```
/* link95.c

Following DOS 16 bit code more or less parses a *.lnk
file as generated by Win9x.  I post it cause someone
asked about the format of *.lnk files, and I haven't
seen anything about this anywhere before.  Likely someone
has done a more complete job and I just don't know about
it!  Please let me know if this is useful, or you know
the complete solution.
    w_kranz@conknet.com

2/24/99 revisite, rename previous version link95a.c
   Note see win95\resource.lzh:shortcut.exe, got somewhere
   (probably Microsoft) dumps shortcuts

16 bit code for parsing win95 *.lnk files
    Used MSC 6.0  (ie WORD => unsigned short => 2 bytes)

Much of data in file still unknown, of initial 0x4C bytes in
*.lnk file, much is constant but there is variability from
one file to the next.  Can parse to the descriptive strings
at the end of the file with following:

On my machine, a dump at the beginning of the file always looks
the same up through offset 0x13:
   00000: 4C 00 00 00  01 14 02 00  00 00 00 00  C0 00 00 00
   00010: 00 00 00 46
Next byte in file varies, probably a bitmap per below.

Following the header at offset 0x4C in file is a two byte word
which is the length of 1st variable length block.
This length is # bytes to skip to reach next block or 0.
If zero block is empty?  Skip bytes read to next length.
The block length includes the two bytes in the length word!

CAUTION, following seems to be true but always?
There are 3 such blocks (including zeros for empty blocks).

Then a string space starts (its organized slightly differently).
Each two byte (word) length is the number of bytes in string
which follows the word.  There are no NUL terminators, if
desired you must add them.  It appears this region is always
terminated by two words, both equal to zero.  I think one
could read till found first string block of zero length.
This is the end of the useable data in the file.

The byte at offset 0x14 in file seems to identify
the individual blocks and strings included in file.
I define the following bitmaps, each bit indicates
a different function and associated block or string.

Look at low nibble for blocks of data as follows (probably):
1 => ?  occurs but meaning escapes me, see com114~1.lnk
2 => target program, see compus~1.lnk with above
4 => ?  have never even seen this
8 => path componets, see notepad.lnk & cdplay~1.lnk
     note also forces an extra string!

In addition to bit 4 (mask value 8) of low nibble
Look at the high nibble for strings.  It is a bitmap
of which strings are expected in the order shown below.
looking at magic # byte at 0x14
```

```
0x10 => working directory
0x20 => arguments for program
0x40 => icon file name
per above if low nibble & 8, 1st string is
    The path to *.exe relative to current location of *.lnk

One expects any combination of bits above with 1 to 4
strings possible.
Interesting in that strings do not appear to be in unicode!

Now it gets pretty vague.  I don't fully understand
the various blocks, and almost nothing in the header.
Not clear why strings should be arranged differently than
data blocks.  In particular why do block length differently
and why have empty block if bit map tells us which ones
are there?  The bit map concept is just a guess for the
data blocks, but seems to work for the strings so logical.
Maybe one is supposed to use bitmap info also, only thing
I've done for verification is:

If ( (low nibble of byte at 0x14) & 0x2)
    shortcut.exe says there is a target, otherwise none

In general each block is made up from a number of sub
blocks.  Each two byte word at the begining of a block defines
the sub block length.  The blocks do NOT seem to occur in order
of their bit map value, target type 2 always last:
Type     description
8       1st sub block is 0xE unknown bytes
        Each additional sub block is one level in path spec.
        ie 2nd is drive, remainder are nodes in path
        or file name at end of block.  Per above if this
        exists, 1st string below seems to be relative path to
        program from location of *.lnk file.

2       This contains the full path to the target program.
        Starts with 0x2D unknown bytes, followed by a
        null terminated path to the program at offset 0x2d.
        WARNING in general above is true, but a couple of my
        shortcuts just had "C:\" at this offset,
        then another sub block, then the rest of the path
        in a third sub block.
        This occurs if the fifth word in block is 0x3
        rather than 0x1.  Seems to give local path and
        network path.  See code below.

I'd like this to be a little cleaner, I don't love Microsoft,
but to have two different methods of storing data in the same
little file seems odd.  Likely I overcomplicated things somewhere,
but don't see it.  If you do let me know!
*/

#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>

#define BUFSZ 512    // global working buffer in case needs to be big
char buf[BUFSZ];
#define HEADSZ 0x4c  // don't know what is in here! only identified two bytes

main(int argc, char *argv[])
{
```

```
    unsigned char tndx,tmask,head[HEADSZ];
    long flen,len=0L;
    char *type,*str;
    int i,fp,rd,cnt=0,scnt=0;
    unsigned *uptr,sz=0x4c; /* fixed size of initial region */
    if(argc < 2)
    {
         printf("\nusage: link95 <filename>\ndisplays link info");
         exit (1);
    }
    if((fp = open(argv[1],O_RDONLY|O_BINARY)) == EOF)
    {
         printf("\nfailed to open: %s",argv[1]);
         exit(1);
    }
    else if (read(fp,head,HEADSZ) != HEADSZ)
    {
         printf("\nfailed to read a header of length %x (hex) bytes)",
              HEADSZ);
         exit(1);
    }
    len = HEADSZ;
    flen = filelength(fp);
    while(cnt < 3 && (rd = read_block(fp,buf,BUFSZ)) >= 0)
    {
         if(rd == 0)
              len+= sizeof(unsigned); // empty block
         else
              len += rd; // size of block, includes unsigned
         str = "None";
         type = NULL;
         switch(++cnt)
         {
              /* hopefully I'll know what other blocks are some day...
                 my experience has been that if there is a 3rd data block
                 which isn't empty, its the Target block associated
                 with bit 0x2.  Maybe there is an id embedded in the
                 blocks, but I don't see it...
              */
              case 3:
                 uptr = (unsigned *) &buf[0];
                 // should I be checking for an 0x2 in bitmap?
                 type = "Target";
                 if(rd > 0x2d)  // 0x2d == *(uptr+6)
                    str = buf+0x2d; // display absolute path string
                    // above works for me but could be variable offset ie
*(uptr+6)
                 printf("\n%s: %s",type,str);
                 // CAUTION not always as simple as above
                 if(*(uptr+4) >= 3 && *(uptr+12) < rd)
                 {  // skip over computer name to create local target
                    printf("%s",buf + *(uptr+12));
                 }
                 break;
         }
    }

    if((head[0x14] & 8))
    { /* if bit set then had a data block above and
         first string should be extra relative path.
         Note that shortcut.exe does NOT display this string
       */
        if((rd = read_string(fp,buf,BUFSZ)) <= 0)
```

```
            printf("\nNo strings found");
        else
        {
            printf("\nrelative path string: %s",buf);
            len += sizeof(unsigned) + rd; // word + strlen was read
         }
    }

    tmask = head[0x14] >> 4; // high nibble of byte
    // check all three possible states
    for(tndx = 1;len < flen && tndx <= 4;)
    {
         switch(tndx)
         {  // the following match shortcut.exe output
              case 1:
                   type ="Working directory";
                   break;
              case 2:
                   type ="Arguments";
                   break;
              case 4:
                   type = "Icon file";
                   break;
         }
         if(tndx & tmask)
         {
             // there should be another string
             if((rd = read_string(fp,buf,BUFSZ)) <= 0)
             {
                  printf("\nerror reading string");
                  break;
             }
             else
                  len += sizeof(unsigned) + rd; // word + strlen was read
         }
         else
              strcpy(buf,"None");

         tndx = tndx << 1;
         printf("\n%s: %s",type,buf);
    }
    printf("\nread %ld bytes out of %ld in file",len,flen);
}

/* each block starts with an unsigned word
   may contain nested block
   or word may be zero, in this case block empty, just
   advance 2 bytes for this word
   word read at start of block is returns, ie 0 for empty block
*/
read_block(int fp,unsigned char *buf,unsigned msz)
{
    int rd,sz;
    unsigned *uptr = (unsigned *)buf;
    if(msz < sizeof(unsigned) ||
       read(fp,buf,sizeof(unsigned)) != sizeof(unsigned))
         return(-1);
    else if(*uptr > msz)
         return(-2);
    sz = *uptr - sizeof(unsigned); // additional to read in
    if(sz > 0 && read(fp,buf+sizeof(unsigned),sz) != sz)
         return(-1);
    else
```

```
        return(*uptr);  // bytes read
}

/* file ends with a series of strings, logic is similar to
   above, but length is # chars in string, and they aren't nul
   terminated, I append a NUL to make printing easier if it fits
   Note I'm NOT putting len in buffer this time, want a string
*/
read_string(int fp,unsigned char *buf,unsigned msz)
{
    int rd,sz;
    if(read(fp,&sz,sizeof(unsigned)) != sizeof(unsigned))
         return(-1);
    else if(sz > msz)
         return(-2);
    if(sz > 0 && read(fp,buf,sz) != sz)
         return(-1);
    else
    {
         if(sz >= 0 && sz < msz)
              buf[sz] = 0;
         return(sz);  // bytes read into string buffer, ie its length
    }
}
```