Decoding Quantimet 920 binary image files

C. Rappoldt

Internal Report 16, June 1988

Agricultural University
Department of Theoretical Production Ecology
P.O. Box 430
6700 AK Wageningen
The Netherlands

List of Internal Reports of Department of Theoretical Production Ecology:

No:

1 D. Barél, F. van Egmond, C. de Jonge, M.J.Frissel, M. Leistra, C.T. de Wit. 1969. Simulatie van de diffusie in lineaire, cylindrische en sferische systemen. Werkgroep: Simulatie van transport in grond en plant.

2 L. Evangelisti and R. van der Weert. 1971. A simulation model for transpiration of crops.

3 J.R. Lambert and F.W.T. Penning de Vries. 1973. Dynamics of water in the soil-plant-atmosphere system: a model named TROIKA.

4 M. Tollenaar. 1971. De fotosynthetische capaciteit.

4a J.N.M. Stricker. 1971. Berekening van de wortellengte per cm^3 grond.

5 L Stroosnijder en H. van Keulen. 1972. Waterbeweging naar de plantenwortel.

6 Th.J.M. Blom and S.R. Troelstra. 1972. A simulation model of the combined transport of water and heat produced by a thermal gradient in porous media.

7a F.W.T. Penning de Vries and H.H. van Laar. 1972. Products and requirements of synthetic processes. Listing of the model and some test runs.

7b F.W.T. Penning de Vries and H.H. van Laar. 1973. Products, requirements and efficiency of biosynthesis. Listing of the model and some test runs.

8 M. Arbab. 1972. A CSMP-program for computing Thornthwaite's classification of climate.

9 P. A. Leffelaar. 1977. A theoretical approach to calculate the anaerobic volume fraction in aerated soil in view of denitrification.

10 L. van Loon and H. Wösten. 1979. A model to simulate evaporation of bare soils in arid regions.

11 C.J.T. Spitters. 1980. Simulating the vegetation dynamics in Sahelian pastures.

12 M. Buil. 1981. Een studie naar de invloed van een aantal diereigenschappen op de primaire en sekundaire productie in de Sahel.

13 D.J. Onstad. 1982. Application of dynamic programming techniques to optimize pest and disease control in winter wheat.

14 P. van de Sanden. 1982. Langgolvige hemelstraling te Niamey.

15 J. Reijerink. 1985. Een simulatie model voor de denitrificatie van NO_3 to N_2 via de intermediaren NO_2 en N_2O.

15a Yan Ying. 1986. A simulation model for dry matter production of maize based on

INTRODUCTION

The Quantimet 920 is able to digitize black and white pictures. They are transformed into 645,120 pixels with a grey level between 0 and 255. The digitized picture may be corrected interactively and a grey level may be chosen above which a pixel is considered to be black. Then, in a process called detection, the grey level of all pixels is compared to the chosen grey level and each pixel becomes either white or black. This results in a so called binary image file. The file contains the binary information on all pixels and has the extension .Q9B (Quantimet 920 Binary).

The Quantimet is designed to perform automatic measurements on objects recognizable on a binary image. An object is a group of (black) pixels, representing for instance a leaf, a bean, a crack or a soil aggregate. It is much harder to measure distances between objects, distances between randomly chosen points and objects, or to do useful things when the (black) pixels form a complicated pattern without isolated groups. A solution to this problem is to transfer binary image files from the quantimet to another machine, on which (self-written) programs run that are able to analyse them.

Programs that make use of binary image files have to read binary data from these files. The programmer may want to look at the image in some arbitrary point to test whether this point is black or white. The Quantimet 920 binary image files with extension .Q9B are difficult to use for that purpose since the image information is coded. A series of black or white pixels is not stored as a (long) series of bits but as a few bytes that give the number of pixels that are black or white. This saves memory space. The .Q9B files do not have a standard length.

This technical note describes a FORTRAN program which decodes Quantimet 920 binary image files and writes the image information to a file with a simpler structure. The resulting file has extension .BIN and contains simply 1 bit per pixel. A .BIN file consists of records that correspond to lines of the image. Since a Quantimet image consists of 720 lines, a .BIN file has 720 records. Each record contains 896 bits, organized as 56 "words" of 16 bits. This means that a certain pixel of the image corresponds to a certain bit of the file. All .BIN files are equally long (80,640 bytes of image data).

The format of the .Q9B files and the decoding algorithm are fully described in the comment header of the main program Q9NORM. A complete understanding requires knowledge of binary, octal and hexadecimal numbers. Further one should know something of the binary representation of a signed INTEGER number.

The use of a .BIN file, however, does not require that background. Reading from file, bits should be organized in groups of 16. Most FORTRAN compilers support the use of 16-bit INTEGER variables. They consist of 2 bytes and are declared as INTEGER*2. A record of a .BIN file can be read then as an array of 56 INTEGER*2

variables. Reading should take place UNFORMATTED. The first pixel of the image is represented by bit 0 of the first word of the first record of the .BIN file. The second pixel by bit 1 ... etc. Finally the lower right pixel of the image is represented by bit 15 of the last word of the 720-th record of the file. A logical function BTEST is available in some compilers and can be used to test the value of a bit of an INTEGER*2 variable (test such a function !!). Otherwise a bit-test procedure has to be written.

The .BIN files can also be read by programs written in another language. When variables are used that are not 16 bits long, one should realize that the order of the bits does not match the order of the pixels anymore (computers generally write the least significant byte first).

Except a listing of the decoding program Q9NORM, also an example program is given that makes use of a .BIN file. This program, called BINLIS reads a .BIN file and produces a matrix printer listing of the image. Use has been made of a FORTRAN function BTEST available in VAX-FORTRAN and in Microsoft FORTRAN version 4.00 for an IBM-PC. This function is only used to test the bits 0-14. Testing the sign bit 15, the Microsoft compiler (version 4.00) returns wrong results. Note that the resulting printable file is eight times larger than the .BIN file. The file may be useful, however, for checking certain features of the image by typing it to screen.

USE OF THE DECODING PROGRAM

The main program Q9NORM, the appropriate version of subroutine GETQ9B and the utilities EXTENS, ENTCHA and ILEN should be compiled and linked. The program asks for the name of the .Q9B file to be decoded. This name may be given without or with extension.

The example program BINLIS makes use of the same utilities EXTENS, ENTCHA and ILEN.

ACKNOWLEDGEMENTS

The program is based on a letter of R. Sijffers of Boss B.V. in Waddinxveen, Holland. The letter contains technical notes originating from Richard Hayden of Cambridge Instruments. I thank Mr. Sijffers for the attention he paid to the problem. Further I thank D. Schoonderbeek of the Stichting voor Bodemkartering for his explanantions on the Quantimet.

LISTINGS

Below listing are given of:

- Main program Q9NORM containing the decoding algorithm and a comment header with full documentation. The main program is written in standard FORTRAN-77 except for the use of INTEGER*2 variables.
- Subroutine GETQ9B. The function of this routine is to return a single 16-bits word of the .Q9B input file. Since .Q9B files have no record structure, the reading procedure could not be written in standard FORTRAN-77. Two versions are given, one for a VAX and one for an IBM-PC with Microsoft FORTRAN version 4.00. Note that, in both versions, the input file is opened in an initial section that is executed only once. Subroutine GETQ9B probably has to be adapted when the program is used on a different machine.
- Three utility routines, EXTENS for changing the file name extension, ENTCHA for asking a question and ILEN for determining the actual length of a string. Except for a '$' in routine ENTCHA (to hold the cursor), these utilities are written in standard FORTRAN-77.
- Program BINLIS. This example program makes use of the (non-standard) FORTRAN function BTEST to check the value of the bits representing the image pixels. The program works on a VAX and on an IBM-PC.

```
PROGRAM Q9NORM

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                           *
*        Transforms Quantimet 920 .Q9B binary files into                    *
*         .BIN files containing simply a bit per pixel.                     *
*        ---------------------------------------------                      *
*                                                                           *
*     Author: C. Rappoldt                                                   *
*             Agricultural University                                       *
*             Department of Theoretical Production Ecology                  *
*             P.O. Box 430                                                  *
*             6700 AK Wageningen,  The Netherlands                          *
*     Date: March 1988.                                                     *
*                                                                           *
*     Program based on a letter of R. Sijffers of Boss B.V.                 *
*     in Waddinxveen, Holland. The letter contains the following            *
*     notes of Richard Hayden of Cambridge Instruments:                     *
*                                                                           *
*     - Binary .Q9B image files are not masked by the image frame           *
*       when saved to disk. The entire image (896 pixels x 720 lines)       *
*       is saved in the file.                                               *
*     - Binary image files have no header.                                  *
*     - Pixels are taken from left to right across each line,               *
*       starting with the top line and working down.                        *
*     - The data is coded to reduce the file size.                          *
*     - The binary image data is read from the image store as               *
*       16-bit words (i.e. blocks of 16 binary pixels).                     *
*       Three types of words are recognised:                                *
*       1. All ones (177777 octal)                                          *
*       2. All zero's (000000 octal)                                        *
*       3. Literal - anything else                                          *
*                    (i.e. words containing 1's and 0's)                    *
*                                                                           *
*     - The image is sequentially encoded into a stream of                  *
*       ones, zero's and literals.                                          *
*     - This gives four types of word in the .Q9B data file:                *
*       a. Ones word : Bit 15 is SET to indicate ones                       *
*                      Bit 0..14 contain the number of ones                 *
*                      words (ex. 100005 octal = 5 words                    *
*                      containing all ones)                                 *
*       b. Zeros word: Bit 15 is CLEAR to indicate zeros.                   *
*                      Bit 0..14 contain number of zeros words.             *
*                      (ex. 000005 octal means 5 words                      *
*                      containing all zeros)                                *
*       c. Zero        0, indicates the following word (a single            *
*                      word only) is a literal word.                        *
*       d. Literal     An uncoded word containing 1's and 0's               *
*                      as read by the binary read channel. A                *
*                      literal word is always preceded by 0.                *
*                                                                           *
*       NOTE - the number of zeros words encoded in a .Q9B word             *
*              of type (b) cannot be zero. So it is not possible            *
*              to have a type (b) word of value 0.                          *
*                      ---------------------                                *
```

```
*                                                                       *
*     NOTES ON THIS PROGRAM                                             *
*     ---------------------                                             *
*     The words in the .Q9B file are read as INTEGER*2 variables.       *
*     This means that the first byte of the file should contain         *
*     bit 0-7 of the first 16-bit word and the second byte bit 8-15.    *
*     This is the usual way of writing 2 byte words to memory. The      *
*     total pixel count generated from a .Q9B file by this program      *
*     correspondeds to 720 x 896, confirming this assumption.           *
*                                                                       *
*     The decoded ones-words, zeros-words and literals are also         *
*     in the form of INTEGER*2 variables and are written to an          *
*     unformatted sequential file. Every record of the file             *
*     contains 56 words, corresponding to 56 x 16 = 896 pixels.         *
*     So each record corresponds to a line of the image and the         *
*     output file contains 720 records.                                 *
*                                                                       *
*     A FORTRAN signed INTEGER*2 variable has a negative value if       *
*     bit 15 is SET. So .Q9B words of type (a) have a negative          *
*     value, the words of type (b) have a positive value and the        *
*     zero words have a zero value. The value of a (b) type word        *
*     equals the number of zeros words that should be written           *
*     to the output .BIN file. The get the number of ones-words         *
*     from a type (a) .Q9B word, bit 15 should be cleared.              *
*                                                                       *
*     The simplest way to clear bit 15 of a type (a) word is using      *
*     the IBCLR intrinsic function. This appears not to work with       *
*     the Microsoft FORTRAN 77 compiler (version 4.00) on an IBM-PC.    *
*     (The function works correctly on INTEGER*4 ; there seems to       *
*     be a compiler bug for INTEGER*2 arguments). Therefore, to         *
*     clear bit 15 the following machine independent procedure is       *
*     used: tranform to INTEGER*4 and add 32768.                        *
*     An example illustrates the method:                                *
*     - A type (a) word is found ; value -32763                         *
*     - The binary representation is  1000 0000 0000 0101               *
*     - The hexadecimal representation is  8005, clearly showing        *
*       that the desired result is 5.                                   *
*     - Tranforming to INT*4 gives           FFFF8005 (hex)             *
*     - Adding 32768 (=00008000 (hex)) gives 00000005, with value 5     *
*                                                                       *
*     USING THE PROGRAM                                                 *
*     -----------------                                                 *
*     The main program Q9NORM is written in standard FORTRAN 77         *
*     except for the usage of INTEGER*2 variables. The SUBROUTINES      *
*     belonging to it are:                                              *
*     GETQ9B - Reads a 16-bit word from the .Q9B input file. A VAX      *
*              and an IBM-PC version have been written. The routines    *
*              deviate from standard FORTRAN 77 in their use of         *
*              INTEGER*2 variables and in file access.                  *
*     EXTENS - Replaces the extension of a filename.                    *
*              Machine independent FORTRAN-77 utility program.          *
*     ENTCHA - For entering a string from the keyboard.                 *
*              Machine independent utility program. Standard FORTRAN    *
*              except a '$' in a FORMAT string (holding the cursor).    *
*     ILEN   - Determines the length of a character string.             *
```

```
*             Machine independent FORTRAN-77 utility program.        *
*     So the VAX and the IBM-PC version differ in GETQ9B only.       *
*                                                                    *
*     INPUT FILE                                                     *
*     ----------                                                     *
*       The program has been tested on a Quantimet 920 binary        *
*     file TEST.Q9B (present on floppy). This file has been          *
*     sent from a PDP 11 machine (RT 11 operating system) to         *
*     an IBM-PC using an 8-bit Kermit protocol. From the IBM         *
*     it was sent to a VAX mainframe (also using 8-bit Kermit).      *
*       On the IBM the file arrived as a long series of binary       *
*     data without structure. Therefore, in SUBROUTINE GETQ9B        *
*     (IBM version) the input file is opened as an unformatted,      *
*     sequential, binary file. Microsoft FORTRAN 77 (version 4.00)   *
*     enables the user to read single words until End_Of_File        *
*     reached.                                                       *
*       The VAX brings some artificial structure into the file.      *
*     Reading it as a stream of words appeared to be impossible.     *
*     It is split up in records of 510 bytes, eached stored in a     *
*     physical block of 512 bytes on disk. The first two bytes of    *
*     each block give the length of the record. This does not lead to *
*     problems except in reading the last record of the file, which  *
*     has a different length. Record length can only be determined   *
*     by means of a Q format in FORMATTED access. Therefore, at      *
*     first, all record lengths are read from file. Reading data     *
*     the file is re-opened for UNFORMATTED access. Filetype         *
*     specification on the VAX:                                      *
*     File organization:  Sequential                                 *
*     Record format:      Variable length, maximum 510 bytes         *
*     Record attributes:  None                                       *
*                                                                    *
*     USING THE .BIN FILE                                            *
*     -------------------                                            *
*     The file produced by this program is used by opening it        *
*     as a sequential, unformatted file. It contains 720 records     *
*     for the 720 image lines. Each record contains 56 16-bit        *
*     words that can be read as INTEGER*2 values. The pixel          *
*     sequence for each line is then bit 0 ... bit 15 of word 1      *
*                                   bit 0 ... bit 15 of word 2       *
*                                     ...........                    *
*                                   bit 0 ... bit 15 of word 56      *
*     The contents of a .BIN file can be stored into an array of     *
*     INTEGER*2 variables to reduce disk access. One may declare     *
*     for instance:                                                  *
*           INTEGER*2 IMAGE                                          *
*           DIMENSION IMAGE(720,56)                                  *
*     and read image data from file going from line 1 to line 720.  *
*     A second possibility is to transform the file into a DIRECT    *
*     access file with a separate program.                           *
*                                                                    *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
*     all variables are declared
      INTEGER*2 BUF16,BUFFER,WORD
      INTEGER BPOINT,IPIC,ILINE,IWORD,N,NPIXEL,LLINIWORD,N,NPIXEL,LLINES,LWORDS
      PARAMETER (LWORDS=56,LLINES=720)
      DIMENSION BUFFER(LWORDS)
      CHARACTER*40 NAME,FILEIN,FILEOU
      LOGICAL EOF

*     number of pixels in image
      NPIXEL = 16 * LWORDS * LLINES

*     enter filename and set extensions
      CALL ENTCHA ('Name of Quantimet .Q9B binary file',NAME)
      CALL EXTENS (NAME,'Q9B',0,FILEIN)
      CALL EXTENS (NAME,'BIN',1,FILEOU)

*     get first word from Quantimet input file ; initialize counters
      CALL GETQ9B (FILEIN,WORD,EOF)
      IWORD = 1
      ILINE = 0
      IPIC = 0

*     open output file
      OPEN (41,FILE=FILEOU,STATUS='NEW',ACCESS='SEQUENTIAL',
     $         FORM='UNFORMATTED')

*     do while loop
10    IF (.NOT.EOF) THEN

         IF (WORD.EQ.0) THEN
*           next word = literal
            CALL GETQ9B (FILEIN,WORD,EOF)
            IF (EOF) STOP 'Unexpected End_Of_File'
            IWORD = IWORD + 1

            IF (WORD.NE.0) THEN
*              literal found
               N = 1
               IPIC= IPIC + 16
*              set 16 bits for output buffer equal to .Q9B word
               BUF16 = WORD
            ELSE

*              a zero literal found ; may be file format error
               IF (IPIC.EQ.NPIXEL) THEN
*                 all pixels were already found ; no error
                  N = 0
               ELSE
                  WRITE (*,'(A,2(/,A,I7,A))')
     $               '  ERROR in .Q9B file format after',
     $               '  reading',IWORD,' words from file',
     $               '  and decoding ',IPIC,' bits'
                  STOP
               END IF
            END IF
```

```
          ELSE IF (WORD.LT.0) THEN
*            a 1-word found ; clear bit 15 by transforming
*            the 16 bit .Q9B word to INTEGER*4 and adding 32768
             N = WORD
             N = N + 32768
             IPIC = IPIC + 16*N
*            set 16 bits for output buffer to FFFF (hex)
             BUF16 = -1

          ELSE IF (WORD.GT.0) THEN
*            a zero'th word found
             N = WORD
             IPIC = IPIC + 16*N
*            set 16 bits for output buffer to 0000 (hex)
             BUF16 = 0
          END IF

*         fill buffer and write to output file when necessary
20        IF (N.GT.0) THEN
             N = N - 1
             BPOINT = BPOINT + 1
             BUFFER(BPOINT) = BUF16

             IF (BPOINT.EQ.LWORDS) THEN
*               full image line decoded ; write it to file
                ILINE = ILINE + 1
                WRITE (41) BUFFER
                BPOINT = 0
*               message to screen
                IF (MOD(ILINE,20).EQ.0) WRITE (*,'(A,I3,A,I6,A)')
     $           ' Image line ',ILINE,' decoded ; ',IWORD,' words read'
             END IF
          GOTO 20
          END IF

*         get next word
          CALL GETQ9B (FILEIN,WORD,EOF)
          IWORD = IWORD + 1
       GOTO 10
       END IF

       WRITE (*,'(1X,I6,A)')
      $ IPIC,' pixels found ; transformation completed'
       STOP
       END
```

```
      SUBROUTINE GETQ9B (FILEIN,WORD16,EOF)

*                V A X    V E R S I O N

*     Reads a 16 bits words from a .Q9B file on a VAX
*     At each CALL the routine returns the next word on the
*     file to the calling program until End_Of_File reached.

*     In an inital part a filename is asked for interactively.
*     Then record lengths are determined and the length of the
*     file in bytes is displayed. The first word is returned.
*     In an initial part record lengths are determined.

*     In subsequent CALL's all other words in the file are
*     returned. Word and record counting is done internally.

*     dummy variables of the routine:
*     FILEIN - name of input file ; used in first CALL only
*     WORD16 - returned word ; 16 bits INTEGER
*     EOF    - logical is .TRUE. when End_Of_File reached

*     Author: Kees Rappoldt
*     Date: March 1988

      IMPLICIT NONE

      INTEGER I,IOS,LENGTH,LENMAX,FILEN,NW,IWOUT
      INTEGER RLEN,ILR,IREC,NREC,LFIL,ILEN
      PARAMETER (LENMAX=510,ILR=500)

      INTEGER*2 WORD,WORD16
      DIMENSION WORD(LENMAX/2),RLEN(ILR)

      CHARACTER FILNAM*40,FILEIN*(*)
      LOGICAL EOF,ERROR,INIT,EXIST
      DATA INIT/.FALSE./

      IF (.NOT.INIT) THEN
*        get local copy of input filename
         LFIL = ILEN (FILEIN)
         IF (LFIL.GT.40) STOP 'ERROR in GETQ9B: filename too long'
         FILNAM(1:LFIL) = FILEIN(1:LFIL)

*        check existance
         INQUIRE (FILE=FILNAM(1:LFIL),EXIST=EXIST)
         IF (.NOT.EXIST) THEN
            WRITE (*,'(1X,2A)')
     $       'ERROR in GETQ9B: cannot find file ',FILNAM(1:LFIL)
            STOP
         END IF

*        get array for formatted input to read record lengths
         OPEN (40,FILE=FILNAM(1:LFIL),ACCESS='SEQUENTIAL',
     $    STATUS='OLD',FORM='FORMATTED',RECORDTYPE='VARIABLE')
```

```
*        record counter
         NREC = 0

*        do while loop
10       CONTINUE
         READ (40,'(Q)',IOSTAT=IOS) LENGTH
         EOF = IOS.LT.0
         IF (.NOT.EOF) THEN
            IF (IOS.GT.0)
     $       STOP 'ERROR in GETQ9B: in reading record length'
            IF (LENGTH.GT.LENMAX)
     $       STOP 'ERROR in GETQ9B: record length > LENMAX'
            NREC = NREC + 1

*           store record length in array
            RLEN(NREC) = LENGTH

*           determine filelength in bytes
            FILEN = FILEN + LENGTH
         GOTO 10
         END IF

*        close file and message
         CLOSE (40)
         WRITE (*,'(3A,I7,A,/)')
     $    ' File ',FILNAM(1:LFIL),' contains ',FILEN,' bytes'

*        open file for unformatted reading
         OPEN (40,FILE=FILNAM(1:LFIL),ACCESS='SEQUENTIAL',STATUS='OLD',
     $    FORM='UNFORMATTED',RECORDTYPE='VARIABLE')

*        set record and word output counter to 0
         IWOUT = 0
         IREC = 0

*        initial part completed
         INIT = .TRUE.
      END IF


      IF (IWOUT.EQ.NW .AND. IREC.EQ.NREC) THEN
*        last record was sent
         WRITE (*,'(/,3A)') ' End_Of_File ',FILNAM(1:LFIL),
     $   ' reached by input routine GETQ9B'
         CLOSE (40)
         INIT = .FALSE.
         EOF = .TRUE.
         WORD16 = 0
         RETURN

      ELSE IF (IREC.EQ.0 .OR. IWOUT.EQ.NW) THEN
*        read first/new record
         IREC = IREC + 1
         NW = RLEN(IREC) / 2
         READ (40,IOSTAT=IOS) (WORD(I),I=1,NW)
```

```
*        error check
         ERROR = IOS.GT.0
         EOF = IOS.LT.0
         IF (ERROR) THEN
            WRITE (*,'(A,I2)')
     $       ' ERROR in GETQ9B: during READ operation IOS=',IOS
            STOP
         ELSE IF (EOF) THEN
            STOP 'ERROR in GETQ9B: unexpected End_Of_File'
         END IF

*        reset IWOUT
         IWOUT = 0
       END IF

*      return (next) word
       IWOUT  = IWOUT + 1
       WORD16 = WORD(IWOUT)
       EOF    = .FALSE.

       RETURN
       END
```

```
      SUBROUTINE GETQ9B (FILEIN,WORD16,EOF)

*              I B M - P C    V E R S I O N

*     Reads a 16 bits words from a .Q9B file on an IBM PC.
*     At each CALL the routine returns the next word on the
*     file to the calling program until End_Of_File reached.

*     In an inital part a filename is asked for interactively.
*     Then record lengths are determined and the length of the
*     file in bytes is displayed. The first word is returned.
*     In an initial part record lengths are determined.

*     In subsequent CALL's all other words in the file are
*     returned. Word and record counting is done internally.

*     dummy variables of the routine:
*     FILEIN - name of input file ; used in first CALL only
*     WORD16 - returned word ; 16 bits INTEGER
*     EOF    - logical is .TRUE. when End_Of_File reached

*     Author: Kees Rappoldt
*     Date: March 1988

      INTEGER ILEN,LFIL,IWOUT,IOS,FILEN
      INTEGER*2 WORD16

      CHARACTER FILNAM*40,FILEIN*(*)
      LOGICAL EOF,INIT,EXIST
      DATA INIT/.FALSE./

      IF (.NOT.INIT) THEN
*        get local copy of input filename
         LFIL = ILEN (FILEIN)
         IF (LFIL.GT.40) STOP 'ERROR in GETQ9B: filename too long'
         FILNAM(1:LFIL) = FILEIN(1:LFIL)

*        check existance
         INQUIRE (FILE=FILNAM(1:LFIL),EXIST=EXIST)
         IF (.NOT.EXIST) THEN
            WRITE (*,'(1X,2A)')
     $       'ERROR in GETQ9B: cannot find file ',FILNAM(1:LFIL)
            STOP
         END IF

*        get array for formatted input to read record lengths
         OPEN (40,FILE=FILNAM(1:LFIL),ACCESS='SEQUENTIAL',
     $    STATUS='OLD',FORM='BINARY')

*        set word output counter to 0
         IWOUT = 0

*        initial part completed
         INIT = .TRUE.
      END IF
```

```
*     read word from file
      READ (40,IOSTAT=IOS) WORD16

      IF (IOS.LT.0) THEN
*        last record was sent
         FILEN = 2 * IWOUT
         WRITE (*,'(/,3A,/,A,I5,A)')
     $   ' End_Of_File ',FILNAM(1:LFIL),
     $   ' reached by input routine GETQ9B',
     $   ' after reading ',FILEN,' bytes'
         CLOSE (40)
         INIT = .FALSE.
         EOF = .TRUE.
         WORD16 = 0
         RETURN

      ELSE IF (IOS.GT.0) THEN
         WRITE (*,'(A,I2)')
     $    ' ERROR in GETQ9B: during READ operation IOS=',IOS
         STOP
      END IF

*     return (next) word
      IWOUT  = IWOUT + 1
      EOF    = .FALSE.

      RETURN
      END
```

```
      SUBROUTINE EXTENS (FILEIN,NEWEXT,ICHECK,FILEOU)

*     Change extension of filename.
*     Output filename is filled with characters of input
*     filename and new extension until end is reached
*     Output filename is in uppercase characters

*     FILEIN - Input name with old extension or without extension  (I)
*     NEWEXT - New extension ; is set to uppercase                 (I)
*     ICHECK - check request parameter                             (I)
*              0 = no check
*              1 = check on equal output and input extension
*     FILEOU - Output filename with new extension in uppercase     (O)

*     Author: Kees Rappoldt
*     Date: March 1988

      IMPLICIT REAL (A-Z)
      CHARACTER*(*) FILEIN,FILEOU,NEWEXT
      CHARACTER*1 DUM,OLD,NEW
      INTEGER ICHECK,I,IC,IEXT,ILEN,ILEXT1,ILEXT2,ILFIL,INAME,IP,LOUT
      LOGICAL CHECK

*     length of input filename and new extension
      ILFIL = ILEN (FILEIN)
      ILEXT2 = ILEN (NEWEXT)
      IF (ILFIL.EQ.0) STOP 'ERROR in EXTENS: no filename supplied'
      IF (ILEXT2.EQ.0) STOP 'ERROR in EXTENS: no new extension supplied'

*     position of point ; dimensioned length output name
      IP = INDEX (FILEIN,'.')
      LOUT = LEN (FILEOU)

*     length of first part of filename ; length of input extension
      IF (IP.EQ.0) THEN
*        no input extension
         INAME = ILFIL
         ILEXT1 = 0
      ELSE
         INAME = IP-1
         ILEXT1 = ILFIL-IP
      END IF
      IF (LOUT.LE.INAME+2)
     $ STOP 'ERROR in EXTENS: output string too short for new name'

*     initialize output name and pointer
      I = 0
      FILEOU = ' '

10    IF (I.LT.INAME) THEN
*        transfer character from input name to output name
         I = I + 1
*        transform it to uppercase
         DUM = FILEIN (I:I)
         IC = ICHAR (DUM)
```

```
         IF (IC.GE.97 .AND. IC.LE.122) DUM=CHAR (64+MOD (IC,32))
         FILEOU(I:I) = DUM
      GOTO 10
      END IF

*     set point
      I = I + 1
      FILEOU(I:I) = '.'

*     actual length of new extension
      ILEXT2 = MIN (ILEXT2, LOUT-I)
*     if old and new extension are equally long ; check !!
      CHECK = (ILEXT1.EQ.ILEXT2) .AND. (ICHECK.NE.0)

20    IF (I .LT. INAME+1+ILEXT2) THEN
*        get (next) extension character
         I = I + 1

         IF (CHECK) THEN
*           get (next) character old extension for comparison
            OLD = FILEIN(I:I)
            IC = ICHAR (OLD)
            IF (IC.GE.97 .AND. IC.LE.122) OLD=CHAR (64+MOD (IC,32))
         END IF

*        get (next) character new extension
         IEXT = I-INAME-1
         NEW = NEWEXT(IEXT:IEXT)
         IC = ICHAR (NEW)
         IF (IC.GE.97 .AND. IC.LE.122) NEW=CHAR (64+MOD (IC,32))

*        a difference is found ?
         IF (CHECK .AND. OLD.NE.NEW) CHECK=.FALSE.

*        put into output string
         FILEOU(I:I) = NEW
      GOTO 20
      END IF

      IF (CHECK) STOP 'ERROR in EXTENS: old and new extension equal'

      RETURN
      END
```

```
      SUBROUTINE ENTCHA (QUEST,X)

*     Interactive entry of a character string.
*     Writes the text QUEST on screen as a "question" and
*     returns entered string to calling program.

*     QUEST - character string for instance 'name' (I)
*     X     - entered character string (O)

*     Author: Kees Rappoldt
*     Date: Aug 87


      IMPLICIT REAL (A-Z)
      CHARACTER*(*) QUEST,X
      INTEGER LQ,LX

*     declared string lengths
      LQ = LEN (QUEST)
      LX = LEN (X)

*     ask the question and read the answer
10    CONTINUE
      WRITE (*,'(A50,A$)') QUEST(1:LQ),': '
      READ (*,'(A)',ERR=20,END=30) X(1:LX)
      RETURN

*     error during READ : give message and try again
20    CONTINUE
      WRITE (*,'(/,A,/,A,/)')
     $   ' Enter a CHARACTER string !!',
     $   ' (or  <CTRL> Z  to STOP execution)'
      GOTO 10

30    CONTINUE

      STOP 'End_Of_File detected by ENTCHA ; program STOP'
      END
```

```
      INTEGER FUNCTION ILEN (STRING)

*     this subroutine determines the significant length (ILEN)
*     of a string (STRING), if the string is empty a zero is returned

*     ILEN, returned length (O)
*     STRING, input string  (I)

*     Author: Daniel van Kraalingen
*     Date: Aug 87


      CHARACTER*(*) STRING

      DO 10 ILEN=LEN(STRING),1,-1
         IF (STRING(ILEN:ILEN).NE.' ') RETURN
10    CONTINUE

      RETURN
      END
```

```
      PROGRAM BINLIS

*     produces a printer listing of a .BIN image file
*     (resulting file contains 1 byte per pixel !!!!!)

      IMPLICIT REAL (A-Z)
      INTEGER*2 BUFFER
      CHARACTER*1 LINE
      INTEGER LWORDS,LLINES,SIZCOL,ICOL,IC
      INTEGER IW1,IW2,IW,IL,IP,IBIT,BUF
      PARAMETER (LWORDS=56,LLINES=720,SIZCOL=8,ICOL=7)
      DIMENSION BUFFER(LWORDS),LINE(128)
      CHARACTER*40 NAME,FILEIN,FILEOU

*     filename manipulation
      CALL ENTCHA ('Name of .BIN input file',NAME)
      CALL EXTENS (NAME,'BIN',0,FILEIN)
      CALL EXTENS (NAME,'LIS',1,FILEOU)

*     open files
      OPEN (40,FILE=FILEIN,STATUS='OLD',ACCESS='SEQUENTIAL',
     $ FORM='UNFORMATTED')
      OPEN (41,FILE=FILEOU,STATUS='NEW')

      DO 40 IC=1,ICOL

         WRITE (41,'(A,I2,A)')
     $   '----------- here column',IC,' -----------'
         IW1 = (IC-1) * SIZCOL + 1
         IW2 = (IC-1) * SIZCOL + SIZCOL

         DO 30 IL=1,LLINES
*           read image line
            READ (40) BUFFER

*           build up listing from word IW1 to word IW2
            IP = 0
            DO 20 IW=IW1,IW2
*              transform to INT*4
               BUF = BUFFER(IW)

*              test bits 0-14
               DO 10 IBIT=0,14
                  IP = IP + 1
                  IF (BTEST(BUF,IBIT)) THEN
*                    the tested bit is set
                     LINE(IP) = 'W'
                  ELSE
*                    the tested bit is zero
                     LINE(IP) = '.'
                  END IF
10             CONTINUE

*              test for sign bit 15
               IP = IP + 1
```

```
               IF (BUF.LT.0) THEN
                  LINE(IP) = 'W'
               ELSE
                  LINE(IP) = '.'
               END IF
20          CONTINUE

         WRITE (41,'(1X,200A1)') LINE
*        message to screen
         IF (MOD(IL,40).EQ.0) WRITE (*,'(1X,A,I2,A,I3,A)')
     $   '  column',IC,' image line ',IL,' listed'
30       CONTINUE

         REWIND (40)
40     CONTINUE

       STOP
       END
```