIL CHR IN SET4CHR ;

    UVAL := ORD(CHR) - ORD('0')           (* get value from digital char *)

  END

  ELSE UVAL := 0                        (* null word has zero sound unit *)

END ;
```

```
   ------------------------------------------------------------------ *)
(* This procedure verifies that a given word exists in the dictionary, *)
(* in other words, exists in any one of the three index data files of  *)
(* the storage diskette. The existence of this word implies that its   *)
(* binary data record also exists in the corresponding binary data.    *)
(* When a word entry had been deleted but the dictionary clean up pro_ *)
(* cedure has not been called to do periodical maintainance on all the *)
(* data files, the index data record of this word still exists in the  *)
(* index data file but the STATUS element has FALSE value.             *)
(*                                                                     *)
(* After the existence of the input word has been confirmed, there are *)
(* two possible continuing processings :                               *)
(* (1) Get all the required information in order to access the related *)
(*     binary data record. This is needed in 'speaking a word'.        *)
(* (2) Update the number of sound units of the input word. This is     *)
(*     needed when 'improve sound of the input word' is being processed*)
(*     and the improve process changes the number of sound units.      *)
(* The index/record number of the binary data record in the associated *)
(* binary data file is needed in both cases above.                     *)
(*                                                                     *)
(* The three index data file have been opened by the calling procedure.*)
(*                                                                     *)
(* Input :                                                             *)
(*   WCHANGE - parameter, it indicates whether the first or second     *)
(*             purpose of this procedure must be processed.            *)
     WRD - parameter, it has the character string of the input word.   *)
     All index data files.                                           *)
(*   FNDREC - parameter, a structured variable where the WUNIT element *)
(*            is used to pass the new value of number of sound units.  *)
(*            (for the second case only)                               *)
(*                                                                     *)
(* Output :                                                            *)
(*   WFOUND - parameter, it indicates whether the input word exists in *)
(*            the specified index data file. The value is TRUE when    *)
(*            the word exists.                                         *)
(*   FNDREC - parameter, a structured variable containing the informa_ *)
(*            tion which is required to access the associated binary   *)
(*            data record of the input word. (for first case only)     *)
(*                                                                     *)
(* The following procedures are called :                               *)
(*   DOVERIFY - local procedure which implements the main processing   *)
(*              of the calling procedure.                              *)
(*   EOF - build in file I/O function, it indicates whether the end of *)
(*         a specified file has been reached.                          *)
(*   SEEK - build in file I/O, it allows randow access to records. The *)
(*     file window variable is moved to a specified record in a file   *)
(*     with the record number provided.                                *)
(*   GET - build in file I/O, it advances the file window variable to  *)
(*     the next record and moves the content of this record to the     *)
(*     buffer variable.                                                *)
(*   PUT - build in file I/O, it advances the file window variable to  *)
     the next record and moves the content of the file buffer varia_ *)
(*     ble to this record.                                             *)
(*                                                                     *)
(* The following global variables of the GLOBAL unit are used :        *)
```

```
(*   SET1CHR - set of starting letters for the words elements of all   *)
(*     index data records of the first index data file.                 *)
(*   SET2CHR - set of starting letters for the words elements of all   *)
(*     index data records of the second index data file.                *)
(*   SET3CHR - set of starting letters for the words elements of all   *)
(*     index data records of the third index data file.                 *)
(*   IFILE1 -  file window variable of the first index data file.       *)
(*   IFILE2 -  file window variable of the second index data file.      *)
(*   IFILE3 -  file window variable of the third index data file.       *)
(* ------------------------------------------------------------------- *)

PROCEDURE WORD_VERIFY ;

VAR CHRSET, PSET, PIDX, PUNIT : INTEGER ;
    FIRSTCHR : CHAR ;
    SWRD : WORD ;
    CHANGE, FOUND, NOTIN : BOOLEAN ;


  (* ------------------------------------------------------------------- *)
  (* For a specified and opened index data file, this local procedure  *)
  (* searches through the data file to find out the index record which *)
  (* has matched word element with the input parameter WSTRG. All the  *)
  (* information required by the WORD_VERIFY procedure are obtained    *)
  (* and returned through PSET, PUNIT and PIDX variables.              *)
  (*                                                                   *)
  (* Input :                                                           *)
  (*   WSTRG - parameter, a word which is used in the seaching of the  *)
  (*           specified index file.                                   *)
  (*   XFILE - parameter, file window variable of the first, second or *)
  (*           third index data file. The searching of preceding para_ *)
  (*           meter (a word) is done in the related index data file.  *)
  (*                                                                   *)
  (* Output :                                                          *)
  (*   PSET - variable, the possible values are 1, 2 or 3. It indicates*)
  (*          whether the first, second or third binary data file has  *)
  (*          the associated binary data record of the input word.     *)
  (*   PUNIT - variable, it has the total number of sound units of the *)
  (*           input word. It is also the size of the associated binary*)
  (*           data record.                                            *)
  (*   PIDX - variable, it contains the index number (record number)   *)
  (*          of the associated binary data record in the related      *)
  (*          binary data file.                                        *)
  (*                                                                   *)
  (* The following procedures are called :                             *)
  (*   EOF, SEEK, PUT, GET                                             *)
  (*                                                                   *)
  (* The following global variables of the GLOBAL unit are used :      *)
  (*   None                                                            *)
  (* ------------------------------------------------------------------- *)

  PROCEDURE DOVERIFY ( WSTRG:WORD; VAR XFILE:INDEXFILE ) ;

  VAR RECIDX : INTEGER ;
```

```
EGIN

  RECIDX := 0 ;          (* index number to access the index data file,*)
                         (* to be used by the random access SEEK proc  *)

  WHILE( (NOT(NOTIN)) AND (NOT(EOF(XFILE))) AND (NOT(FOUND)) ) DO
  BEGIN

    (* Found because matched word element and the word (index record)*)
    (* exists in the index data file.                                 *)

    IF ( (WSTRG=XFILE^.STRG) AND (XFILE^.STATUS) ) THEN
    BEGIN

      FOUND := TRUE ;
      PSET := CHRSET ;                  (* Value of CHRSET is from *)
                                        (* the calling procedure   *)
      IF (CHANGE) THEN                  (* Second purpose *)
      BEGIN                             (* Assign new number of sound *)
        XFILE^.UNITT := PUNIT ;         (* units into the index record*)
        SEEK(XFILE, RECIDX) ;           (* These 2 procedures make   *)
        PUT(XFILE)                      (* the modification permanent*)
      END                               (* in the index data file    *)
      ELSE PUNIT := XFILE^.UNITT ;           (* main purpose, get the *)
                                             (* number of sound units *)
      PIDX := XFILE^.IDXX               (* get index/record number *)

    END

    ELSE IF ( WSTRG < XFILE^.STRG )
      THEN NOTIN := TRUE ;

    GET(XFILE) ;                        (* get next index data record *)
    RECIDX := RECIDX + 1                (* for random access of   *)
                                        (* next index data record *)
  END

END ;


BEGIN

  CHANGE := WCHANGE ;                   (* local copy *)
  SWRD := WRD ;                         (* local copy *)

  FOUND := FALSE ;
  NOTIN := FALSE ;

  IF ( CHANGE ) THEN                      (* second case, get the new *)
    PUNIT := FNDREC.WUNIT ;               (* number of sound units    *)

  .* from the first letter of the word, determine which one of the     *)
  (* three index data files that the word belongs to.                 *)

  FIRSTCHR := WRD[1] ;
```

```
F ( FIRSTCHR IN SET1CHR )                (* 1st set of starting letters *)
  THEN CHRSET := 1
  ELSE IF ( FIRSTCHR IN SET2CHR )        (* 2nd set of starting letters *)
    THEN CHRSET := 2
    ELSE IF ( FIRSTCHR IN SET3CHR )      (* 3rd set of starting letters *)
      THEN CHRSET := 3
      ELSE CHRSET := 4 ;                 (* others *)

CASE CHRSET OF

  1: BEGIN                                (* belong to first index file *)

       DOVERIFY(SWRD, IFILE1) ;                  (* do the verification *)

       SEEK(IFILE1, 0) ;                  (* move the file window varia *)
       GET(IFILE1)                        (* back to the beginning of   *)
                                          (* the index data file, for   *)
     END ;                                (* processing of next word    *)

  2: BEGIN                                (* second index data file *)

       DOVERIFY(SWRD, IFILE2) ;           (* verification *)

       SEEK(IFILE2, 0) ;                  (* move back to the beginning *)
       GET(IFILE2)                        (* of the index data file     *)

     END ;

  3: BEGIN                                (* third index data file *)

       DOVERIFY(SWRD, IFILE3) ;           (* verification *)

       SEEK(IFILE3, 0) ;                    (* move to the beginning of *)
       GET(IFILE3)                          (* the index file           *)

     END ;

  4: BEGIN          (* input word does not have valid starting letter *)

       WRITE('> 1ST CHARACTER OF WORD IS NOT IN THE DEFINED') ;
       WRITELN(' SET OF CHARACTERS !') ;

     END

END ;

WFOUND := FOUND ;                        (* Assignning information to *)
FNDREC.WIDX := PIDX ;                    (* the output parameters.    *)
FNDREC.WSET := PSET                      (* The number of sound units *)
IF ( NOT CHANGE ) THEN                   (* is only needed when pro_  *)
  FNDREC.WUNIT := PUNIT ;                (* cessing the first case.   *)

D ;
```

```
   ------------------------------------------------------------------ *)
   This procedure does merge sort on two index data files, the result  *)
(* is placed in a combined index data file. All three preceding data   *)
(* files have been opened by the calling procedure. The key of the     *)
(* merge sort process is the word elements of the index data records,  *)
(* they must be in alphabetical ascending order.                       *)
(*                                                                     *)
(* Input :                                                             *)
(*   X1FILE - parameter, file window variable of an index data file to *)
(*            be merge sorted, first entry in the parameter list.      *)
(*   X2FILE - parameter, file window variable of an index data file to *)
(*            be merge sorted, second entry in the parameter list.     *)
(*   YFILE - parameter, file window variable of the combined index     *)
(*           data file.                                                *)
(*                                                                     *)
(* Output :                                                            *)
(*   content of the resulting combined index data file                 *)
(*                                                                     *)
(* The following procedures are called :                               *)
(*   ASSIGN_ELEM - local procedure, it assigns the content of a source *)
(*     index data record to a destination index data record.           *)
(*   EOF - build in file I/O function, it indicates whether the end of *)
(*         a specified file has been reached.                          *)
(*                                                                     *)
(* The following global variables of the GLOBAL unit are used :        *)
(*   None                                                              *)
   ------------------------------------------------------------------ *)

PROCEDURE FILE_SORT ;

VAR WRD1, WRD2 : WORD ;


  (* ---------------------------------------------------------------- *)
  (* This local procedure copies current data record of the source in_ *)
  (* dex data file to the current data record of the destination index *)
  (* data file. Both data files have been opened by the calling proce_ *)
  (* dures, the file window variables point to the respective current  *)
  (* data records.                                                     *)
  (*                                                                   *)
  (* Input :                                                           *)
  (*   SRCE - parameter, file window variable of either one of the two *)
  (*          index data files to be merge sorted. It points to the    *)
  (*          index data record which is going to be copied to an index*)
  (*          record of the combined index file.                       *)
  (*   DSTN - parameter, file window variable of the combined index    *)
  (*          data file.                                               *)
  (*                                                                   *)
  (* Output :                                                          *)
  (*   an index data record of the combined index data file            *)
  (*                                                                   *)
   * The following procedures are called :                            *)
  (*   GET - build in file I/O, it advances the file window variable to*)
  (*     the next record and moves the content of this record to the   *)
  (*     buffer variable.                                              *)
```

```
(*   PUT - build in file I/O, it advances the file window variable to*)
(*     the next record and moves the content of the file buffer va_  *)
(*     riable to this record.                                        *)
(*                                                                   *)
(* The following global variables of the GLOBAL unit are used :      *)
(*   None                                                            *)
(* ----------------------------------------------------------------- *)

PROCEDURE ASSIGN_ELEM( VAR SRCE, DSTN : INDEXFILE ) ;

BEGIN

  DSTN^.STRG := SRCE^.STRG ;          (* assign various data elements *)
  DSTN^.UNITT := SRCE^.UNITT ;        (* of an index data record      *)
  DSTN^.IDXX := SRCE^.IDXX ;
  DSTN^.STATUS := SRCE^.STATUS ;

  PUT(DSTN) ;                      (* move to next data record of the *)
                                   (* destination index data file.    *)
  GET(SRCE)                        (* same for the source index file. *)

END ;


BEGIN

(* Loop, each pass processes two data records, one record from each  *)
(* index data file but only one data record will be chosen to build  *)
(* the content of the combined index data file. This combined file   *)
(* requires that the word elements of all its index data records in  *)
(* alphabetical ascending order. Therefore, the data record with the *)
(* smaller word element is chosen.                                   *)
(*                                                                   *)
(* End of loop processing occurs when anyone of the two index data   *)
(* files has no more data records.                                   *)

WHILE ( (NOT(EOF(X1FILE))) AND (NOT(EOF(X2FILE))) ) DO
BEGIN

  WRD1 := X1FILE^.STRG ;                (* get the word elements of   *)
  WRD2 := X2FILE^.STRG ;                (* both current index records.*)
                                            (* choose the data record *)
  IF ( WRD1 <= WRD2 )                       (* with smaller word elem *)
    THEN ASSIGN_ELEM(X1FILE, YFILE)         (* then assign it to the  *)
    ELSE ASSIGN_ELEM(X2FILE, YFILE)         (* combined index file.   *)

END ;

(* When one of the two index data files has no more data records,    *)
(* assign the remaining index data records of the other data file to *)
(* the combined index data file.                                     *)
                                   (* first index file of the parameter *)
IF ( EOF(X1FILE) )                 (* list has no more data records     *)
THEN
  WHILE ( NOT(EOF(X2FILE)) ) DO          (* assign remaining of the *)
```

```
      ASSIGN_ELEM(X2FILE, YFILE)              (* second index data file. *)

  ELSE
    WHILE ( NOT(EOF(X1FILE)) ) DO             (* assign remaining of the *)
      ASSIGN_ELEM(X1FILE, YFILE)              (* first index data file   *)

END ;


BEGIN


END.
```

```
(* ------------------------------------------------------------------ *)
   This is the MODULE1 unit of Voice/Digital and Digital/Voice Conver_ *)
(* sion on a Microcomputer project.                                    *)
(*                                                                     *)
(* The Pascal language procedures here create, print and purge the     *)
(* dictionary.                                                         *)
(*                                                                     *)
(* The GLOBAL and UTILITY units are used here.                         *)
(*                                                                     *)
(* The compiler option $S++ is invoked here, more memory space is      *)
(* available for the compiling process but the speed is decreased.     *)
(* ------------------------------------------------------------------ *)

(*$S++*)                          (* compiler option *)

UNIT MODULE1 ;

INTERFACE

  USES GLOBAL, UTILITY ;                  (* declare the units used here *)

  (* declare all the procedures of this module here *)

  PROCEDURE BLD_DIRECTORY ;
  PROCEDURE PRT_ENTRIES ( CHOICE : INTEGER ) ;
  PROCEDURE CLR_DICTIONARY ;
  PROCEDURE BLD_VOICE ;


IMPLEMENTATION
```

```
    ------------------------------------------------------------------ *)
    This procedure builds the directory of all index elements of the    *)
(* dictionary, in other words, the three index data files of the whole *)
(* dictionary in the STORE: storage diskette are created.              *)
(*                                                                     *)
(* The actual process of creating an index data file is accomplished   *)
(* by calling the BLDIDX procedure of UTILITY unit, providing the      *)
(* index data file has been opened. The first letters of all the       *)
(* word elements in an index data file belong to same set of alphabet  *)
(* letters, each one of the three index data files is associated with  *)
(* a distinct set of alphabet letters.                                 *)
(*                                                                     *)
(* Input :                                                             *)
(*   None                                                              *)
(*                                                                     *)
(* Output :                                                            *)
(*   Three index data files of the dictionary                          *)
(*                                                                     *)
(* The following procedures are called :                               *)
(*   DISKETTE_ONLINE - UTILITY unit, checks whether the STORE: dis_    *)
(*     kette is on line. Creates new and empty first index data file   *)
(*     when diskette on line status has been confirmed.                *)
(*   BLDIDX - UTILITY unit, builds the content of a given index data   *)
(*     file where the first letters of all the word elements belong    *)
(       to a specific set of alphabet letters.                        *)
     CLOSE - build in file I/O procedure, closes a given file and      *)
       makes it permanent.                                            *)
(*   REWRITE - build in file I/O procedure, creates a new and empty    *)
(*     index data file with the given filename.                        *)
(*                                                                     *)
(* The following global variables of the GLOBAL unit are used :        *)
(*   SET1CHR - set of the first letters of all the word elements of    *)
(*     the first index data file.                                      *)
(*   IFILE1 - file window variable for the first index data file.      *)
(*   SETCHRS - contains the current set of first letter.               *)
(*   CHRS1 - display message for the SET1CHR set of letters.           *)
(*   I2FNAME - filename of the second index data file.                 *)
(*   SET2CHR - set of the first letters of all the word elements of    *)
(*     the second index data file.                                     *)
(*   IFILE2 - file window variable for the second index data file.     *)
(*   CHRS2 - display message for the SET2CHR set of letters.           *)
(*   I3FNAME - filename of the third index data file.                  *)
(*   SET3CHR - set of the first letters of all the word elements of    *)
(*     the third index data file.                                      *)
(*   IFILE3 - file window variable for the third index data file.      *)
(*   CHRS3 - display message for the SET3CHR set of letters.           *)
(* ------------------------------------------------------------------ *)

PROCEDURE BLD_DIRECTORY;

    CHECKNUM, STRTIDX : INTEGER ;

BEGIN

  PAGE(OUTPUT) ;
```

```
  RITELN('-------------------------------------------------------') ;
  WRITELN(' BUILD THE DIRECTORY OF INDEX RECORDS OF THE DICTIONARY ') ;
  WRITELN('-------------------------------------------------------') ;
  WRITELN ;

  (* First index data file with the words elements start with letters  *)
  (* in the range of A,..,H. New index file, therefore the index       *)
  (* number starts from zero.                                          *)

  CHECKNUM := 1 ;                (* new first index data file indicator *)
  DISKETTE_ONLINE(CHECKNUM) ;       (* create new and empty file *)
  SETCHRS := SET1CHR ;              (* first set of starting letters *)
  STRTIDX := 0 ;                    (* first index number *)
  BLDIDX(IFILE1, CHRS1, STRTIDX) ;     (* create content of this file *)
  CLOSE(IFILE1, LOCK) ;                (* close this index data file *)

  (* Second index data file with the words elements start with letters *)
  (* in the range of I,..,P. New index file, therefore the index       *)
  (* number starts from zero.                                          *)
                                    (* create new and empty file *)
  REWRITE(IFILE2, I2FNAME) ;        (* with given filename       *)
  SETCHRS := SET2CHR ;
  STRTIDX := 0 ;
  BLDIDX(IFILE2, CHRS2, STRTIDX) ;
  CLOSE(IFILE2, LOCK) ;

  (* Third index data file with the words elements start with letters  *)
  (* in the range of Q,..,Z. New index file, therefore the index       *)
  (* number starts from zero.                                          *)

  REWRITE(IFILE3, I3FNAME) ;
  SETCHRS := SET3CHR ;
  STRTIDX := 0 ;
  BLDIDX(IFILE3, CHRS3, STRTIDX) ;
  CLOSE(IFILE3, LOCK)

END ;
```

```
(* ------------------------------------------------------------------ *)
(* This is a local procedure.                                         *)
(* It displays the content of an index data file in a table form      *)
(* after the related index data file has been opened and the file     *)
(* window variable is provided.                                       *)
(*                                                                    *)
(* Input :                                                            *)
(*   XFILE - parameter, file window variable of an index data file    *)
(*     which has been opened by the calling procedure.                *)
(*                                                                    *)
(* Output :                                                           *)
(*   Content of index data files on the screen                        *)
(*                                                                    *)
(* The following procedure is called :                                *)
(*   GET - build in file I/O procedure, advances the file window va_  *)
(*     riable by one record, in other words, get next index record.   *)
(* ------------------------------------------------------------------ *)

PROCEDURE DO_PRINT( VAR XFILE:INDEXFILE ) ;

BEGIN

  WHILE( NOT(EOF(XFILE)) ) DO       (* process one index record *)
  BEGIN                             (* in each pass of the loop *)

    WRITE(XFILE^.STRG:20) ;         (* display the word *)
    WRITE(XFILE^.UNITT:3) ;         (* the total unit of sound *)
    WRITE(XFILE^.IDXX:6) ;          (* the index number *)

    IF ( XFILE^.STATUS ) THEN       (* the existence status *)
      WRITELN('T':6)
    ELSE
      WRITELN('F':6) ;

    GET(XFILE)                      (* get next index record *)

  END

END ;


(* ------------------------------------------------------------------ *)
(* This procedure displays the content of index data files in a table *)
(* form. The actual process of displaying an index data file is       *)
(* carried out by calling the local procedure DO_PRINT by providing   *)
(* the file window variable, after the related index data file has been*)
(* opened in this procedure.                                          *)
(*                                                                    *)
(* The value of parameter CHOICE determines the index file to be dis_ *)
(* played :                                                           *)
    1 - first index data file STORE:INDEX1.DATA                      *)
    2 - second index data file STORE:INDEX2.DATA                     *)
    3 - third index data file STORE:INDEX3.DATA                      *)
(*  4 - all three index data files                                    *)
(*                                                                    *)
```

```
   The display table consists of four columns :                          *)
   (1) the word                                                          *)
(* (2) total number of sound unit of this word                          *)
(* (3) index of this word element in its index data file                *)
(* (4) status of this word element, letter T or F, respectively, re_    *)
(*     presenting whether this word element exists in or has been de_   *)
(*     leted from the dictionary                                        *)
(*                                                                      *)
(* Input :                                                              *)
(*   CHOICE - parameter                                                 *)
(*                                                                      *)
(* Output :                                                             *)
(*   Content of index data files on the screen                          *)
(*                                                                      *)
(* The following procedures are called :                                *)
(*   DISKETTE_ONLINE - UTILITY unit, checks whether the STORE: dis_     *)
(*     kette is on line. Open the existing first index data file when   *)
(*     diskette on line status has been confirmed.                      *)
(*   DO_PRINT - local procedure, it carries out the actual processing   *)
(*     of displaying the content of an index data file when the file    *)
(*     has been opened and the file window variable is provided.        *)
(*   RESET - build in file I/O procedure, open an existing index data   *)
(*     file.                                                            *)
(*   CLOSE - build in file I/O procedure, closes a given file.          *)
                                                                         *)
   The following global variables of the GLOBAL unit are used :         *)
     I1FNAME - filename of the first index data file.                    *)
(*   IFILE1 - file window variable for the first index data file.       *)
(*   I2FNAME - filename of the second index data file.                  *)
(*   IFILE2 - file window variable for the second index data file.      *)
(*   I3FNAME - filename of the third index data file.                   *)
(*   IFILE3 - file window variable for the third index data file.       *)
(* ---------------------------------------------------------------------*)

PROCEDURE PRT_ENTRIES ;

VAR CHECK : INTEGER ;

BEGIN

  PAGE(OUTPUT) ;
  WRITELN('************************************') ;
  WRITELN(' USE CRTL-S TO STOP OUTPUT STREAM') ;
  WRITELN(' ANOTHER CRTL-S TO RESUME PRINTING') ;
  WRITELN('************************************') ;
  WRITELN;
  WRITELN('                        VOICE') ; ;
  WRITELN('            WORD         UNIT INDEX STATUS') ;
  WRITELN('            ----        ----- ----- ------') ;

  CHECK := 3 ;              (* existing first index data file indicator *)
  DISKETTE_ONLINE( CHECK ) ;     (* check STORE: diskette on line by *)
                                 (* opening first index data file    *)
  CLOSE ( IFILE1 ) ;             (* close it *)
```

```
 I-*)                                (* compiler option, no I/O checking *)

(* First index data file with the words elements start with letters  *)
(* in the range of A,..,H.                                           *)

IF ( CHOICE = 1 OR CHOICE = 4 ) THEN       (* it is the choice or *)
BEGIN                                      (* display all three   *)

  RESET(IFILE1, I1FNAME) ;          (* open the existing index file *)

  IF ( IORESULT = 0 ) THEN          (* process only after the file *)
    DO_PRINT(IFILE1)                (* is opened successfully      *)
  ELSE
    WRITELN('> ERROR IN OPENING FIRST INDEX DATA FILE ') ;

  CLOSE(IFILE1)                     (* close the file *)

END ;

(* Second index data file with the words elements start with letters *)
(* in the range of I,..,P.                                           *)

IF ( CHOICE = 2 OR CHOICE = 4 ) THEN
BEGIN

  RESET(IFILE2, I2FNAME) ;

  IF ( IORESULT = 0 ) THEN
    DO_PRINT(IFILE2)
  ELSE
    WRITELN('> ERROR IN OPENING SECOND INDEX DATA FILE ') ;

  CLOSE(IFILE2)

END ;

(* Third index data file with the words elements start with letters  *)
(* in the range of Q,..,Z.                                           *)

IF ( CHOICE = 3 OR CHOICE = 4 ) THEN
BEGIN

  RESET(IFILE3, I3FNAME) ;

  IF ( IORESULT = 0 ) THEN
    DO_PRINT(IFILE3)
  ELSE
    WRITELN('> ERROR IN OPENING THIRD INDEX DATA FILE ') ;

  CLOSE(IFILE3)

 END

(*$I+*)                             (* compiler option, resume I/O checking *)
```

```
(* ------------------------------------------------------------------ *)
(* This procedure deletes the dictionary, in other words, all the     *)
(* index data files and binary data files are deleted permanently from *)
(* the STORE: storage diskette.                                       *)
(*                                                                    *)
(* Input :                                                            *)
(*   None                                                             *)
(*                                                                    *)
(* Output :                                                           *)
(*   None                                                             *)
(*                                                                    *)
(* The following procedures are called :                              *)
(*   DISKETTE_ONLINE - UTILITY unit, checks whether the STORE: dis_   *)
(*     kette is on line. Open the existing first index data file when *)
(*     its on line status has been confirmed.                         *)
(*   CLOSE - build in file I/O procedure, closes a given file and then *)
(*     deletes it permanently from the diskette.                      *)
(*   RESET - build in file I/O procedure, opens an existing file with *)
(*     the given filename.                                            *)
(*                                                                    *)
(* The following global variables of the GLOBAL unit are used :       *)
(*   VOLUMENAME - contains the storage diskette name which is STORE:  *)
(*   I1FNAME - filename of the first index data file.                  *)
(*   IFILE1 - file window variable for the first index data file.     *)
(*   I2FNAME - filename of the second index data file.                *)
(*   IFILE2 - file window variable for the second index data file.    *)
(*   I3FNAME - filename of the third index data file.                 *)
(*   IFILE3 - file window variable for the third index data file.     *)
(*   B1FNAME - filename of the first binary data file.                *)
(*   BFILE1 - file window variable for the first binary data file.    *)
(*   B2FNAME - filename of the second binary data file.               *)
(*   BFILE2 - file window variable for the second binary data file.   *)
(*   B3FNAME - filename of the third binary data file.                *)
(*   BFILE3 - file window variable for the third binary data file.    *)
(* ------------------------------------------------------------------ *)

PROCEDURE CLR_DICTIONARY ;

VAR IDX : INTEGER ;
    YN : CHAR ;

BEGIN

  PAGE(OUTPUT) ;
  WRITELN('> CLEAR ALL DATAFILES OF DICTIONARY ON DISKETTE ',VOLUMENAME) ;
  WRITELN ;
  WRITE('  ALL CLEAR TO GO ? [Y/N] ') ;
  READLN(YN) ;
  WRITELN ;

   F ( YN = 'Y' ) THEN
  BEGIN

    IDX := 3 ;              (* existing first index data file indicator *)
    DISKETTE_ONLINE(IDX) ;          (* check & request STORE: diskette *)
```

```
                                        (* diskette on line, open the file *)
    CLOSE(IFILE1, PURGE) ;              (* close and delete the file *)

(*$I-*)                               (* compiler option, no I/O checking *)

    (* second index data file *)

    RESET( IFILE2, I2FNAME ) ;                  (* open the file *)
    IF ( IORESULT = 0 ) THEN              (* delete it permanently *)
      CLOSE(IFILE2, PURGE) ;              (* when it exists        *)

    (* third index data file *)

    RESET( IFILE3, I3FNAME ) ;
    IF ( IORESULT = 0 ) THEN
      CLOSE(IFILE3, PURGE) ;

    (* first binary data file *)

    RESET( BFILE1, B1FNAME ) ;
    IF ( IORESULT = 0 ) THEN
      CLOSE(BFILE1, PURGE) ;

    (* second binary data file *)

    RESET( BFILE2, B2FNAME ) ;
    IF ( IORESULT = 0 ) THEN
      CLOSE(BFILE2, PURGE) ;

    (* third binary data file *)

    RESET( BFILE3, B3FNAME ) ;
    IF ( IORESULT = 0 ) THEN
      CLOSE(BFILE3, PURGE) ;

(*$I+*)                           (* compiler option, resumes I/O checking *)

    WRITELN('> ALL DATA FILES OF DICTIONARY HAS BEEN CLEARED')

  END

  ELSE WRITELN('> OPERATION IS ABORTED !!!')

END ;
```

```
  ------------------------------------------------------------------- *)
  This procedure builds all binary data elements of the dictionary,   *)
(* in other words, the three binary data files of the whole dictionary *)
(* in the STORE: storage diskette are created.                         *)
(*                                                                     *)
(* The actual process of creating the content of an binary data file   *)
(* is accomplished by calling the GETVOICE procedure of UTILITY unit,  *)
(* providing the binary data file and the related index data file have *)
(* been opened.                                                        *)
(*                                                                     *)
(* The first letters of all the words which associate with a binary    *)
(* data file belong to the same set of alphabet letters, each one of   *)
(* the three binary data files is associated with a distinct set of    *)
(* alphabet letters.                                                   *)
(*                                                                     *)
(* Input :                                                             *)
(*   None                                                              *)
(*                                                                     *)
(* Output :                                                            *)
(*   Three binary data files of the dictionary                         *)
(*                                                                     *)
(* The following procedures are called :                               *)
(*   DISKETTE_ONLINE - UTILITY unit, checks whether the STORE: dis_    *)
(*     kette is on line. Creates new and empty first binary data file  *)
     ,  when its on line status has been confirmed.                    *)
     .  GETVOICE - UTILITY unit, builds the content of a given binary data*)
        file where the first letters of all the word elements belong    *)
(*      to a specific set of alphabet letters.                          *)
(*   CLOSE - build in file I/O procedure, closes a given file and      *)
(*     makes it permanent.                                             *)
(*   REWRITE - build in file I/O procedure, creates a new and empty    *)
(*     binary data file with the given filename.                       *)
(*   RESET - build in file I/O procedure, opens an existing index data *)
(*     file.                                                           *)
(*                                                                     *)
(* The following global variables of the GLOBAL unit are used :        *)
(*   BFILE1 - file window variable for the first binary data file.     *)
(*   I1FNAME - filename of the first index data file.                  *)
(*   IFILE1 - file window variable for the first index data file.      *)
(*   SET1CHR - set of the first letters of all the word elements of    *)
(*     the first index data file.                                      *)
(*   SETCHRS - contains the current set of first letters.              *)
(*   CHRS1 - display message for the SET1CHR set of letters.           *)
(*   B2FNAME - filename of the second binary data file.                *)
(*   BFILE2 - file window variable for the second binary data file.    *)
(*   I2FNAME - filename of the second index data file.                 *)
(*   IFILE2 - file window variable for the second index data file.     *)
(*   SET2CHR - set of the first letters of all the word elements of    *)
(*     the second index data file.                                     *)
     ,  CHRS2 - display message for the SET2CHR set of letters.        *)
     .  B3FNAME - filename of the third binary data file.              *)
     .  BFILE3 - file window variable for the third binary data file.  *)
     .  I3FNAME - filename of the third index data file.               *)
(*   IFILE3 - file window variable for the third index data file.      *)
(*   SET3CHR - set of the first letters of all the word elements of    *)
```

```
      the third index data file.                                          *)
    CHRS3 - display message for the SET3CHR set of letters.               *)
(* ---------------------------------------------------------------------- *)

PROCEDURE BLD_VOICE ;

VAR CHECKNUM : INTEGER ;
    KEYIN : CHAR ;

BEGIN

  PAGE(OUTPUT) ;
  GOTOXY(0, 5) ;
  WRITELN('-------------------------------------') ;
  WRITELN(' BUILD THE VOICE PART OF DICTIONARY') ;
  WRITELN(' VOICE INPUT DEVICE CONNECTED ?') ;
  WRITELN(' STORAGE DISKETTE ON LINE ?') ;
  WRITELN(' PRESS A KEY WHEN READY TO GO') ;
  WRITELN('-------------------------------------') ;
  WRITELN ;
  READLN(KEYIN) ;

  (* first binary data file which associates with the first index data *)
  (* file having starting letters of all word elements in the range of *)
  (* A,..,H.                                                           *)

  CHECKNUM := 2 ;                (* new first binary data file indicator *)
  DISKETTE_ONLINE(CHECKNUM) ;    (* check & request STORE: diskette on   *)
                                 (* line, create the new binary data file*)
  RESET(IFILE1, I1FNAME) ;       (* open existing first index data file *)

  SETCHRS := SET1CHR ;                    (* first set of starting letters *)
  GETVOICE(BFILE1, IFILE1, CHRS1) ;             (* create content of this *)
                                                (* new binary data file   *)
  CLOSE(BFILE1, LOCK) ;       (* make this new file permanent & close it *)
  CLOSE(IFILE1) ;                (* close the index data file *)

  (* second binary data file which associates with the second index    *)
  (* data file having starting letters of all word elements in the     *)
  (* range of I,..,P.                                                  *)

  REWRITE(BFILE2, B2FNAME) ;      (* create new second binary data file *)

  RESET(IFILE2, I2FNAME) ;

  SETCHRS := SET2CHR ;
  GETVOICE(BFILE2, IFILE2, CHRS2) ;

  CLOSE(BFILE2, LOCK) ;
  CLOSE(IFILE2) ;

  (* third binary data file which associates with the third index data *)
  (* file having starting letters of all word elements in the range of *)
  (* Q,..,Z.                                                           *)
```

```
  <EWRITE(BFILE3, B3FNAME) ;

  RESET(IFILE3, I3FNAME) ;

  SETCHRS := SET3CHR ;
  GETVOICE(BFILE3, IFILE3, CHRS3) ;

  CLOSE(BFILE3, LOCK) ;
  CLOSE(IFILE3) ;

  WRITELN ;
  WRITELN('> END OF THE VOICE INPUT PART') ;
  WRITELN('  ---------------------------') ;
  WRITELN
END ;

(* The main program is empty. No need to define any process here.      *)

BEGIN


END .
```

```
(* ------------------------------------------------------------------ *)
(* This is the MODULE2 unit of Voice/Digital and Digital/Voice Conver_ *)
(* sion on a Microcomputer project.                                    *)
(*                                                                     *)
(* The Pascal language procedures here do update, insertion and dele_  *)
(* tion processes on the dictionary.                                   *)
(*                                                                     *)
(* The GLOBAL and UTILITY units are used and the assembly language     *)
(* routine DIGITAL is called here.                                     *)
(*                                                                     *)
(* The default volumename of the storage diskette is STORE: and it is  *)
(* the value of VOLUMENAME global variable. When user wants an alter_  *)
(* native storage diskette name, it should be assigned to VOLUMENAME   *)
(* variable in the initialization section of the application program   *)
(* and the GLOBAL unit must be USED by the program.                    *)
(*                                                                     *)
(* The compiler option $S++ is invoked here, more memory space is      *)
(* available for the compiling process but the speed is decreased.     *)
(* ------------------------------------------------------------------ *)

(*$S++*)                          (* compiler option *)

UNIT MODULE2 ;


INTERFACE

  USES GLOBAL, UTILITY ;                    (* declare the units used *)

  (* declare all the procedures of this module here *)

  PROCEDURE ADD_XENTRIES ;
  PROCEDURE ADD_BENTRIES ;
  PROCEDURE CMB_XENTRIES ;
  PROCEDURE CMB_BENTRIES ;
  PROCEDURE DO_DELETE ;
  PROCEDURE DO_CLNUP ;
  PROCEDURE IMPROVE_SOUND ;


IMPLEMENTATION

  (* declare the assembly language routine to be called here *)

  PROCEDURE DIGITAL(VAR BDATA:SOUND; DTEMPO,DUNIT:INTEGER) ; EXTERNAL ;
```

```
   ------------------------------------------------------------------- *)
   This procedure inserts new index data records to the index data     *)
(* files. For each index file, the new index data records will have    *)
(* index numbers in a sequence starting with the total number of data  *)
(* record in the associated binary data file. The first data record of *)
(* the binary file is associated with zero index number.               *)
(*                                                                     *)
(* For each index data file, a temporary index file is created to store*)
(* the new index data records. The word elements of these new index    *)
(* records must be input in alphabetical ascending order. The actual   *)
(* process of creating the content of a temporary index file with new  *)
(* index records is accomplished by calling the BLDIDX procedure of    *)
(* UTILITY unit.                                                       *)
(*                                                                     *)
(* Input :                                                             *)
(*   None                                                              *)
(*                                                                     *)
(* Output :                                                            *)
(*   Three temporay index data files called X1.DATA, X2.DATA and       *)
(*   X3.DATA                                                           *)
(*                                                                     *)
(* The following procedures are called :                               *)
(*   CNT2_ELEM - UTILITY unit, it counts the total number of digitized *)
(*     binary data records in a binary data file providing the data    *)
     file has been opened by the calling procedure and the file win_  *)
     dow variable is given.                                           *)
   BLDIDX - UTILITY unit, builds the content of a given index data    *)
(*   file providing the file has been opened by the calling procedu_  *)
(*   re and the file window variable is given.                        *)
(*   DISKETTE_ONLINE - UTILITY unit, checks and requests that the     *)
(*     STORE: diskette be on line.                                    *)
(*   CLOSE - build in file I/O procedure, closes a file.              *)
(*   RESET - build in file I/O procedure, opens an existing binary    *)
(*     data file.                                                     *)
(*   REWRITE - build in file I/O procedure, creates a new and empty   *)
(*     temporary index data file.                                     *)
(*   CONCAT - build in string function, concatenates two or more cha_ *)
(*     racter strings.                                                *)
(*                                                                    *)
(* The following global variables of the GLOBAL unit are used :       *)
(*   IFILE1 - file window variable for the first index data file. It  *)
(*     is also used for the three temporary index data files, one at a*)
(*     time.                                                          *)
(*   B1FNAME - filename of the first binary data file.                *)
(*   BFILE1 - file window variable for the first binary data file.    *)
(*   SETCHRS - contain the current set of starting letters.           *)
(*   SET1CHR - set of the starting letters of all the words elements  *)
(*     of the first index data file.                                  *)
(*   CHRS1 - display message string for the SET1CHR set of letters.   *)
   B2FNAME - filename of the second binary data file.                 *)
   BFILE2 - file window variable for the second binary data file.     *)
   SET2CHR - set of the starting letters of all the words elements    *)
     of the second index data file.                                   *)
(*   CHRS2 - display message string for the SET2CHR set of letters.   *)
(*   B3FNAME - filename of the third binary data file.                *)
```

```
    BFILE3 - file window variable for the third binary data file.      *)
    SET3CHR - set of the starting letters of all the words elements    *)
(*    of the third index data file.                                    *)
(*  CHRS3 - display message string for the SET3CHR set of letters.     *)
(*  VOLUMENAME - contains the storage diskette name STORE:             *)
(*                                                                     *)
(* ------------------------------------------------------------------- *)

PROCEDURE ADD_XENTRIES ;

VAR CHECKNUM, STRTIDX : INTEGER ;

BEGIN

  PAGE(OUTPUT) ;
  WRITELN('----------------------------') ;
  WRITELN(' INDEX DATA FILES INSERTION ') ;
  WRITELN('----------------------------') ;
  WRITELN ;

  (* Check and request the STORE: storage diskette be on line by open_ *)
  (* ing the existing first index data file.                           *)

  CHECKNUM := 3 ;                (* existing first index file indicator *)
  ISKETTE_ONLINE(CHECKNUM) ;                (* open the index data file *)
  CLOSE( IFILE1 ) ;              (* close it *)

  (* insert new index data records to the first index data file, in    *)
  (* other words, create the associated temporay index data file. The  *)
  (* word elements of the new index records have starting letters in   *)
  (* the range of A,..,H.                                              *)

  RESET(BFILE1,B1FNAME) ;        (* open the existing first binary file *)
  CNT2_ELEM(BFILE1, STRTIDX) ;      (* counts the data records to get   *)
                                    (* the new records' 1st index number*)
  CLOSE(BFILE1) ;                (* close the binary file *)

  REWRITE(IFILE1,CONCAT(VOLUMENAME,'X1.DATA')) ;  (* create new tempo_ *)
                                                  (* rary index file   *)
  SETCHRS := SET1CHR ;                  (* first set of starting letters *)
  BLDIDX(IFILE1, CHRS1, STRTIDX) ;            (* build the content *)
  CLOSE(IFILE1, LOCK) ;                       (* close the new index file *)

  (* insert new index data records to the second index data file, in   *)
  (* other words, create the associated temporay index data file. The  *)
  (* word elements of the new index records have starting letters in   *)
  (* the range of I,..,P.                                              *)

  RESET(BFILE2, B2FNAME) ;
  CNT2_ELEM(BFILE2, STRTIDX) ;
  CLOSE(BFILE2) ;

  REWRITE(IFILE2,CONCAT(VOLUMENAME,'X2.DATA')) ;
  SETCHRS := SET2CHR ;                  (* second set of starting letters *)
  BLDIDX(IFILE2, CHRS2, STRTIDX) ;
```

```
  CLOSE(IFILE2, LOCK) ;

  (* insert new index data records to the third index data file, in    *)
  (* other words, create the associated temporay index data file. The  *)
  (* word elements of the new index records have starting letters in   *)
  (* the range of Q,..,Z.                                              *)

  RESET(BFILE3, B3FNAME) ;
  CNT2_ELEM(BFILE3, STRTIDX) ;
  CLOSE(BFILE3) ;

  REWRITE(IFILE3,CONCAT(VOLUMENAME,'X3.DATA')) ;
  SETCHRS := SET3CHR ;                  (* third set of starting letters *)
  BLDIDX(IFILE3, CHRS3, STRTIDX) ;
  CLOSE(IFILE3, LOCK)

END ;
```

```
(* ------------------------------------------------------------------ *)
   This procedure inserts new binary data records to the binary data   *)
(* files. For each binary file, the new binary data records will be    *)
(* inserted starting at the end of the binary data file. The input of  *)
(* each digitized binary data record must be in the same sequence num_ *)
(* ber as its associated index data record.                            *)
(*                                                                     *)
(* For each binary data file, a temporary binary file is created to    *)
(* store the new binary data records. The actual process of creating   *)
(* the content of a temporary binary data file with new binary records *)
(* is accomplished by calling the GETVOICE procedure of UTILITY unit.  *)
(*                                                                     *)
(* Input :                                                             *)
(*   None                                                              *)
(*                                                                     *)
(* Output :                                                            *)
(*   Three temporay binary data files called B1.DATA, B2.DATA and      *)
(*   B3.DATA                                                           *)
(*                                                                     *)
(* The following procedures are called :                               *)
(*   GETVOICE - UTILITY unit, builds the content of a given binary data*)
(*     file providing the file has been opened by the calling procedu_ *)
(*     re and the file window variable is given.                       *)
(*   DISKETTE_ONLINE - UTILITY unit, checks and requests that the      *)
(*     STORE: diskette be on line.                                     *)
(*   REWRITE - build in file I/O procedure, creates a new and empty    *)
(*     temporary binary data file.                                     *)
(*   RESET - build in file I/O procedure, opens an existing temporary  *)
(*     index data file.                                                *)
(*   CLOSE - build in file I/O procedure, closes a file.               *)
(*   CONCAT - build in string function, concatenates two or more cha_  *)
(*     racter strings.                                                 *)
(*                                                                     *)
(* The following global variables of the GLOBAL unit are used :        *)
(*   IFILE1 - file window variable for the first index data file. It   *)
(*     is also used for the first temporary index data file.           *)
(*   BFILE1 - file window variable for the first temporary binary file.*)
(*   SETCHRS - contain the current set of starting letters.            *)
(*   SET1CHR - set of the starting letters of all the words elements   *)
(*     of the associated first index data file.                        *)
(*   CHRS1 - display message string for the SET1CHR set of letters.    *)
(*   IFILE2 - file window variable for the second temporary index file.*)
(*   BFILE2 - file window variable for the second temporary binary file*)
(*   SET2CHR - set of the starting letters of all the words elements   *)
(*     of the associated second index data file.                       *)
(*   CHRS2 - display message string for the SET2CHR set of letters.    *)
(*   IFILE3 - file window variable for the third temporary index file. *)
(*   BFILE3 - file window variable for the third temporary binary file.*)
(*   SET3CHR - set of the starting letters of all the words elements   *)
(*     of the associated third index data file.                        *)
(*   CHRS3 - display message string for the SET3CHR set of letters.    *)
(*   VOLUMENAME - contains the storage diskette name STORE:            *)
(* ------------------------------------------------------------------ *)

PROCEDURE ADD_BENTRIES ;
```

```
  : CHECKNUM, STRTIDX : INTEGER ;

BEGIN

  PAGE(OUTPUT) ;
  WRITELN('------------------------------------------------------------') ;
  WRITELN(' APPENDING NEW BINARY DATA RECORDS INTO BINARY DATA FILES ') ;
  WRITELN('------------------------------------------------------------') ;
  WRITELN ;

  (* Check and request the STORE: storage diskette be on line by open_ *)
  (* ing the existing first index data file.                           *)

  CHECKNUM := 3 ;                (* existing first index file indicator *)
  DISKETTE_ONLINE(CHECKNUM) ;                 (* open the index data file *)
  CLOSE( IFILE1 ) ;              (* close it *)

  (* Append new binary data records to the first binary data file, in  *)
  (* other words, create the associated temporay binary data file.     *)

                                    (* create new temporary binary file *)
  REWRITE(BFILE1,CONCAT(VOLUMENAME,'B1.DATA')) ;
                                  (* open existing temporary index file *)
   ESET(IFILE1, CONCAT(VOLUMENAME,'X1.DATA')) ;

  ;ETCHRS := SET1CHR ;
  GETVOICE(BFILE1, IFILE1, CHRS1) ;          (* create content of     *)
                                             (* temporary binary file *)
  CLOSE(BFILE1, LOCK) ;        (* close the temporary binary file *)
  CLOSE(IFILE1) ;              (* close the temporary index file *)

  (* Append new binary data records to the second binary data file, in *)
  (* other words, create the associated temporay binary data file.     *)

  REWRITE(BFILE2,CONCAT(VOLUMENAME,'B2.DATA')) ;
  RESET(IFILE2, CONCAT(VOLUMENAME,'X2.DATA')) ;

  SETCHRS := SET2CHR ;
  GETVOICE(BFILE2, IFILE2, CHRS2) ;

  CLOSE(BFILE2, LOCK) ;
  CLOSE(IFILE2) ;

  (* Append new binary data records to the third binary data file, in  *)
  (* other words, create the associated temporay binary data file.     *)

  REWRITE(BFILE3,CONCAT(VOLUMENAME,'B3.DATA')) ;
  RESET(IFILE3, CONCAT(VOLUMENAME,'X3.DATA')) ;

  ;ETCHRS := SET3CHR ;
  ;ETVOICE(BFILE3, IFILE3, CHRS3) ;

  CLOSE(BFILE3, LOCK) ;
  CLOSE(IFILE3)
```

```
(* ------------------------------------------------------------------ *)
(* This local procedure makes an exact copy of an index data file by *)
(* copying each consecutive index data record to an empty index data *)
(* file. Both index files have been opened by the calling procedure  *)
(* and the file window variables are provided.                       *)
(*                                                                   *)
(* Input :                                                           *)
(*   DEST - parameter, file window variable of the destination index *)
(*          data file.                                               *)
(*   SRCE - parameter, file window variable of the source index data *)
(*          file.                                                    *)
(*                                                                   *)
(* Output :                                                          *)
(*   Content of the destination (DEST) index data file               *)
(*                                                                   *)
(* The following procedures are called :                             *)
(*   EOF - build in file I/O function, checks whether the end of a   *)
(*         specified file has been reached.                          *)
(*   PUT - build in file I/O function, advances the file window va_  *)
(*     riable to the next record and puts the content of file buffer *)
(*     variable into this record.                                    *)
(*   GET - build in file I/O function, advances the file window va_  *)
(*     riable to the next record and moves the content of this       *)
(*     record into the file buffer variable.                         *)
(*                                                                   *)
(* The following global variables of the GLOBAL unit are used :      *)
(*   None                                                            *)
(* ------------------------------------------------------------------ *)

PROCEDURE COPY_XFILE(VAR DEST, SRCE:INDEXFILE) ;

BEGIN

  WHILE ( NOT(EOF(SRCE)) ) DO        (* loop processing one record at *)
  BEGIN                              (* a time until the end of file  *)

    DEST^.STRG := SRCE^.STRG ;            (* assigning word element *)
    DEST^.UNITT := SRCE^.UNITT ;       (* total number of sound unit *)
    DEST^.IDXX := SRCE^.IDXX ;                      (* index number *)
    DEST^.STATUS := SRCE^.STATUS ;      (* valid and existing status *)

    PUT(DEST) ;            (* advancing the file window variable *)
    GET(SRCE)              (* advancing the file window variable *)

  END

END ;


(* ------------------------------------------------------------------ *)
   This procedure is the second and last step of inserting new index  *)
   data records into the index data files. For each pair of index data *)
   file and its associated temporary index data file, they are combined*)
(* and the result is a bigger index data file. In each file of the pair*)
(* of index data files, the word elements of all the index data records*)
```

```
   are in ascending alphabetical order. In the resulting index data    *)
   file, the alphabetical order is kept.                               *)
(*                                                                     *)
(* The actual process of combining a pair of index data files is car_  *)
(* ried out in three substeps :                                        *)
(* (1) Merge Sort this pair of index data files in alphabetical as_    *)
(*     cending order, the result is a combined index data file.        *)
(*     This substep is accomplished by calling the FILE_SORT procedure.*)
(* (2) Delete the pair of index data files.                            *)
(*     Create a new index data file with the same filename as the ori_ *)
(*     ginal index data file which was just deleted.                   *)
(* (3) Copy the content of the combined index data file into this      *)
(*     new and empty index data file. This substep is accomplished by  *)
(*     calling the COPY_XFILE local procedure.                         *)
(*                                                                     *)
(* Input :                                                             *)
(*   Three pairs of index data files and temporary index data files.   *)
(*     INDEX1.DATA and X1.DATA                                         *)
(*     INDEX2.DATA and X2.DATA                                         *)
(*     INDEX3.DATA and X3.DATA                                         *)
(*                                                                     *)
(* Output :                                                            *)
(*   Three index data files, each of them is the combination result of *)
(*   its original index data file and the associated temporary index   *)
    data file.                                                         *)
                                                                       *)
   The following procedures are called :                               *)
(*   COPY_XFILE - local procedure, it copies the content of a combina_ *)
(*     tion index data file into an empty index data file. The data    *)
(*     files have been opened by the calling procedure and both file   *)
(*     window variables are provided.                                  *)
(*   FILE_SORT - UTILITY unit, builds the content of a combined        *)
(*     index data file from a pair of index data file and temporary    *)
(*     index data file, the word elements of all index records are in  *)
(*     alphabetical ascending order. All three data files have been    *)
(*     opened and the file window variables are provided.              *)
(*   CLOSE - build in file I/O procedure, closes an index data file.   *)
(*   RESET - build in file I/O procedure, opens an existing index data *)
(*           file.                                                     *)
(*   REWRITE - build in file I/O procedure, creates a new and empty    *)
(*     index data file.                                                *)
(*   CONCAT - build in string function, concatenates two or more cha_  *)
(*     racter strings.                                                 *)
(*                                                                     *)
(* The following global variables of the GLOBAL unit are used :        *)
(*   IFILE1 - file window variable for the three index data files, it  *)
(*            is used for only one data file at a time.                *)
(*   IFILE2 - file window variable for the three temporary index data  *)
(*            files, it is used for only one data file at a time.      *)
(*   IFILE3 - file window variable for the three combined index data   *)
(*            files, it is used for only one data file at a time.      *)
    I1FNAME - filename of the first index data file.                   *)
    I2FNAME - filename of the second index data file.                  *)
(*   I3FNAME - filename of the third index data file.                  *)
(*   VOLUMENAME - contains the storage diskette name STORE:            *)
```

```
  ------------------------------------------------------------------ *)

PROCEDURE CMB_XENTRIES ;

VAR I : INTEGER ;
    FN1, FN2 : FILENAME ;

BEGIN

  PAGE(OUTPUT) ;
  WRITELN('> DOING SORT MERGE ON INDEX DATA FILES NOW') ;
  WRITELN('  PLEASE BE PATIENT') ;
  WRITELN('  ----------------------------------------') ;

  FOR I := 1 TO MAXFILE DO          (* loop, each pass processes only one *)
  BEGIN                             (* pair of related index data files   *)

    IF ( I = 1 ) THEN               (* first pass *)
    BEGIN
      FN1 := I1FNAME ;                                  (* uses first pair of  *)
      FN2 := CONCAT(VOLUMENAME,'X1.DATA')               (* related index files *)
    END

    ELSE IF ( I = 2 ) THEN          (* second pass *)
    BEGIN
      FN1 := I2FNAME ;                                  (* uses second pair of *)
      FN2 := CONCAT(VOLUMENAME,'X2.DATA')               (* related index files *)
    END
    ELSE                            (* third and last pass *)
    BEGIN
      FN1 := I3FNAME ;                                  (* uses third pair of  *)
      FN2 := CONCAT(VOLUMENAME,'X3.DATA')               (* related index files *)
    END ;

    RESET(IFILE1, FN1) ;      (* open existing index data file *)
    RESET(IFILE2, FN2) ;      (* open existing temporary index data file *)
                              (* create a new combined index data file *)
    REWRITE(IFILE3, CONCAT(VOLUMENAME,'Y.DATA')) ;

    FILE_SORT(IFILE1, IFILE2, IFILE3) ;                 (* merge pair of    *)
                                                        (* index data files *)

    CLOSE(IFILE1, PURGE) ;          (* delete original index data file *)
    CLOSE(IFILE2, PURGE) ;          (* delete the temporary index file *)
    CLOSE(IFILE3, LOCK) ;           (* close & keep the combined file *)

                              (* open existing combined index data file *)
    RESET(IFILE3, CONCAT(VOLUMENAME,'Y.DATA')) ;
    REWRITE(IFILE1, FN1) ;          (* create a new index data file *)

    COPY_XFILE(IFILE1, IFILE3) ;          (* copy content of index file *)

    CLOSE(IFILE1, LOCK) ;           (* close & keep the index data file *)
    CLOSE(IFILE3, PURGE)            (* delete the combined index file *)
```

```
C_ND
END ;
```

```
 * ------------------------------------------------------------------ *)
(* This local procedure appends a binary data file by copying each   *)
(* consecutive binary data record into the end of another binary      *)
(* data file. Both binary files have been opened by the calling pro_  *)
(* cedure and the file window variables are provided.                 *)
(*                                                                    *)
(* Input :                                                            *)
(*   DEST - parameter, file window variable of the destination        *)
(*          binary data file.                                         *)
(*   SRCE - parameter, file window variable of the source binary      *)
(*          data file.                                                *)
(*                                                                    *)
(* Output :                                                           *)
(*   Content of the destination (DEST) binary data file               *)
(*                                                                    *)
(* The following procedures are called :                              *)
(*   EOF - build in file I/O function, checks whether the end of a    *)
(*         specified file has been reached.                           *)
(*   PUT - build in file I/O function, advances the file window va_   *)
(*     riable to the next record and puts the content of file buffer  *)
(*     variable into this record.                                     *)
(*   GET - build in file I/O function, advances the file window va_   *)
(*     riable to the next record and moves the content of this        *)
(*     record into the file buffer variable.                          *)
(*                                                                    *)
(* The following global variables of the GLOBAL unit are used :       *)
(*   None                                                             *)
(* ------------------------------------------------------------------ *)

PROCEDURE COPY_BFILE(VAR DEST, SRCE:BINARYFILE) ;

BEGIN

  WHILE ( NOT(EOF(SRCE)) ) DO        (* loop processing one record at *)
  BEGIN                              (* a time until the end of file  *)

    DEST^ := SRCE^ ;          (* assigning the digitized binary data *)

    PUT(DEST) ;               (* advancing both file *)
    GET(SRCE)                 (* window variables    *)

  END

END ;


(* ------------------------------------------------------------------ *)
(* This procedure is the second and last step of inserting new binary *)
(* data records into the binary data files. For each binary data file, *)
(* its associated temporary binary data file is appended at the end of *)
(* the file and the result is a bigger binary data file.              *)
(*                                                                    *)
(* The actual process of combining a pair of binary data files is car_ *)
(* ried out in four substeps :                                        *)
(* (1) Create a new and empty combined binary data file. By calling   *)
```

```
      the COPY_BFILE procedure, the binary data file of the pair is   *)
      copied into this combined file.                                 *)
(* (2) By calling the COPY_BFILE procedure again, the temporary binary *)
(*     data file of the pair is copied at the end of this combined     *)
(*     file.                                                           *)
(* (3) Delete the pair of binary data files.                           *)
(*     Create a new binary data file with the same filename as the     *)
(*     original binary data file which was just deleted.               *)
(* (4) Copy the content of the combined binary data file into this     *)
(*     new and empty binary data file. This substep is accomplished    *)
(*     by calling the COPY_BFILE local procedure.                      *)
(*                                                                     *)
(* It is possible that the preceding combining process does not need   *)
(* the combined binary data file. The temporary binary data file can   *)
(* be copied (appended) directly at the end of its associated binary   *)
(* data file by calling the COPY_BFILE procedure only once. Apple II   *)
(* disk file needs contigous storage area, if the original binary data *)
(* file has another disk file right after it in the disk, no expansion *)
(* can be made. Therefore, the preceding simple method will not work   *)
(* most of the time.                                                   *)
(*                                                                     *)
(* Input :                                                             *)
(*   Three pairs of binary data file and temporary binary data files.  *)
(*     BINARY1.DATA and B1.DATA                                        *)
(*     BINARY2.DATA and B2.DATA                                        *)
      BINARY3.DATA and B3.DATA                                        *)
                                                                      *)
(* Output :                                                            *)
(*   Three binary data files, each of them is the combination result   *)
(*   of its original binary data file and the associated temporary     *)
(*   binary data file.                                                 *)
(*                                                                     *)
(* The following procedures are called :                               *)
(*   COPY_BFILE - local procedure, it copies the content of a source   *)
(*     binary data file into the end of a destination binary data file.*)
(*     Both data files have been opened by the calling procedure and   *)
(*     the file window variables are provided.                         *)
(*   CLOSE - build in file I/O procedure, closes a binary data file.   *)
(*   RESET - build in file I/O procedure, opens an existing binary     *)
(*           data file.                                                *)
(*   REWRITE - build in file I/O procedure, creates a new and empty    *)
(*     binary data file.                                               *)
(*   CONCAT - build in string function, concatenates two or more cha_  *)
(*     racter strings.                                                 *)
(*                                                                     *)
(* The following global variables of the GLOBAL unit are used :        *)
(*   BFILE1 - file window variable for the three binary data files, it *)
(*            is used for only one data file at a time.                *)
(*   BFILE2 - file window variable for the three temporary binary data *)
(*            files, it is used for only one data file at a time.      *)
(*   BFILE3 - file window variable for the three combined binary data  *)
(*            files, it is used for only one data file at a time.      *)
    B1FNAME - filename of the first binary data file.                  *)
(*   B2FNAME - filename of the second binary data file.                *)
(*   B3FNAME - filename of the third binary data file.                 *)
```

```
    VOLUMENAME - contains the storage diskette name STORE:                *)
   ------------------------------------------------------------------------ *)

PROCEDURE CMB_BENTRIES ;

VAR I : INTEGER ;
    FN1, FN2 : FILENAME ;

BEGIN

  PAGE(OUTPUT) ;
  WRITELN('> MERGEING BINARY DATA FILES NOW') ;
  WRITELN('  PLEASE BE PATIENT') ;
  WRITELN('  ------------------------------') ;

  FOR I := 1 TO MAXFILE DO          (* loop, each pass processes only a  *)
  BEGIN                             (* pair of related binary data files *)

    IF ( I = 1 ) THEN                        (* first pass *)
    BEGIN
      FN1 := B1FNAME ;                                (* first binary file *)
      FN2 := CONCAT(VOLUMENAME,'B1.DATA')             (* first temporary  *)
    END                                               (* binary data file *)

    ELSE IF ( I = 2 ) THEN                   (* second pass *)
    BEGIN
      FN1 := B2FNAME ;                                (* second binary file *)
      FN2 := CONCAT(VOLUMENAME,'B2.DATA')             (* second temporary *)
    END                                               (* binary data file *)

    ELSE                                     (* third pass *)
    BEGIN
      FN1 := B3FNAME ;                                (* third binary file *)
      FN2 := CONCAT(VOLUMENAME,'B3.DATA')             (* third temporary  *)
    END ;                                             (* binary data file *)

                                  (* create new combined binary data file *)
    REWRITE(BFILE3, CONCAT(VOLUMENAME,'Z.DATA')) ;
    RESET(BFILE1, FN1) ;                 (* open existing binary data file *)
    COPY_BFILE(BFILE3, BFILE1) ;         (* copy into combined binary file *)
    CLOSE(BFILE1, PURGE) ;                  (* delete the binary data file *)

    RESET(BFILE2, FN2) ;           (* open existing temporary binary file *)
    COPY_BFILE(BFILE3, BFILE2) ;              (* append to the end of *)
                                              (* combined binary file *)
    CLOSE(BFILE2, PURGE) ;            (* delete the temporary binary file *)
    CLOSE(BFILE3, LOCK) ;       (* close & keep the combined binary file *)

                               (* open existing combined binary data file *)
    RESET(BFILE3, CONCAT(VOLUMENAME,'Z.DATA')) ;
    REWRITE(BFILE1, FN1) ;                  (* create new binary data file *)
    COPY_BFILE(BFILE1, BFILE3) ;            (* copy into empty binary file *)

    CLOSE(BFILE1, LOCK) ;             (* close & keep the binary data file *)
    CLOSE(BFILE3, PURGE)              (* delete the combined binary file *)
```

```
C
C   END
C
    END ;
```

```
(* ------------------------------------------------------------- *)
(* This is a local procedure. For each index data file with the file *)
(* window variable provided, this procedure prompts the user to      *)
(* input the words to be deleted from the dictionary, pressing of    *)
(* RETURN key ends the process. All the input words must be in alpha_*)
(* betical ascending order and in the same set of starting letters.  *)
(*                                                                   *)
(* For each input word, if it is found as the word element of an     *)
(* index data record in the current index data file, this index re_  *)
(* cord is deleted from the index file. Actually, only the status    *)
(* element's value of the index record is changed to FALSE boolean   *)
(* value. The real deletion of index data records is accomplished by *)
(* calling the DO_CLNUP procedure. Each call of this procedure dele_ *)
(* tes words from the same index data file, in order words, words    *)
(* with starting letters in the same set of letters are deleted.     *)
(*                                                                   *)
(*  Input :                                                          *)
(*    IFILE1 - parameter, file window variable of an index data file *)
(*                                                                   *)
(*  Output :                                                         *)
(*    None                                                           *)
(*                                                                   *)
(*  The following procedures are called :                            *)
(*    LENGTH - build in string function, gets the length of a charac_*)
(*      ter string. The returned value is zero for NULL string.      *)
(*    EOF - build in file I/O function, indicates whether the end of *)
(*      a specified file has been reached.                           *)
(*    PUT - build in file I/O procedure, advances the file window va_*)
(*      riable to the next record and puts the content of file buffer*)
(*      variable into this record.                                   *)
(*    GET - build in file I/O procedure, advances the file window va_*)
(*      riable to the next record and moves the content of this re_  *)
(*      cord into the file buffer variable.                          *)
(*    SEEK - build in file I/O procedure, allows random access to    *)
(*      record. The file window variable (file pointer) is moved to  *)
(*      a specified record in a file. The record number (index num_  *)
(*      ber) of the specified record is provided.                    *)
(*                                                                   *)
(*  The following global variables of the GLOBAL unit are used :     *)
(*    None                                                           *)
(* ------------------------------------------------------------- *)

PROCEDURE GO_DELETE(VAR IFILE1:INDEXFILE) ;

VAR WRD : WORD ;
    IDX1, L : INTEGER ;
    FOUND, NOTIN : BOOLEAN ;

BEGIN

  WRITELN ;
  L := 1 ;
  IDX1 := 0 ;

  (* loop, each pass deletes only one word. The input words must be  *)
```

```
    (* in alphabetical ascending order with starting letters in the    *)
    (* same set of letters. The processing of this procedure is com_   *)
    (* pleted when the end of file has been encountered or a null word *)
    (* has been input.                                                 *)

    WHILE ( (L <> 0) AND (NOT(EOF(IFILE1))) ) DO
    BEGIN

      FOUND := FALSE ;              (* word is yet not found in dictionary *)
      NOTIN := FALSE ;                 (* assume word is in the dictionary *)
      WRITE('WORD: ') ;                     (* prompts user to input  *)
      READLN(WRD) ;                         (* the word to be deleted *)
      L := LENGTH(WRD) ;

      IF ( L <> 0 ) THEN                    (* only non null word input *)
      BEGIN

        WHILE ( (NOT(EOF(IFILE1))) AND
                (NOT(FOUND)) AND (NOT(NOTIN)) ) DO
        BEGIN

          IF ( WRD = IFILE1^.STRG ) THEN            (* find the match *)
          BEGIN
            FOUND := TRUE ;
            IFILE1^.STATUS := FALSE ;       (* update the status element*)
            SEEK(IFILE1, IDX1) ;            (* of the index data record *)
            PUT(IFILE1)                     (* in the index data file   *)
          END
          ELSE IF ( WRD < IFILE1^.STRG )        (* input word is not in *)
            THEN NOTIN := TRUE ;                (* the dictionary, base *)
                                                (* on the alpha. ascend.*)
                                                (* order characteristic *)
          GET( IFILE1 ) ;                      (* get next index record *)
          IDX1 := IDX1 + 1

        END

      END ;

      IF ( NOTIN ) THEN
        WRITELN('> WORD DOES NOT EXIST !')
      ELSE IF ( EOF(IFILE1) ) THEN
        WRITELN('> END OF FILE !') ;

      WRITELN

    END

  END ;

  (* ------------------------------------------------------------------ *)
  (* This procedure processes the deletion of words from the dictionary, *)
  (* in other words, related index data records are deleted from the    *)
  (* index data file of the dictionary.                                 *)
```

```
                                                                        *)
   The three index data files are processed in sequence, one after      *)
(* another. After an index file has been opened, the task of deleting   *)
(* index data records which associate with the same set of starting     *)
(* letters is passed to GO_DELETE local procedure, providing the file   *)
(* window variable is available.                                        *)
(*                                                                      *)
(* Input :                                                              *)
(*   Three index data files of the dictionary                           *)
(*                                                                      *)
(* Output :                                                             *)
(*   None                                                               *)
(*                                                                      *)
(* The following procedures are called :                                *)
(*   GO_DELETE - local procedure. It does the deletion of index data    *)
(*     records from a specified index data file when the file window    *)
(*     variable is given.                                               *)
(*   RESET - build in file I/O procedure, open an existing index data   *)
(*     file.                                                            *)
(*   CLOSE - build in file I/O procedure, close an index data file.     *)
(*                                                                      *)
(* The following global variables of the GLOBAL unit are used :         *)
(*   I1FNAME - filename of the first index data file.                    *)
(*   IFILE1 - file window variable for the first index data file.       *)
     CHRS1 - display message for the SET1CHR set of letters.            *)
     I2FNAME - filename of the second index data file.                  *)
     IFILE2 - file window variable for the second index data file.      *)
(*   CHRS2 - display message for the SET2CHR set of letters.            *)
(*   I3FNAME - filename of the third index data file.                   *)
(*   IFILE3 - file window variable for the third index data file.       *)
(*   CHRS3 - display message for the SET3CHR set of letters.            *)
(* ------------------------------------------------------------------ *)

PROCEDURE DO_DELETE ;

VAR FN, CHRS : FILENAME ;
    I : INTEGER ;
    INPKEY : CHAR ;

BEGIN

  PAGE(OUTPUT) ;
  WRITELN('-------------------------------------') ;
  WRITELN(' DELETE ENTRIES FROM THE DICTIONARY ') ;
  WRITELN('-------------------------------------') ;
  WRITELN ;

  FOR I := 1 TO MAXFILE DO                    (* loop, only process one  *)
  BEGIN                                       (* index file in each pass *)

    IF ( I = 1 ) THEN                         (* first index data file *)
    BEGIN
      CHRS := CHRS1 ;
      FN := I1FNAME
    END
```

```
    ELSE IF ( I = 2 ) THEN                    (* second index data file *)
    BEGIN
      CHRS := CHRS2 ;
      FN := I2FNAME
    END

    ELSE                                      (* third index data file *)
    BEGIN
      CHRS := CHRS3 ;
      FN := I3FNAME
    END ;

    WRITELN('WORDS START WITH ',CHRS) ;
    WRITELN('END OF INPUT, RETURN KEY ONLY') ;
    WRITELN('-----------------------------') ;

 (*$I-*)

    RESET(IFILE1, FN) ;                      (* open the index data file *)

    IF ( IORESULT <> 0 ) THEN
      WRITELN('> ERROR IN OPENING INDEX DATA FILE ',FN)
    ELSE
      GO_DELETE(IFILE1) ;                    (* process deletion of *)
                                             (* index data records  *)
    CLOSE(IFILE1) ;                          (* close the file *)

(*$I+*)

    WRITE('RETURN KEY: ') ;
    READLN(INPKEY)

  END

END ;
```

```
   ------------------------------------------------------------------ *)
   This is the second and last step of deleting words from the dictio_ *)
(* nary. The temporary deleted index data records and their associated *)
(* binary data records are deleted permanently, respectively, from the *)
(* index data files and binary data files of the dictionary.           *)
(*                                                                     *)
(* The data records deletion process is accomplished by copying the    *)
(* non deleted data records into a temporary data file. After the ori_ *)
(* ginal data file has been deleted and recreated with empty content,  *)
(* the content of the temporay data file is copied into the empty data *)
(* file by calling COPY_XFILE or COPY_BFILE procedure, respectively,   *)
(* for the index or binary data file.                                  *)
(*                                                                     *)
(* Input :                                                             *)
(*   Three pairs of index and binary data files.                       *)
(*                                                                     *)
(* Output :                                                            *)
(*   Clean up version of the three pairs of index and binary data files*)
(*                                                                     *)
(* The following procedures are called :                               *)
(*   EOF - build in file I/O procedure, it indicates whether the end   *)
(*         of a specified file has been réached.                       *)
(*   GET - build in file I/O procedure, it advances the file window    *)
(*     variable to the next record and moves the content of this       *)
(*     record to the file buffer variable.                             *)
(*   PUT - build in file I/O procedure, it advances the file window    *)
(*     variable to the next record and puts the content of the file    *)
(*     buffer variable into this record.                               *)
(*   RESET - build in file I/O procedure, open an existing data file.  *)
(*   CLOSE - build in file I/O procedure, closes a given file.         *)
(*   REWRITE - build in file I/O procedure, creates a new and empty    *)
(*             data file with the given filename.                      *)
(*   GO_ACT1 - local procedure, it goes through an opened index data   *)
(*     file and marks down all the deleted index data records using    *)
(*     the IARRAY array.                                               *)
(*   GO_ACT2 - local procedure, it goes through an opened binary data  *)
(*     file and copies all the non deleted binary data records into a  *)
(*     temporary binary data file. For each copied binary record, the  *)
(*     associated element in the LARRAY array contains its new record  *)
(*     number (index number) in the temporary data file.               *)
(*   GO_ACT3 - local procedure, it goes through an opened index data   *)
(*     file and copies all the non deleted index data records into a   *)
(*     temporary index data file. For each copied index record, its    *)
(*     index element is assigned with the record number of its asso_   *)
(*     ciated binary data record in the temporay binary data file.     *)
(*     This new record number is obtained from the LARRAY array using  *)
(*     the old record number (index number element) as index.          *)
(*   COPY_XFILE - local procedure, it copies the content of a source   *)
(*     index data file into a destination index data file.             *)
(*   COPY_BFILE - local procedure, it copies the content of a source   *)
(*     binary data file into a destination binary data file.           *)
(*                                                                     *)
(* The following global variables of the GLOBAL unit are used :        *)
(*   I1FNAME - filename of the first index data file.                  *)
(*   I2FNAME - filename of the second index data file.                 *)
```

```
    I3FNAME - filename of the third index data file.                    *)
    B1FNAME - filename of the first binary data file.                   *)
(*  B2FNAME - filename of the second binary data file.                  *)
(*  B3FNAME - filename of the third binary data file.                   *)
(*  IFILE1 - file window variable for the three index data files,       *)
(*           only one at a time.                                        *)
(*  IFILE2 - file window variable for any temporary index data file,   *)
(*           only one at a time.                                        *)
(*  IFILE1 - file window variable for the three binary data files,      *)
(*           only one at a time.                                        *)
(*  IFILE2 - file window variable for any temporary binary data file,  *)
(*           only one at a time.                                        *)
(* ------------------------------------------------------------------- *)

PROCEDURE DO_CLNUP ;

CONST DELETEWORD = -1 ;
      KEEPWORD = 0 ;

VAR IARRAY : ARRAY[0..100] OF INTEGER ;
    IFNX, BFNX, FN : FILENAME ;
    I,J,K,IDX1,IDX2,IDX3 : INTEGER ;
    INPKEY : CHAR ;
    BFREADY : BOOLEAN ;


  (* ------------------------------------------------------------------- *)
  (* This is a local procedure, it goes through an opened index data     *)
  (* file and marks down all the deleted index data records using the    *)
  (* IARRAY integer array. The size of preceding array is the maximum    *)
  (* capacity of index data file, its index starts from zero which is    *)
  (* also the first record number of the index file.                     *)
  (*                                                                     *)
  (* There is a relationship between data records and array elements     *)
  (* of current index data file and IARRAY array. The index element of   *)
  (* an index data record is exactly the same with the index number of   *)
  (* its associated array element in the IARRAY array. When an index     *)
  (* data record is supposed to be deleted, its associated array ele_    *)
  (* ment is assigned -1 value, otherwise the value is 0.                *)
  (* ------------------------------------------------------------------- *)

  PROCEDURE GO_ACT1 ;                          (* inside DO_CLNUP procedure *)

  BEGIN

    IDX1 := 0 ;                               (* counter of deleted records *)

    WHILE ( NOT(EOF(IFILE1)) ) DO              (* loop, processes one index *)
    BEGIN                                      (* data record in each pass  *)

      IF ( NOT(IFILE1^.STATUS) ) THEN             (* a deleted index record *)
      BEGIN

                                               (* get the index element,    *)
        IDX2 := IFILE1^.IDXX ;                 (* uses it to index the array*)
        IARRAY[IDX2] := DELETEWORD ;           (* and assigns a -1 value    *)
```

```
      IDX1 := IDX1 + 1                    (* increment the counter *)

    END ;

    GET(IFILE1)                           (* get next index data record*)

  END

END ;


(* ------------------------------------------------------------------ *)
(* This is a local procedure, it goes through an opened binary data  *)
(* file and copies all the non deleted binary data records into a    *)
(* temporary binary data file.                                       *)
(*                                                                   *)
(* There is a one to one relationship between data records and array *)
(* elements of current binary data file and IARRAY array. The record *)
(* number of a data record is exactly the same with the index number *)
(* of its associated array element in the IARRAY array.              *)
(*                                                                   *)
(* For each binary data record, the record number is used to index   *)
(* an element of IARRAY array. When the preceding array element has  *)
(* 0 value then the current binary data record must be copied, other_*)
 * wise it is a deleted record. For each copied binary record, the   *)
(* associated element in the IARRAY array has a different function   *)
(* now, it contains the new record number (index number) in the tem_ *)
(* porary data file.                                                 *)
(* ------------------------------------------------------------------ *)

PROCEDURE GO_ACT2 ;                       (* inside DO_CLNUP procedure *)

BEGIN

  IDX2 := 0 ;               (* old record number *)
  IDX3 := 0 ;               (* new record number *)

  WHILE ( NOT(EOF(BFILE1)) ) DO          (* loop, each pass processes *)
  BEGIN                                  (* one binary data record    *)

    IF ( IARRAY[IDX2] = KEEPWORD ) THEN            (* a copied record *)
    BEGIN

      IARRAY[IDX2] := IDX3 ;             (* assigned with new record # *)

      IDX3 := IDX3 + 1 ;                 (* increment new record  *)
                                         (* number for next usage *)
      BFILE2^ := BFILE1^ ;          (* assignning binary data record &  *)
      PUT(BFILE2)                   (* put in the temporary binary file *)

    END ;

    GET(BFILE1) ;                (* get next data record of binary data*)
    IDX2 := IDX2 + 1             (* file & increment the record number *)
```

```
    END

  END ;


  (* ------------------------------------------------------------------ *)
  (* This is a local procedure, it goes through an opened index data    *)
  (* file and copies all the non deleted index data records into a      *)
  (* temporary index data file.                                         *)
  (*                                                                    *)
  (* For each copied index data record, its index element is changed    *)
  (* and is assigned with the record number of its associated binary    *)
  (* data record in the temporay binary data file. This new record      *)
  (* number is obtained from the IARRAY array using the old record      *)
  (* number in the index number element as index.                       *)
  (* ------------------------------------------------------------------ *)

  PROCEDURE GO_ACT3 ;                          (* inside DO_CLNUP procedure *)

  BEGIN

    WHILE ( NOT(EOF(IFILE1)) ) DO              (* loop, each pass processes *)
    BEGIN                                      (* one index data record     *)

      IF ( IFILE1^.STATUS ) THEN                    (* a copied data record *)
      BEGIN

        IDX2 := IFILE1^.IDXX ;                 (* get the old record number *)

        IFILE2^.STATUS := TRUE ;                 (* set up data record of   *)
        IFILE2^.STRG := IFILE1^.STRG ;           (* the temporary index file*)
        IFILE2^.IDXX := IARRAY[IDX2] ;           (* using new record number *)
        IFILE2^.UNITT := IFILE1^.UNITT ;

        PUT(IFILE2)                        (* put in the temporary index file *)

      END ;

      GET( IFILE1 )                    (* get next record of the index file *)

    END

  END ;


BEGIN

  (* set up filenames of the temporary index and binary data files *)

  IFNX := CONCAT(VOLUMENAME,'IDX.DATA') ;
  BFNX := CONCAT(VOLUMENAME,'BIN.DATA') ;

  (* loop, each pass only processes one pair of related index and       *)
  (* binary data files.                                                 *)
```

```
FOR I := 1 TO MAXFILE DO
BEGIN

  BFREADY := FALSE ;                    (* the binary file is not ready *)
  FOR J := 0 TO 100 DO                   (* assume all words are copied *)
    IARRAY[J] := KEEPWORD ;

  IF ( I=1 ) THEN                        (* use first index data file *)
    FN := I1FNAME
  ELSE IF ( I=2 ) THEN                   (* use second index data file *)
    FN := I2FNAME
  ELSE                                   (* use third index data file *)
    FN := I3FNAME ;

  (*$I-*)                          (* compiler option, no I/O checking *)

  (* marks the deleted index data records and count them *)

  RESET(IFILE1, FN) ;                    (* open the index data file *)
  IF ( IORESULT = 0 )                    (* only process when the  *)
    THEN GO_ACT1                         (* index file exists,     *)
    ELSE IDX1 := 0 ;                     (* otherwise counter is 0 *)
  CLOSE(IFILE1) ;

  (*$I+*)                          (* compiler option, resume I/O checking *)

  (* only process when the current index data file has index data    *)
  (* records to be deleted, the counter value is non zero.           *)

  IF ( IDX1 > 0 ) THEN
  BEGIN

    IF ( I=1 ) THEN                       (* use first binary data file *)
      FN := B1FNAME
    ELSE IF ( I=2 ) THEN                 (* use second binary data file *)
      FN := B2FNAME
    ELSE IF ( I=3 ) THEN                  (* use third binary data file *)
      FN := B3FNAME ;

(*$I-*)

    RESET(BFILE1, FN) ;                    (* open the binary data file *)
    IF ( IORESULT = 0 )                    (* process only when the *)
      THEN BFREADY := TRUE ;               (* binary file exists    *)
    CLOSE(IFILE1)

(*$I+*)

  END ;

  (* only process when the current index data file has index data    *)
  (* records to be deleted (counter value is non zero) and the       *)
  (* associated binary data file exists.                             *)
```

```
  IF ( (IDX1 > 0) AND (BFREADY) ) THEN
  BEGIN

(*$I-*)

    REWRITE(BFILE2, BFNX) ;          (* create new temporay binary file *)
    RESET(BFILE1, FN ) ;             (* open existing binary data file  *)

    IF ( IORESULT = 0 )                    (* copy all the non    *)
      THEN GO_ACT2 ;                       (* deleted data record *)

    CLOSE(BFILE2, LOCK) ;            (* keep temporary file *)
    CLOSE(BFILE1, PURGE) ;           (* delete the original file *)

(*$I+*)

    IF ( I=1 ) THEN                        (* use first index data file *)
      FN := I1FNAME
    ELSE IF ( I=2 ) THEN                   (* use second file *)
      FN := I2FNAME
    ELSE IF ( I=3 ) THEN                   (* use third file *)
      FN := I3FNAME  ;

(*$I-*)

    REWRITE(IFILE2, IFNX) ;          (* create new temporay index file *)
    RESET(IFILE1, FN) ;              (* open existing index data file  *)

    IF ( IORESULT = 0 )                    (* copy all the non deleted *)
      THEN GO_ACT3 ;                       (* index data records       *)

    CLOSE(IFILE2, LOCK) ;                  (* keep the temporary file  *)
    CLOSE(IFILE1, PURGE) ;                 (* delete the original file *)

    REWRITE(IFILE1, FN) ;            (* recreate original index file *)
    RESET(IFILE2, IFNX) ;                        (* open temporary file *)

    IF ( IORESULT = 0 ) THEN
      COPY_XFILE(IFILE1, IFILE2) ;               (* copy content of file *)

    CLOSE(IFILE1, LOCK) ;            (* keep the original index file *)
    CLOSE(IFILE2, PURGE) ;                   (* delete temporary file *)

(*$I+*)

    IF ( I=1 ) THEN                      (* use first binary data file *)
      FN := B1FNAME
    ELSE IF ( I=2 ) THEN                 (* use second file *)
      FN := B2FNAME
    ELSE IF ( I=3 ) THEN                 (* use third file *)
      FN := B3FNAME ;

(*$I-*)

    REWRITE(BFILE1, FN) ;            (* recreate original binary file *)
```

```
      RESET(BFILE2, BFNX) ;                          (* open temporary file *)

      IF ( IORESULT=0 ) THEN
        COPY_BFILE(BFILE1, BFILE2) ;                 (* copy content of file *)

      CLOSE(BFILE1, LOCK) ;                  (* keep the original index file *)
      CLOSE(BFILE2, PURGE)                          (* delete temporary file *)

  (*$I+*)

    END

  END

END ;
```

```
{ ---------------------------------------------------------------- *)
{ This is a local procedure. It accesses directly a data record of  *)
{ a specified binary data file, the digitized binary data of this   *)
{ data record is replaced by the improved digitized binary data     *)
{ which has been placed in the VOICE global variable.               *)
{                                                                   *)
{ Input :                                                           *)
{   XFILE - parameter, it is the file window variable of a binary   *)
{           data file.                                              *)
{   IDX - parameter, index (record) number of the data record to be *)
{         updated.                                                  *)
{   VOICE - global variable, it has the new digitized binary data.  *)
{                                                                   *)
{ Output :                                                          *)
{   Updated binary data record of the specified binary data file.   *)
{                                                                   *)
{ The following procedures are called :                             *)
{   SEEK - build in file I/O procedure, allow random access to a    *)
{          data record. The file window variable (file pointer) is  *)
{          moved to a specified data record with record number IDX. *)
{   GET - build in file I/O procedure, advance the file window va_  *)
{         riable to the next record and move the content of this    *)
{         record into the file buffer variable.                     *)
{   PUT - build in file I/O procedure, advance the file window va_  *)
{         riable to the next record and puts the content of file    *)
{         buffer variable into this record.                         *)
{                                                                   *)
{ The following global variable of the GLOBAL unit is used :        *)
{   VOICE                                                           *)
{ ---------------------------------------------------------------- *)

PROCEDURE DOIMPROVE ( VAR XFILE:BINARYFILE; IDX:WORDRANGE ) ;

BEGIN

  SEEK(XFILE, IDX) ;                    (* access the binary data *)
  GET(XFILE) ;                          (* record directly        *)

  XFILE^ := VOICE ;                     (* assignning with improved data *)

  SEEK(XFILE, IDX) ;                    (* place into the binary *)
  PUT(XFILE)                            (* data file permanently *)

END ;


{ ---------------------------------------------------------------- *)
{ This procedure improves the sound of words. The digitized binary  *)
{ data records of the words to be improved are replaced with new di_ *)
{ gitized binary data record.                                       *)
{                                                                   *)
{ It is not necessary to input all the words to be improved in alpha_ *)
{ betical ascending order.                                          *)
{                                                                   *)
{ Requirement : call DO6_RESET & DO6_CLOSE procedures by the calling  *)
```

```
   program, respectively, before and after the process of this proce_  *)
   dure.                                                               *)
(*                                                                     *)
(* The processing steps for one word to be improved are :             *)
(* (1) Prompts user to input the word and its total number of unit     *)
(*     sound.                                                          *)
(* (2) Verifies whether this word exists in the dictionary. Only con_  *)
(*     tinues the following two steps when the word exists.            *)
(* (3) prompts user to input the sound of this word and get the digi_  *)
(*     tized binary data record.                                       *)
(* (4) Accesses the associated binary data record from the binary data *)
(*     file and replaces it with the new digitized data for improvement*)
(*                                                                     *)
(* Input :                                                             *)
(*   words and the improved sound of words                             *)
(*                                                                     *)
(* Output :                                                            *)
(*   None                                                              *)
(*                                                                     *)
(* The following procedures are called :                               *)
(*   GET_WORDUNIT - UTILITY unit, prompts user and get a word and its  *)
(*     total number of sound unit interactively.                       *)
(*   WORD_VERIFY - UTILITY unit, verifies whether an input word exists *)
(*     in the dictionary.                                              *)
     DIGITAL - assembly language routine which does the digitizing     *)
       process on the input voice.                                     *)
     DOIMPROVE - local procedure, accesses the binary data record of   *)
(*     the related binary data file and replacing the old digitized    *)
(*     binary data with the improved digitized binary data.            *)
(*   LENGTH - build in string function, get the length of a character  *)
(*     string.                                                         *)
(*                                                                     *)
(* The following global variables of the GLOBAL unit are used :        *)
(*   CHRSALL - display message for all the starting letters of words   *)
(*     in the dictionary.                                              *)
(*   SETCHRS - contains the current set of starting letters.           *)
(*   SET1STCHR - set of all the starting letters of the words in the   *)
(*     dictionary.                                                     *)
(*   VOICE - a buffer which contains the digitized binary data of the  *)
(*     input voice for the word to be improved                         *)
(*   DTEMPO - delay constant of the voice digitizing process.          *)
(*   BFILE1 - file window variable for the first binary data file.     *)
(*   BFILE2 - file window variable for the second binary data file.    *)
(*   BFILE3 - file window variable for the third binary data file.     *)
(*   VOLUMENAME - contains the storage diskette volumename, the default*)
(*     name is STORE:.                                                 *)
(* ------------------------------------------------------------------- *)

PROCEDURE IMPROVE_SOUND ;

   : IDX, UNITVAL:INTEGER ;
     CHANGE, FOUND:BOOLEAN ;
     WRD:WORD ;
     FNDREC:ELEM ;
     CHR:CHAR ;
```

```
.GIN

  CHANGE := TRUE ;
  PAGE(OUTPUT) ;
  WRITELN('-------------------------------------------------------') ;
  WRITELN(' UPDATE THE VOICE DATA OF WORDS') ;
  WRITELN(' PRESS RETURN KEY AFTER SOUND INPUT DEVICE IS ONLINE') ;
  WRITELN('-------------------------------------------------------') ;
  READLN(CHR) ;
  WRITELN ;

  WRITELN('> 1ST CHARACTER OF WORD MUST BE ',CHRSALL) ;

  (* Loop, each pass processes only one word. The end of this loop     *)
  (* occurs when no word is input after the prompt, only RETURN key is *)
  (* pressed.                                                          *)

  SETCHRS := SET1STCHR ;
  REPEAT

    FOUND := FALSE ;
    GET_WORDUNIT(WRD, UNITVAL) ;              (* get a word & its total *)
    WRITELN ;                                 (* number of sound unit   *)

    IF (LENGTH(WRD) > 0) THEN                 (* process only when a *)
    BEGIN                                     (* word has been input *)

      FNDREC.WUNIT := UNITVAL ;
                                                  (* verify whether the *)
      WORD_VERIFY(CHANGE, FOUND, WRD, FNDREC) ;   (* input word exists  *)
                                                  (* in the dictionary  *)

      IF (FOUND) THEN              (* process only when the word exists *)
      BEGIN

        WRITELN('>'PRESS ANY KEY WHEN READY FOR SOUND INPUT') ;

        DIGITAL(VOICE,DTEMPO,UNITVAL) ;          (* digitizing process *)

        READLN ;               (* for the key pressed before sound input *)
        WRITELN('> END OF SOUND INPUT !') ;

        IDX := FNDREC.WIDX ;
        CASE FNDREC.WSET OF                 (* update binary data record *)

          1: DOIMPROVE(BFILE1, IDX) ;          (* first binary data file *)

          2: DOIMPROVE(BFILE2, IDX) ;                (* second data file *)

          3: DOIMPROVE(BFILE3, IDX)                   (* third data file *)

        END   (* CASE *)

      END
```

```
      ELSE
      BEGIN

        WRITE('> THIS WORD: ',WRD,' DOES NOT EXIST IN ') ;
        WRITELN(VOLUMENAME,' DISKETTE')

      END

    END

  UNTIL LENGTH(WRD) = 0 ;

  WRITELN ;
  WRITELN('> END OF SOUND IMPROVEMENT ON WORDS IN  STORE:  DISKETTE') ;
  WRITELN('  ------------------------------------------------------')

END ;

(* Empty main program. No need to define any process in here.          *)

BEGIN
```

```
(* ------------------------------------------------------------------ *)
(* This is the MODULE3 unit of Voice/Digital and Digital/Voice Conver_ *)
(* sion on a Microcomputer project.                                    *)
(*                                                                     *)
(* The Pascal language procedures here do the digitizing of the input  *)
(* voice and the reverse process of speaking (analoging) it back       *)
(* through a connected speaker. The actual processing of digitizing    *)
(* and analoging are accomplished by calling the related assembly      *)
(* language routines. For digitizing, the DIGITAL routine is called.   *)
(* The ANALOG1 and ANALOG2 routines are called for the speaking pro_    *)
(* cess, respectively, through the speaker of Apple II or casette re_  *)
(* corder.                                                             *)
(*                                                                     *)
(* Only the GLOBAL unit is used here.                                  *)
(*                                                                     *)
(* The compiler option $S++ is invoked here, more memory space is      *)
(* available for the compiling process but the speed is decreased.     *)
(* ------------------------------------------------------------------ *)

(*$S++*)                          (* compiler option *)

UNIT MODULE3 ;


INTERFACE

USES GLOBAL ;                    (* declare the unit to be used here *)

  (* declare all the procedures of this module here *)

  PROCEDURE SPEAK ( UNITT : UNITRANGE ) ;
  PROCEDURE SPEAK_WORD ( FNDREC : ELEM ) ;
  PROCEDURE LISTEN ( UNITT : UNITRANGE ) ;


IMPLEMENTATION

  (* declare the assembly language routines to be called here *)

  PROCEDURE DIGITAL( VAR VOICE:SOUND; DTEMPO:INTEGER;
                     DUNIT:UNITRANGE); EXTERNAL ;

  PROCEDURE ANALOG1( VAR VOICE:SOUND; ATEMPO:INTEGER;
                     AUNIT:UNITRANGE); EXTERNAL ;

  PROCEDURE ANALOG2( VAR VOICE:SOUND; ATEMPO:INTEGER;
                     AUNIT:UNITRANGE); EXTERNAL ;
```

```
   ------------------------------------------------------------------ *)
   This procedure calls the assembly language routine DIGITAL to do    *)
(* the actual process of digitizing (listening) the input voice.       *)
(*                                                                     *)
(* Input :                                                             *)
(*   UNITT - parameter, total number of sound units of the input voice.*)
(*   DTEMPO - global variable, delay constant of the digitizing pro_   *)
(*            cess where the value was assigned in the calling program *)
(*                                                                     *)
(* Output :                                                            *)
(*   VOICE - global variable, it is a buffer which contains the        *)
(*           digitized binary data of the input voice                  *)
(*                                                                     *)
(* The following procedure is called :                                 *)
(*   DIGITAL - external procedure, listening the input voice.          *)
(*                                                                     *)
(* The following global variables of the GLOBAL unit are used :        *)
(*   VOICE                                                             *)
(*   DTEMPO                                                            *)
(* ------------------------------------------------------------------ *)

PROCEDURE LISTEN ;

BEGIN

   IGITAL( VOICE, DTEMPO, UNITT )

END ;
```

```
   ------------------------------------------------------------------ *)
   This procedure calls the assembly language routines ANALOG1 or      *)
(* ANALOG2 to do the actual process of analoging (speaking) the digi_  *)
(* tized binary sound data.                                            *)
(*                                                                     *)
(* Input :                                                             *)
(*   UNITT - parameter, total number of sound units of the digitized   *)
(*           binary sound data.                                        *)
(*   ATEMPO - global variable, delay constant of the analoging process *)
(*            where the value was assigned in the calling program.     *)
(*   VOICE - global variable, it is the buffer which contains the      *)
(*           digitized binary data.                                    *)
(*   SPKER - global variable, it decides whether the speaker of Apple  *)
(*           II or casette recorder is used in the voice output, the   *)
(*           value was assigned in the calling program.                *)
(*                                                                     *)
(* Output :                                                            *)
(*   Voice at the chosen speaker                                       *)
(*                                                                     *)
(* The following procedures are called :                               *)
(*   ANALOG1 - external procedure, speaking on the Apple II speaker.   *)
(*   ANALOG2 - external procedure, speaking on the attached casette    *)
(*             recorder speaker.                                       *)
(*                                                                     *)
   The following global variables of the GLOBAL unit are used :       *)
     VOICE, ATEMPO, SPKER.                                            *)
   ------------------------------------------------------------------ *)

PROCEDURE SPEAK ;

BEGIN

   IF ( SPKER > CASETTE )
     THEN ANALOG1 ( VOICE, ATEMPO, UNITT )          (* Apple II *)
     ELSE ANALOG2 ( VOICE, ATEMPO, UNITT )          (* casette recorder *)

END ;
```

```
   ------------------------------------------------------------------ *)
   This procedure calls the assembly language routines ANALOG1 or      *)
(* ANALOG2 to do the actual process of speaking a specific word which  *)
(* exists in the dictionary. The digitized binary data of this word is *)
(* obtained from the related binary data file by using the information *)
(* in the parameter.                                                   *)
(*                                                                     *)
(* Input :                                                             *)
(*   FNDREC - parameter, information to access the related binary data *)
(*            file in order to obtained the digitized binary data of   *)
(*            the related word to be spoken                            *)
(*   ATEMPO - global variable, delay constant of the analoging process *)
(*            where the value was assigned in the calling program      *)
(*   SPKER - global variable, it decides whether the speaker of Apple  *)
(*           II or casette recorder is used in the voice output, the   *)
(*           value was assigned in the calling program                 *)
(*                                                                     *)
(* Output :                                                            *)
(*   Voice at the chosen speaker                                       *)
(*                                                                     *)
(* The following procedures are called :                               *)
(*   ANALOG1 - external procedure, speaking on the Apple II speaker.   *)
(*   ANALOG2 - external procedure, speaking on the speaker of the      *)
(*             attached casette recorder.                              *)
     SEEK - build in file I/O procedure, allow ramdow access of a spe_ *)
            cific record with index (record) number BIDX. The file     *)
            window variable (file pointer) is moved to a specified re_ *)
(*          cord in the data file.                                     *)
(*   GET - build in file I/O procedure, advance the file window varia_ *)
(*         ble to the next record and move the content of this record  *)
(*         into the file buffer variable.                              *)
(*                                                                     *)
(* The following global variables of the GLOBAL unit are used :        *)
(*   SPKER, ATEMPO                                                     *)
(*   VOICE - buffer contains the binary sound data of the word.        *)
(*   BFILE1 - file window variable of the first binary data file.      *)
(*   BFILE2 - file window variable of the second binary data file.     *)
(*   BFILE3 - file window variable of the third binary data file.      *)
(* ------------------------------------------------------------------ *)

PROCEDURE SPEAK_WORD ;

VAR BIDX, BUNIT, BSET : INTEGER ;

BEGIN

  BIDX := FNDREC.WIDX ;        (* index in the related binary data file.*)
  BSET := FNDREC.WSET ;             (* set # of words in the dictionary.*)
  BUNIT := FNDREC.WUNIT ;                 (* total number of unit sound.*)

  CASE BSET OF
                                    (* member of the 1st set of words, *)
    1:BEGIN                         (* move the file pointer directly  *)
        SEEK(BFILE1, BIDX) ;        (* to the desired record.          *)
        GET(BFILE1) ;             (* obtain the record of digitized data.*)
```

```
      VOICE := BFILE1^            (* assign data to the global buffer.*)
    END ;

  2:BEGIN                         (* it is a member of the 2nd set *)
      SEEK(BFILE2, BIDX) ;        (* of words in the dictionary    *)
      GET(BFILE2) ;
      VOICE := BFILE2^
    END ;

  3:BEGIN                         (* it is a member of the 3rd set *)
      SEEK(BFILE3, BIDX) ;        (* of words in the dictionary    *)
      GET(BFILE3) ;
      VOICE := BFILE3^
    END

 END ; (*CASE*)

 IF ( SPKER > CASETTE )                          (* speak the word *)
   THEN ANALOG1( VOICE, ATEMPO, BUNIT)           (* use Apple II speaker *)
   ELSE ANALOG2( VOICE, ATEMPO, BUNIT)           (* use casette recorder *)

END ;

(* The main program is empty, no need to define any process in here.   *)

BEGIN


END.
```

There are 3 Assembly Language routines in this file, they are:
DIGITAL, ANALOG and ANALOG2.

The POP and PUSH macros are called by all 3 routines.

In this listing, the dollar ($) sign is used instead of 'AT' sign for the indirect index addressing mode.

```
;
; This POP macro pops a 2 bytes address of a returned parameter
; from the stack.
;
;  ---------------------
    .MACRO POP
;  ---------------------
;
        PLA                         ; get lower byte address from stack
        STA     %1                  ; assign to lower byte of a variable
        PLA                         ; get higher byte address from stack
        STA     %1+1                ; assign to higher byte of a variable
        .ENDM
;
; This PUSH macro pushs a 2 bytes address of a returned parameter
; from the stack.
;
;  ----------------------
    .MACRO PUSH
;  ----------------------
;
        LDA     %1+1                ; get the higher byte address
        PHA                         ; push it into the stack
        LDA     %1                  ; get the lower byte address
        PHA                         ; push it into the stack
        .ENDM
```

Obtain the digitized binary data of input sound and place it into the VOICE array buffer parameter which is passed by the Pascal calling routine.
The VOICE array has capacity of 1280 (X'04FF') bytes, depending on the value of parameter IDX (1, 2, 3, 4 or 5), respectively, only 255 (X'00FE') bytes, 510 (X'01FD') bytes, 765 (X'02FC') bytes, 1020 (X'03FB') bytes or 1275 bytes of this array will be filled with binary sound data.

```
;  ------------------------
    .PROC DIGITAL,3
;  ------------------------
```

```
;
; Zero page variables definition
;
RET     .EQU    0               ; store return address
ADRL    .EQU    2               ; store lower & upper bytes of
ADRH    .EQU    3               ;  VOICE array starting address
INIL    .EQU    4               ; store lower & upper bytes of
INIH    .EQU    5               ;  VOICE array starting address
CNTL    .EQU    6               ; lower & upper bytes of a counter,
CNTH    .EQU    7               ;  store the size of VOICE to be used
ADVAL   .EQU    8               ; 0 or 1, state of the A/D sampling
ENDL    .EQU    9               ; last byte of the subset of VOICE
ENDH    .EQU    0A              ;  array used, these are the address
IDX     .EQU    0B              ; store the unit of sound
TEMPO   .EQU    0C              ; store the delay loop constant
TMP     .EQU    0D              ; temporary storage
WVAL    .EQU    0E              ; temporary var for delay cycles
KEYB0   .EQU    0C000           ; loc to get keyboard input
KEYB1   .EQU    0C010           ; loc to clear keyboard input
SPKER   .EQU    0C030           ; loc to toggle Apple speaker
INPJK   .EQU    0C060           ; loc to sample the digital input

;
; Save the return address and get the all the parameters
;
        POP     RET             ; get the return address in RET
        PLA                     ; get the value of unit sound
        STA     IDX             ;  parameter and store it in IDX
        PLA                     ; ignore the higher byte
        PLA                     ; get the value of delay constant
        STA     TEMPO           ;  parameter and store it in TEMPO
        PLA                     ; ignore the higher byte
        PLA                     ; get the starting addr of VOICE array
        STA     INIL            ;  parameter, lower byte, store it in
        STA     ADRL            ;  INIL & ADRL
        PLA                     ; similar but higher byte, store it
        STA     INIH            ;  in INIH & ADRH
        STA     ADRH

;
; From the value of sound unit (IDX), get one of the 5 sizes of
; VOICE array to be utilized and place its value in lower and
; higher bytes of counter, respectively, CNTL and CNTH.
;
CNTER   LDA     #0FE            ; LSB of any size always ends with
        STA     CNTL            ;  0FA, 0FB, 0FC, 0FD or 0FE
        LDA     #01
        CMP     SIZE            ; 1 unit size ?
        BNE     ITWO
        LDA     #00
        STA     CNTH            ; Yes, 255 = X'00FE' bytes
        JMP     ZEROY
ITWO    LDA     #02
        CMP     SIZE            ; 2 units size ?
```

```
        BNE     ITHRE
        DEC     CNTL
        LDA     #01
        STA     CNTH            ; Yes, 510 = X'01FD' bytes
        JMP     ZEROY
ITHRE   LDA     #03
        CMP     SIZE            ; 3 units size ?
        BNE     IFOUR
        DEC     CNTL
        LDA     #02             ; Yes, 765 = X'02FC' bytes
        STA     CNTH
        JMP     ZEROY
IFOUR   LDA     #04
        CMP     SIZE            ; 4 units size ?
        BNE     IFIVE
        DEC     CNTL
        LDA     #03             ; Yes, 1020 = X'03FB' bytes
        STA     CNTH
        JMP     ZEROY
IFIVE   DEC     CNTL
        LDA     #04 .           ; 5 units size, thus
        STA     CNTH            ; 1275 = X'04FA' bytes
;
; For the subset of VOICE array to be used, initialize each element
; with 0 value. The variables ADRL & ADRH have the starting address
; of the array originally. After the initialization of each element,
; the value of ADRL is incremented by 1, and the value of CNTL is
; decremented by 1. The task of initialization is ended when the
; values in both CNTL & CNTH are zero.
; After the increment of ADRL, when it is overflow (becomes 0 value),
; the higher byte ADRH is incremented by 1.
; After the decrement of CNTL, when it is underflow (becomes 0 value),
; the higher byte CNTH is decremented by 1 and X'FF' value is
; assigned to CNTL.
;
ZEROY   LDY     #00
IZERO   LDA     #00
        STA     $ADRL,Y         ; init element pointed by ADRL & ADRH
        INC     ADRL            ; increment lower byte
        BNE     COUNT
        INC     ADRH            ; overflow, incre higher byte
COUNT   LDA     #00
        CMP     CNTL            ; check lower byte of counter, if it
        BEQ     HZERO           ; is zero, check higher byte
        DEC     CNTL            ; decrement is nonzero
        JMP     AGAIN
HZERO   CMP     CNTH            ; check higher byte of counter
        BEQ     STRT
        DEC     CNTH            ; decrement if nonzero and assigns
        LDA     #0FF            ; X'FF' to lower byte of counter
        STA     CNTL
AGAIN   JMP     IZERO
;
```

```
; Initialize state of A/D sampling with X'FF', which is none of the
; sampling state values of X'00' or X'01'.
; For the subset of VOICE array, stores the address of the last byte
; into ENDL & ENDH.
; Makes ADRL & ADRH points to the start of VOICE array again.
;
STRT    LDA     #0FF            ; init state of A/D sampling
        STA     ADVAL
        LDA     ADRL            ; ADRL & ADRH have the address of the
        STA     ENDL            ;  last byte used, assign to ENDL &
        LDA     ADRH            ;  ENDH respectively.
        STA     ENDH
        LDA     INIL            ; get the starting address of VOICE
        STA     ADRL            ;  array from INIL & INIH and assign
        LDA     INIH            ;  back to ADRL & ADRH.
        STA     ADRH
;
; Continue replays the digitized A/D binary data on the Apple II
; speaker without analyzing it until a key on the keyboard has
; been pressed to start the process.
;
        STA     KEYB1           ; clear keyboard input
LEADIN  LDA     INPJK           ; get digital data from input port
        AND     #80             ; leftmost bit is the state of sampling
        CMP     ADVAL
        BEQ     WAST1
        STA     ADVAL           ; toggle speaker if current &
        STA     SPKER           ;  previous A/D data are different,
        JMP     GKEY            ;  store the current data.
;
WAST1   STA     WVAL            ; delay cycles if both states of
        STA     WVAL            ;  sampling are the same
        STA     WVAL
;
GKEY    LDA     KEYB0           ; get keyboard input, if any, the
        AND     #80             ;  leftmost bit is nonzero
        BEQ     LEADIN
;
; Delay loop.
; Slow down the sampling rate with a delay constant, the
; default value is 1.
;
        LDX     #00
        LDY     TEMPO
LDELAY  DEY
        BNE     LDELAY
;
; Analyze the digitized input data.
; Toggle the speaker if current and previous A/D sampling state
; are different, otherwise, increment the counter in register X.
;
```

```
GETAD   LDA   INPJK          ; get the digitized data, the state
        AND   #80            ;  of A/D sampling is in the left
        CMP   ADVAL          ;  most bit.
        BNE   WAST2
;
        INX                  ; increment counter X
        CPX   #7F            ; store the counter value if its
        BNE   WAST3          ;  capacity X'7F' has been reached
        BEQ   WAST4
        JMP   BSOUND         ; go to speaker toggling
;
WAST2   STA   WVAL           ; delay cycles
        STA   WVAL
        NOP
        JMP   BSOUND         ; go to speaker toggling
;
WAST3   INC   WVAL           ; delay cycles
        INC   WVAL
        INC   WVAL
        INC   WVAL
        INC   WVAL
        INC   WVAL
        INC   WVAL
        INC   WVAL
        INC   WVAL
        INC   WVAL
        INC   WVAL
        INC   WVAL
        INC   WVAL
        INC   WVAL
        INC   WVAL
        STA   WVAL
        STA   WVAL
        JMP   LDELAY-2       ; back to delay loop before next
;                              sampling
;
BSOUND  STA   SPKER          ; toggle speaker
        STA   ADVAL          ; store the current A/D state value
        JMP   BSTORE         ; go to store the counter value of
;                              previous A/D state value
;
WAST4   STA   WVAL           ; delay cycles
        STA   WVAL
        STA   WVAL

;
; Store the value of counter X to a byte pointed by ADRL & ADRH.
; Afterward, increment ADRL by 1, if result is overflow then
; increment ADRH by 1. The new address in ADRL & ADRH now point
; to the next vacant byte.
;
BSTORE  STX   TMP
        LDA   TMP            ; place the A/D state value into
        EOR   ADVAL          ;  the leftmost bit of X counter
        STA   $ADRL,Y        ; assign to current byte
```

```
            INC     ADRL            ; get addr of next vacant byte
            BNE     WAST5
            INC     ADRH
            NOP
            JMP     NOVFL
;
WAST5       STA     WVAL            ; delay cycles
            STA     WVAL
            STA     WVAL
;
; Check whether the last byte address of the array subset has
; been reached. Current byte address is in ADRL & ADRH, the last
; byte address is in ENDL & ENDH.
; Ending when (ADRL=ENDL) and (ADRH=ENDH).
;
NOVFL       LDA     ADRL            ; compare ADRL with ENDL
            CMP     ENDL            ; not equal, no need for further
            BNE     WAST6           ;  comparison
            LDA     ADRH
            CMP     ENDH            ; compare ADRH with ENDH
            BEQ     FINI            ; end of routine if equal
            JMP     NEXTL
;
WAST6       INC     WVAL            ; delay cycles
            INC     WVAL
            INC     WVAL
            INC     WVAL
            NOP
            NOP
;
NEXTL       LDX     #00             ; init counter of reg X
            NOP
            NOP
            JMP     LDELAY-2        ; back to delay loop before
;                                     next sampling
;
FINI        LDA     INIH            ; push starting address of the
            PHA                     ;  VOICE array buffer into the
            LDA     INIL            ;  stack, return parameter.
            PHA
            PUSH    RET             ; push return address into stack
            RTS
```

Sound is reproduced at the speaker of Apple II using the binary sound data in array parameter VOICE. The size of this array is 1275 (X'04FA') bytes, but only a subset of this array contains the binary sound data.
Depending on the value of parameter IDX (1, 2, 3, 4 or 5), respectively, only 255 (X'00FE') bytes, 510 (X'01FD') bytes, 765 (X'02FC') bytes, 1020 (X'03FB') bytes or 1275 bytes of this array contain binary sound data.

```
;  ------------------------
     .PROC ANALOG1,3
;  ------------------------
;
; Zero page variables definition
;
RET       .EQU   0              ; return address storage
ADRL      .EQU   2              ; storage of VOICE array
ADRH      .EQU   3              ;  parameter's starting address
INIL      .EQU   4              ;   ADRL, INIL - LSB
INIH      .EQU   5              ;    ADRH, INIH - MSB
CNTL      .EQU   6              ; counter, # of bytes of VOICE array
CNTH      .EQU   7              ;  which contain binary sound data
ADVAL     .EQU   8              ; current A/D state value
ENDL      .EQU   9              ; address of the last byte which
ENDH      .EQU   0A             ;  has binary sound data
SIZE      .EQU   0B             ; # of sound units
TEMPO     .EQU   0C             ; delay loop constant
WVAL      .EQU   0D             ; temporary var for delay cycles
OUTL      .EQU   0C030          ; loc to toggle Apple II speaker
;
; Save the return address and get the all the parameters
;
          POP    RET            ; get the return address in RET
          PLA                   ; get the value of unit sound
          STA    SIZE           ;  parameter and store it in IDX
          PLA                   ; ignore the higher byte
          PLA                   ; get the value of delay constant
          STA    TEMPO          ;  parameter and store it in TEMPO
          PLA                   ; ignore the higher byte
          PLA                   ; get the starting addr of VOICE array
          STA    INIL           ;  parameter, lower byte, store it in
          STA    ADRL           ;  INIL & ADRL
          PLA                   ; similar but higher byte, store it
          STA    INIH           ;  in INIH & ADRH
          STA    ADRH           ;
;
; From the value of unit sound parameter (SIZE), determines
; the total number of bytes in VOICE array parameter which
; have been filled with binary sound data.
; The number of bytes are stored in: CNTL - LSB  & CNTH - MSB
```

```
;
CNTER   LDA   #0FE          ; LSB of any size always ends with
        STA   CNTL          ;  0FA, 0FB, 0FC, 0FD or 0FE
        LDA   #01
        CMP   SIZE          ; 1 unit size ?
        BNE   ITWO
        LDA   #00
        STA   CNTH          ; Yes, 255 = X'00FE' bytes
        JMP   GPOS
ITWO    LDA   #02
        CMP   SIZE          ; 2 units size ?
        BNE   ITHRE
        DEC   CNTL
        LDA   #01
        STA   CNTH          ; Yes, 510 = X'01FD' bytes
        JMP   GPOS
ITHRE   LDA   #03
        CMP   SIZE          ; 3 units size ?
        BNE   IFOUR
        DEC   CNTL
        LDA   #02           ; Yes, 765 = X'02FC' bytes
        STA   CNTH
        JMP   GPOS
IFOUR   LDA   #04
        CMP   SIZE          ; 4 units size ?
        BNE   IFIVE
        DEC   CNTL
        LDA   #03           ; Yes, 1020 = X'03FB' bytes
        STA   CNTH
        JMP   GPOS
IFIVE   DEC   CNTL
        LDA   #04           ; 5 units size, thus
        STA   CNTH          ;  1275 = X'04FA' bytes
;
; Find the address of the last byte in VOICE array parameter which
; has binary sound data by decrementing CNTL or CNTH, also increment
; ADRL or ADRH. Ending of searching if both content of CNTL & CNTH
; is zero.
; The value of ADRL & ADRH now is the required address, save it at
: ENDL - LSB and ENDH - MSB. Assign back the starting address of
; VOICE parameter array to ADRL & ADRH from INITL & INIH respectively.
;
GPOS    LDA   #0FF          ; initialize A/D state value, use any
        STA   ADVAL         ;  value except X'00' or X'01'
DOPOS   INC   ADRL          ; increment lower byte ADRL
        BNE   COUNT         ;
        INC   ADRH          ; increment higher byte ADRH if ARDL
COUNT   LDA   #00           ;  overflows
        CMP   CNTL          ;
        BEQ   HZERO         ; decrement lower byte CNTL if
        DEC   CNTL          ;  it is nonzero
        JMP   AGAIN         ;
HZERO   CMP   CNTH          ; CNTL=0 , is CNTH=0 ?
```

```
            BEQ     STRT            ; yes, end of searching
            DEC     CNTH            ; no, decrement CNTH &
            LDA     #0FF            ;  add X'0FF' to CNTL
            STA     CNTL            ;
AGAIN       JMP     DOPOS           ;
;                                   ;
STRT        LDA     ADRL            ; address of last byte
            STA     ENDL            ;  LSB - store in ENDL
            LDA     ADRH            ;  MSB - store in ENDH
            STA     ENDH            ;
            LDA     INIL            ;
            STA     ADRL            ; ADRL & ADRH contain the starting
            LDA     INIH            ;  address of VOICE array again
            STA     ADRH            ;
            LDY     #00             ;

;
; Analyze the byte value pointed by ADRL & ADRH, get the A/D state
; value and the counter value.
; Leftmost bit is A/D state value, either 0 or 1.
; 2nd -> 8th bits is counter value.
;
VALGET      LDA     $ADRL,Y         ; get the byte value
            AND     #7F             ; get the counter value and
            TAX                     ;  store it at register X
            LDA     ADRL,Y          ;
            AND     #80             ; get A/D state value
            CMP     ADVAL           ; compare with previous A/D state value
            BEQ     WASTE1          ; same, do delay loop
            STA     ADVAL
            STA     OUTL            ; toggle speaker if different
            JMP     DECINX
WASTE1      STA     WVAL
            STA     WVAL
            STA     WVAL

;
; Delay loop, default value of delay loop constant is 4.
;
DECINX      LDY     TEMPO           ; get delay loop constant
            DEY                     ;
            BNE     DECINX+2        ;
            DEX                     ; decrement counter value
            BMI     NLOC            ; initial counter value is 0
            BEQ     NLOC            ; go to get next byte value
;                                   ;
WASTE2      NOP                     ; delay cycles
            INC     WVAL            ;
            INC     WVAL            ;
            INC     WVAL            ;
            INC     WVAL            ;
            INC     WVAL            ;
            INC     WVAL            ;
            INC     WVAL            ;
```

```
            INC     WVAL            ;
            INC     WVAL            ;
            INC     WVAL            ;
            INC     WVAL            ;
            INC     WVAL            ;
            INC     WVAL            ;
            INC     WVAL            ;
            JMP     DECINX          ;
;
; Get address of the next byte by incrementing ADRL or ADRH by 1.
; Last byte of binary sound data has been reached if :
;    (ADRL=ENDL) and (ADRH=ENDH)
;
NLOC        INC     ADRL            ; increment lower byte
            BNE     WASTE3
            INC     ADRH            ; increment higher byte if lower
            NOP                     ;  byte overflows
            JMP     NFULL
;
WASTE3      STA     WVAL            ; delay cycles
            STA     WVAL
            STA     WVAL
;
NFULL       LDA     ADRL
            CMP     ENDL            ; is
            BNE     WASTE4          ; no, get next byte value
            LDA     ADRH            ; yes, is ADRH = ENDH ?
            CMP     ENDH            ;
            BEQ     FINI            ; yes, end of binary sound data
            JMP     VALGET          ;
;
WASTE4      STA     WVAL            ; delay cycles
            STA     WVAL
            STA     WVAL
            JMP     VALGET          ; no, go to get next byte value
;
FINI        PUSH    INIL            ; push starting byte address of the
            PUSH    RET             ;  VOICE array parameter and return
            RTS                     ;  address in the calling routine
;                                   ;  into the stack
```

Sound is reproduced at the speaker of cassette recorder using the binary sound data in array parameter VOICE. The size of this array is 1275 (X'04FA') bytes, but only a subset of this array contains the binary sound data.
Depending on the value of parameter IDX (1, 2, 3, 4 or 5), respectively, only 255 (X'00FE') bytes, 510 (X'01FD') bytes, 765 (X'02FC') bytes, 1020 (X'03FB') bytes or 1275 bytes of this array contain binary sound data.

```
;   ----------------------
      .PROC ANALOG22,3
;   ----------------------
;
; Zero page variables definition
;
RET       .EQU   0              ; return address storage
ADRL      .EQU   2              ; storage of VOICE array
ADRH      .EQU   3              ;  parameter's starting address
INIL      .EQU   4              ;   ADRL, INIL - LSB
INIH      .EQU   5              ;    ADRH, INIH - MSB
CNTL      .EQU   6              ; counter, # of bytes of VOICE array
CNTH      .EQU   7              ;  which contain binary sound data
ADVAL     .EQU   8              ; current A/D state value
ENDL      .EQU   9              ; address of the last byte which
ENDH      .EQU   0A             ;  has binary sound data
SIZE      .EQU   0B             ; # of sound units
TEMPO     .EQU   0C             ; delay loop constant
WVAL      .EQU   0D             ; temporary var for delay cycles
OUTL      .EQU   0C020          ; loc to toggle cassette recorder speaker
;
; Save the return address and get the all the parameters
;
          POP    RET            ; get the return address in RET
          PLA                   ; get the value of unit sound
          STA    SIZE           ;  parameter and store it in IDX
          PLA                   ; ignore the higher byte
          PLA                   ; get the value of delay constant
          STA    TEMPO          ;  parameter and store it in TEMPO
          PLA                   ; ignore the higher byte
          PLA                   ; get the starting addr of VOICE array
          STA    INIL           ;  parameter, lower byte, store it in
          STA    ADRL           ;  INIL & ADRL
          PLA                   ; similar but higher byte, store it
          STA    INIH           ;  in INIH & ADRH
          STA    ADRH           ;
;
; From the value of unit sound parameter (SIZE), determines
; the total number of bytes in VOICE array parameter which
; have been filled with binary sound data.
; The number of bytes are stored in: CNTL - LSB  & CNTH - MSB
;
CNTER     LDA    #0FE           ; LSB of any size always ends with
```

```
        STA     CNTL            ; 0FA, 0FB, 0FC, 0FD or 0FE
        LDA     #01
        CMP     SIZE            ; 1 unit size ?
        BNE     ITWO
        LDA     #00
        STA     CNTH            ; Yes, 255 = X'00FE' bytes
        JMP     GPOS
ITWO    LDA     #02
        CMP     SIZE            ; 2 units size ?
        BNE     ITHRE
        DEC     CNTL
        LDA     #01
        STA     CNTH            ; Yes, 510 = X'01FD' bytes
        JMP     GPOS
ITHRE   LDA     #03
        CMP     SIZE            ; 3 units size ?
        BNE     IFOUR
        DEC     CNTL
        LDA     #02             ; Yes, 765 = X'02FC' bytes
        STA     CNTH
        JMP     GPOS
IFOUR   LDA     #04
        CMP     SIZE            ; 4 units size ?
        BNE     IFIVE
        DEC     CNTL
        LDA     #03             ; Yes, 1020 = X'03FB' bytes
        STA     CNTH
        JMP     GPOS
IFIVE   DEC     CNTL
        LDA     #04             ; 5 units size, thus
        STA     CNTH            ;  1275 = X'04FA' bytes
;
; Find the address of the last byte in VOICE array parameter which
; has binary sound data by decrementing CNTL or CNTH, also increment
; ADRL or ADRH. Ending of searching if both content of CNTL & CNTH
; is zero.
; The value of ADRL & ADRH now is the required address, save it at
; ENDL - LSB and ENDH - MSB. Assign back the starting address of
; VOICE parameter array to ADRL & ADRH from INITL & INIH respectively.
;
GPOS    LDA     #0FF            ; initialize A/D state value, use any
        STA     ADVAL           ;  value except X'00' or X'01'
DOPOS   INC     ADRL            ; increment lower byte ADRL
        BNE     COUNT           ;
        INC     ADRH            ; increment higher byte ADRH if ARDL
COUNT   LDA     #00             ;  overflows
        CMP     CNTL            ;
        BEQ     HZERO           ; decrement lower byte CNTL if
        DEC     CNTL            ;  it is nonzero
        JMP     AGAIN           ;
HZERO   CMP     CNTH            ; CNTL=0 , is CNTH=0 ?
        BEQ     STRT            ; yes, end of searching
        DEC     CNTH            ; no, decrement CNTH &
```

```
            LDA     #0FF                ; add X'0FF' to CNTL
            STA     CNTL                ;
AGAIN       JMP     DOPOS               ;
;                                       ;
STRT        LDA     ADRL                ; address of last byte
            STA     ENDL                ; LSB - store in ENDL
            LDA     ADRH                ; MSB - store in ENDH
            STA     ENDH                ;
            LDA     INIL                ;
            STA     ADRL                ; ADRL & ADRH contain the starting
            LDA     INIH                ; address of VOICE array again
            STA     ADRH                ;
            LDY     #00                 ;

;
; Analyze the byte value pointed by ADRL & ADRH, get the A/D state
; value and the counter value.
; Leftmost bit is A/D state value, either 0 or 1.
; 2nd -> 8th bits is counter value.
;
VALGET      LDA     $ADRL,Y             ; get the byte value
            AND     #7F                 ; get the counter value and
            TAX                         ; store it at register X
            LDA     ADRL,Y              ;
            AND     #80                 ; get A/D state value
            CMP     ADVAL               ; compare with previous A/D state value
            BEQ     WASTE1              ; same, do delay loop
            STA     ADVAL
            STA     OUTL                ; toggle speaker if different
            JMP     DECINX
WASTE1      STA     WVAL
            STA     WVAL
            STA     WVAL

;
; Delay loop, default value of delay loop constant is 4.
;
DECINX      LDY     TEMPO               ; get delay loop constant
            DEY                         ;
            BNE     DECINX+2            ;
            DEX                         ; decrement counter value
            BMI     NLOC                ; initial counter value is 0
            BEQ     NLOC                ; go to get next byte value
;                                       ;
WASTE2      NOP                         ; delay cycles
            INC     WVAL                ;
            INC     WVAL                ;
            INC     WVAL                ;
            INC     WVAL                ;
            INC     WVAL                ;
            INC     WVAL                ;
            INC     WVAL                ;
            INC     WVAL                ;
            INC     WVAL                ;
```

```
            INC     WVAL            ;
            INC     WVAL            ;
            INC     WVAL            ;
            INC     WVAL            ;
            INC     WVAL            ;
            JMP     DECINX          ;
;
; Get address of the next byte by incrementing ADRL or ADRH by 1.
; Last byte of binary sound data has been reached if :
;    (ADRL=ENDL) and (ADRH=ENDH)
;
NLOC        INC     ADRL            ; increment lower byte
            BNE     WASTE3
            INC     ADRH            ; increment higher byte if lower
            NOP                     ;  byte overflows
            JMP     NFULL
;
WASTE3      STA     WVAL            ; delay cycles
            STA     WVAL
            STA     WVAL
;
NFULL       LDA     ADRL
            CMP     ENDL            ; is
            BNE     WASTE4          ; no, get next byte value
            LDA     ADRH            ; yes, is ADRH = ENDH ?
            CMP     ENDH            ;
            BEQ     FINI            ; yes, end of binary sound data
            JMP     VALGET          ;
;
WASTE4      STA     WVAL            ; delay cycles
            STA     WVAL
            STA     WVAL
            JMP     VALGET          ; no, go to get next byte value
;
FINI        PUSH    INIL            ; push starting byte address of the
            PUSH    RET             ;  VOICE array parameter and return
            RTS                     ;  address in the calling routine
;                                   ;  into the stack
```

CHAPTER 4

SPECIAL FEATURE OF UCSD PASCAL

4.1. EXTERNAL COMPILATION UNITS

The UCSD Pascal System supports a facility for integrating externally compiled and assembled routines and data structures. Use of separately compiled or assembled structures allows the user to create files of frequently used routines. The user does not have to insert this new structure into each program which calls the structure and then compile the combined text; rather, the LINKER of the Pascal System copies the structure's code directly into the host program's code file.

After a structure is compiled or assembled, the user can use the LINKER explicitly to integrate that structure into any program which correctly calls the structure. Alternatively, the user can add the new structure to a library, using the LIBRARIAN of the Pascal System. When user later RUNs any program which calls a structure in the library, the LINKER will automatically find and link in that structure. Separate compilation or assembly is supported in these areas:

(1) Between portions of programs written in Pascal language.

(2) Between Assembly language routines and Pascal language host program.

(3) Between Assembly language routines.

4.2. PASCAL TO PASCAL LINKAGE : UNIT

An UNIT is a special group of interdependent procedures, functions, and associated data structures which perform a specialized task. The unit is placed in the System Library, and whenever this task is needed within a program, the program indicates that it USES the unit. For example, to use any of the procedures in the GLOBAL UNIT, the host program would simply start as follows:

```
PROGRAM DEMO_GLOBAL (INPUT, OUTPUT) ;
USES GLOBAL ;
```

Each UNIT consists of two parts:

(1) INTERFACE

(2) IMPLEMENTATION

The INTERFACE part immediately follows the UNIT's name line. It declares constant, types, variables, procedures and functions that are PUBLIC. These items can be used by the host program just as if they appear in the explicit declarations at the top of the host program itself. The INTERFACE portion is the only part of the UNIT that is **visible** from the outside, it specifies how a host program can communicate with the UNIT. The actual workings of the UNIT can be changed at any time, but the UNIT will appear to be the same as long as the INTERFACE portion is unchanged.

The IMPLEMENTATION part immediately follows the last declaration in the INTERFACE part. It begins by declaring those constants, types, variables, procedures and functions that are PRIVATE. Items which are declared in the IMPLEMENTATION part are not available to the host program and are used only by the UNIT itself. The IMPLEMENTATION part defines how the UNIT will accomplish its task. This part gives the details of the various procedures and functions declared in the INTERFACE part, and also the private procedures and functions declared in the IMPLEMENTATION part.

At the end of the IMPLEMENTATION part, following the last function and procedure, there is a **main program** portion. This program runs automatically when the host program begins, before the host program is run. It allows user to initialize the system and the host program. The declaration of routine headings in the INTERFACE part is similar to forward declarations; therefore, when the corresponding bodies are defined in the IMPLEMENTATION part, formal parameter specifications are not repeated.

The properly completed UNIT would then be compiled. Any external Assembly language procedures or functions would then be linked in, using the LINKER. Finally, the unit would be installed in a library, SYSTEM.LIBRARY for example, using the LIBRARIAN utility. Once in the library, the unit could then be used by any Pascal host program.

A host program must indicate the UNITs that it USES before the LABEL or CONSTANT declaration part of the program. At the occurance of an USES statement, the Compiler references the INTERFACE

part of the unit as though it is part of the host text itself. Therefore, all constants, types, variables, functions and procedures publicly defined in the unit are global. Name conflicts may arise if the user defines an identifier which has already been publicly declared by the unit. Procedures and functions may not USES units locally.

4.3. PASCAL TO ASSEMBLY LANGUAGE LINKAGE : EXTERNAL PROCEDURE

External procedures are separately assembled Assembly language procedures, often stored in a LIBRARY or diskette. The host program which requires external procedures must have them linked into the compiled code file. Typically, the users write external procedures in Assembly language, to handle low level operations that the Pascal language is not designed to provide. External assembly language procedures are also used for their comparative speed in **real time** applications.

A host program declares that a procedure is external in much the same way as a procedure is declared FORWARD. A standard heading is provided, followed by the keyword EXTERNAL. Calls to the external procedure use standard Pascal syntax, and the compiler checks that calls to the external agree in type and number of parameters with the external declaration. It is the user's responsibility to assure that the Assembly language procedure respects the Pascal external declaration. The Linker checks only that the number of words of parameters agree between the Pascal and Assembly language declarations.

NOTE: For complete information on topics mentioned in this chapter, please refer to:

APPLE PASCAL
LANGUAGE REFERENCE MANUAL
APPLE COMPUTER INC.

REFERENCES

(1) APPLE PASCAL
OPERATING SYSTEM REFERENCE MANUAL
APPLE COMPUTER INC.

(2) APPLE PASCAL
LANGUAGE REFERENCE MANUAL
APPLE COMPUTER INC.

(3) VOICE SYNTHESIS FOR THE TRS-80 COLOR COMPUTER
BYTE, FEBRUARY 1982
WILLIAM BARDEN JR.

(4) VOICE LAB
BYTE, JULY 1983
JOHN E. HOOT

(5) DIGITAL REPRESENTATIONS OF SPEECH SIGNALS
PROCEEDINGS OF THE IEEE
VOLUME 63, APRIL 1975
SCHAFER, LAWRENCE R. AND RONALD W. SCHAFER

(6) AN EXTREMELY LOW COST COMPUTER VOICE RESPONSE SYSTEM
BYTE, FEBRUARY 1981
JAMES C. ANDERSON

(7) APPLE ANALOG TO DIGITAL CONVERSION IN 27 MICROSECONDS
BYTE, OCTOBER 1981
MICHAEL A SEEDS AND HAROLD F LEVISON

(8) THE ATARI TUTORIAL
PART 7 : SOUND
BYTE, MARCH 1982
BOB FRASER

(9) APPROACHING FILTERING DISCRETELY
COMPUTER DESIGN
APRIL 1982
MAZOR, STAN