

```

/*
A Program to Convert Wordstar files (versions 3/4/5/7) to HTML format.
original version Apr 13, 1999

major revision and expansion v. 7, July 13, 2019

syntax is: WS_HTM directory

where directory is a filespec
-----
environment: Linux
-----
This program is in the public domain, and may be
used for any purpose whatsoever, without permission
or notification, but it is offered "as-is" without
any warrentee, express or implied.
-----
This is not debugged code. It is published as a interim measure in case
my health should prevent me from completing it

For Ayantu and Dayanna, Daughters of my Heart

Andrew D. Todd
1249 Pineview Dr., Apt 1
Morgantown, WV 26505

*/
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <dirent.h>
/*-----*/
/*
get in the directory
read the parm and establish that it is a path.
read in all the directory entries for this path,
and put them in an array of directory entries
*/

struct file_table_type
{
char old_filename_ext[32];
char old_f_spec[256];
char new_f_spec[256];
char new_title[256];
}
table_of_old_files[1600];
/*
Wordstar 3 did not support directories internally.
Consequently, huge numbers of files might wind up
being placed in a single directory on a hard drive,
named in a logical fashion, eg. two letters for the book,
two numerals for the chapter, and one letter for
the version. The wordstar program files would go
in the same directory. The user would just make
that directory the default directory, and proceed
as if the rest of the hard drive did not exist.
1600 files seems a prudent allocation.
*/
main(int argc, char *argv[])
{
int num_files;
char dir_name[256];
/*get the directory name*/
if( argc < 2 )
{
printf(" \nDirectory Name Needed\n");
_exit(3);
}
strcpy(&dir_name, argv[1]);
num_files = make_a_table_of_old_files(&dir_name, table_of_old_files);
add_titles_to_table(dir_name, table_of_old_files, num_files);
qsort(
table_of_old_files,
num_files,
sizeof(struct file_table_type),
cmp_final_order
);
process_wordstar_files(dir_name, table_of_old_files, num_files);
_exit(0);
}
/*-----*/

int cmp_final_order(void *name_and_ext1, void *name_and_ext2)
{
int class1 = find_priority_class(name_and_ext1);
int class2 = find_priority_class(name_and_ext2);
if(class1 != class2) return(class1-class2);
else if( (class1 == 1) || (class1 == 3) )
return(strcmpi(name_and_ext1, name_and_ext2));
else if( class = 2)
{
if(
non_ws_ext_ret= strcmpi(
file_ext_of_name_and_ext(name_and_ext1),
file_ext_of_name_and_ext(name_and_ext1)
)
!= 0
)return(non_ws_ext_ret);
else return(strcmpi(name_and_ext1, name_and_ext2));
}
}
/*
For primary Wordstar file,
sort by case independent filename and extension.
For backup files, sort by filename
For non-wordstar files,
sort by case independent extension,
and then by case independent filename
*/
int find_priority_class(char *arg)
{
char *file_ext = file_ext_of_name_and_ext(arg);
if(is_not_wordstar_file_extension(file_ext))
return(2);
else if( strcmpi(file_ext, "bak")== 0) return(3);
else return(1);
}
/*
returns 1 for a primary wordstar file,
3 for a backup,
and 2 for something which is not a wordstar document
*/
/*-----*/

int make_a_table_of_old_files(
char *dir_name,
struct
file_table_type
table_of_old_files[]
)
{
int i, num_files;
struct dirent *dir_entry;
struct stat *stat_buffer;
DIR *dr = opendir(dir_name);
i=0;
while( dir_entry = readdir(dr) != NULL)
if(dir_entry -> d_type == DT_REG)
{
strcpy(table_of_old_files[i]->old_f_spec,
dir_entry->d_name);
stat(dir_entry->d_name, stat_buffer);
file_name_and_ext_of_filespec(
table_of_old_files[i]->old_filename_ext,
dir_entry->d_name
);
html_filespec_from_old_filespec(
table_of_old_files[i]->new_f_spec,
dir_entry->d_name
);
build_initial_title_line(
table_of_old_files[i]->new_title,
table_of_old_files[i]->old_filename_ext,
stat_buffer->st_size,
stat_buffer->st_mtime
);
i++;
}
num_files=i;
closedir(dr);
return(num_files)
}
/*
read in all the entries for this directory,
and put those that are ordinary files
into an array of directory entries,
together with their time/date stamps and sizes,
and original and new names,
and return the number of such files
*/
/*-----*/

```

```

add_titles_to_table(
    char *dir_name,
    struct
        file_table_type
        table_of_old_files[],
    int num_files
)
{
    char *comma_ptr, *title_file_title, *title_file_fn_ext;
    char new_line[200], titles_pf_file_spec[256];
    FILE *fp;
    qsort(
        table_of_old_files,
        num_files,
        sizeof(struct file_table_type),
        strcmpi
    );
    /*get in the TITLES.PF file*/
    strcpy(titles_pf_file_spec, dir_name);
    strcat(titles_pf_file_spec, "/TITLES.PF");
    fp=fopen(titles_pf_file_spec, "r");
    if(fgets(new_line, 200, fp) == NULL) break;
    else for(;;)
    {
        if(fgets(new_line, 200, fp) == NULL) break;
        else
        {
            /*and merge each title to the appropriate file record*/
            if(comma_ptr=strchr(new_line, ',') != NULL)
            {
                title_file_title=comma_ptr+1;
                title_file_fn_ext=new_line;
                *comma_ptr='\0';
                if(
                    address_of_match =
                        bsearch(
                            title_file_fn_ext,
                            table_of_old_files,
                            num_files,
                            sizeof(struct file_table_type),
                            strcmpi
                        )
                    != NULL)
                    strcat(
                        table_of_old_files[
                            record_address_to_index(
                                address_of_match,
                                table_of_old_files,
                                sizeof(struct file_table_type)
                            )-> new_title,
                            title_file_title
                        );
            }
        }
    }
    fclose(fp);
}
/*
ancestral MS-DOS only supported 8-character file names,
and 3-character file type extensions. Like many other
contemporary applications, additional information
was placed in a titles file, maintained by the
auxiliary application, Profinder. For purposes
of matching up records, ignore case, which is
not significant in MS-DOS
*/
/*-----*/
record_address_to_index(
    address_of_record,
    address_of_array,
    size_of_record
)
{
    return(address_of_record-address_of_array)/size_of_record;
}

```

```

/*-----*/
build_initial_title_line(
    char *new_title,
    char *old_filename_ext,
    off_t st_size,
    struct timespec st_mtim
)
{
    char date_str[32];
    ctime_r(st_mtim, &date_str);
    sprintf(new_title, "%s %s %u ", old_filename_ext, &date_str, st_size );
}
/*-----*/
char *file_name_and_ext_of_filespec(char *name_and_ext, char *filespec)
{
    char *slash_loc;
    if(filespec == NULL) return(NULL);
    else if(filespec[0] == '\0') return(NULL);
    else if(filespec[strlen(filespec)-1] == '/') return(NULL);
    else if(
        slash_loc = strrchr(filespec, '/')
        != NULL
    ) return(slash_loc+1);
    else return(filespec);
}
/*
If the argument is NULL, or the string is empty, return NULL
If the last character is a slash, return NULL.
Otherwise return whatever is after the last slash.
If no slash found, return the whole string.
*/
/*-----*/
char *file_ext_of_name_and_ext(char *name_and_ext)
{
    char *last_period_loc;
    int chars_after_period;
    if(name_and_ext == NULL) return(NULL);
    else if(name_and_ext[0] == '\0') return(NULL);
    else
    {
        last_period_loc = strrchr(name_and_ext, '.');
        chars_after_period=
            name_and_ext
            +strlen(name_and_ext)
            -last_period_loc;
        if(last_period_loc == name_and_ext)return(NULL);
        else if (
            (chars_after_period <= 4)
            &&
            (chars_after_period >= 1)
        ) return(last_period_loc);
        else return(NULL);
    }
}
/*
If the argument is NULL, or the string is empty, return NULL
Otherwise find the last period,
which is not the first character,
nor more than four characters from the end,
and point to whatever is after it.
If no such period found,
or it is the last character,
return NULL
*/
/*-----*/
char *html_filespec_from_old_filespec(
    char *new_filespec,
    char *old_filespec
)
{
    char name_and_ext[32]
    char *file_extension;
    strcpy(new_filespec, old_filespec);
    if(
        file_extension= file_ext_of_name_and_ext(
            file_name_and_ext_of_filespec(
                &name_and_ext, new_filespec
            )
        ) != NULL
    )
    {
        *(file_extension-1)='-';
        strcat(new_filespec, ".html");
    }
    else strcat(new_filespec, "____.html");
    return(new_filespec);
}
/*
if the file extension is not empty,
turn the preceding period into an underscore, and
append ".html"
If it is empty, append "____.html"
*/

```

```

/*-----*/
int is_not_wordstar_file_extension(char *file_extension)
{
    char c_file_extension[10];
    strcpy(c_file_extension, file_extension);
   strupr(&c_file_extension);
    if(
        (strcmp(&c_file_extension, "PF") == 0)
        ||
        (strcmp(&c_file_extension, "GIF") == 0)
        ||
        (strcmp(&c_file_extension, "JPG") == 0)
        ||
        (strcmp(&c_file_extension, "TXT") == 0)
        ||
        (strcmp(&c_file_extension, "BAT") == 0)
        ||
        (strcmp(&c_file_extension, "ASM") == 0)
        ||
        (strcmp(&c_file_extension, "BAS") == 0)
        ||
        (strcmp(&c_file_extension, "C") == 0)
        ||
        (strcmp(&c_file_extension, "COM") == 0)
        ||
        (strcmp(&c_file_extension, "EXE") == 0)
        ||
        (strcmp(&c_file_extension, "OVR") == 0)
        ||
        (strcmp(&c_file_extension, "INS") == 0)
        ||
        (strcmp(&c_file_extension, "WKS") == 0)
        ||
        (strcmp(&c_file_extension, "ZIP") == 0)
    )return(1);
    else return(0);
}
/*

```

Distinguish file types which are not going to be converted, ignoring capitalization (file case is not significant in MS-DOS)

There ia no uniform convention for what extension a Wordstar file has. The program did not enforce or auto-suggest one, the default was no extension at all, and while the manual suggested the use of DOC, this was overtaken by Microsoft Word's superior claims. I myself used WS3, WS5, and WS7. Thus we can only identify things which are not Wordstar files, and err on the side of nondestructively generating a certain amount of garbage.

Parenthetically, WordStar had a non-dcument mode for editing programs and scripts. It generated straight ASCII, except for the file being padded out with Cntrl-Zs at the end. It was convenient if you already knew how to use WordStar, but it did not have the features of modern programmers' editors, such as syntax recognition.

BUT the new table of contents file should nonetheless list files if they have titles
 */

```

/*-----*/
process_wordstar_files(
    char *dir_name,
    struct
        file_table_type
        table_of_old_files[],
    int num_files
)
{
    int f_tc_out;
    int i;
    char titles_pf_file_spec[256];
    strcpy(&titles_pf_file_spec, dir_name);
    strcat(&titles_pf_file_spec, "/TITLES_PF.html");

    f_tc_out = open(
        &titles_pf_file_spec,
        O_WRONLY|O_CREAT|O_EXCL|O_BINARY, S_IREAD|S_IWRITE
    );
    if( f_tc_out == -1) return(-2);
    /*create a header */
    write_s(f_tc_out, "<HTML>\r\n<HEAD>\r\n<TITLE>");
    write_html_s(f_tc_out, dir_name);
    write_s(f_tc_out, "</TITLE>\r\n</HEAD>\r\n<BODY>\r\n");
    for(i=0; i < num_files; i++)
    {
        if(
            is_not_wordstar_file_extension(
                file_ext_of_name_and_ext(
                    table_of_old_files[i]
                    ->old_filename_ext
                )
            )
        )
            make_table_of_contents_entry(
                f_tc_out,
                table_of_old_files[i]
                ->old_f_spec,
                table_of_old_files[i]
                ->new_title
            );
        else
        {
            make_table_of_contents_entry(
                f_tc_out,
                table_of_old_files[i]
                ->new_f_spec,
                table_of_old_files[i]
                ->new_title
            );
            translate_wordstar_file_to_html(
                table_of_old_files[i]
                ->old_f_spec,
                table_of_old_files[i]
                ->new_f_spec,
                table_of_old_files[i]
                ->new_title
            );
        }
    }
    /* now add the tail and close*/
    write_s(f_tc_out, "\r\n</BODY>\r\n</HTML>");
    close(f_tc_out);
    return(1);
}
/*
    Go through the table,
    and convert every wordstar file,
    and create an entry for every file in titles_pf.html
*/

```

```

/*-----*/
make_table_of_contents_entry( int f_tc_out,
                             char *file_linked_to,
                             char *title
                             )
{
    new_fn_ext[32]
    file_name_and_ext_of_filespec(&new_fn_ext, file_linked_to);
    write_s(f_tc_out, "\r\n<P><A href=\"");
    write_s(f_tc_out, &new_fn_ext);
    write_s(f_tc_out, "\>\r\n");
    write_html_s(f_tc_out, title);
    write_s(f_tc_out, "</A><P>");
}
/*-----*/
translate_wordstar_file_to_html(
    char *wordstar_file,
    char *html_file,
    char *title
    )
{
    int bytes_read, f_in, f_out;
    int start_point_offset = 80000;
    unsigned char input_buffer[1000000];
    f_in = open(wordstar_file, O_RDONLY|O_BINARY);
    if(f_in == -1) return(-1);
    f_out = open(
        html_file,
        O_WRONLY|O_CREAT|O_EXCL|O_BINARY, S_IREAD|S_IWRITE
        );
    if( f_out == -1) return(-2);
    /*create a header */
    write_s(f_out, "<HTML>\r\n<HEAD>\r\n<TITLE>");
    write_html_s(f_out, title);
    write_s(f_out, "</TITLE>\r\n</HEAD>\r\n<BODY>\r\n");
    /* read the file into a buffer. Leave 80K vacant up front */
    bytes_read = read(
        f_in,
        (input_buffer+start_point_offset),
        (sizeof(input_buffer)-start_point_offset)
        );
    if(bytes_read <= 1) return(-3);
    parse_buffer(
        f_out,
        input_buffer,
        sizeof(input_buffer),
        start_point_offset,
        bytes_read
        );
    /* now add the tail and close*/
    write_s(f_out, "\r\n</BODY>\r\n</HTML>");
    close(f_in);
    close(f_out);
    return(1);
}

```

```

/*-----*/
parse_buffer( int f_out,
              unsigned char input_buffer[],
              int size_of_buffer,
              int start_point_offset,
              int number_of_bytes
              )
{
    char curr_c, c_s, next_c, c_after_next;

    int index_to_buffer;

    manage_flip_flop_char_attr(f_out, curr_c, "initialize");
    set_curr_column("hard");
    /* strip off Cntrl-Z's from the end */
    for(
        index_to_buffer=start_point_offset+number_of_bytes;
        index_to_buffer >= start_point_offset;
        index_to_buffer--
        )
    {
        if(input_buffer[index_to_buffer] != '\x1d')
        {
            number_of_bytes= index_to_buffer-start_point_offset;
            break
        }
    }
}
/*
Wordstar was originally a CP/M program,
and was mechanically translated to MS-DOS v.1,
using a "Cross-Assembler." Consequently,
Wordstar 3 used the old record-oriented file system,
in which files were allocated, read, and written,
in chunks of 128? bytes. Hence, Cntrl-Z was used
to pad the file out to the nearest whole record length.

Cross assemblers should doubtless be explained for the record.
The Intel 8086/8088-MS--DOS v,1 architecture was not a
superset of th Zilog Z-80--CP/M architecture,
but it was a superset of something which had a
one-to-one correspondence with the Zilog--CP/M
architecture. The assembly-language source code
for a CP/M program could be run through a cross-assembler
which would apply a series of
regular transformations, producing MS-DOS
assembly-language source code, which required
only a moderate amount of rewriting to be correct.
Thus the IBM PC began life with a large variety
of available commercial software.
*/

/*and now do the conversion */
for(
    index_to_buffer=start_point_offset;
    index_to_buffer <= start_point_offset+number_of_bytes ;
    index_to_buffer++
    )
/*
Do over number_of_bytes from Start_point_offset,
but additional material may be patched in at the front,
and used up,
which is why there is an offset.
*/
{
    curr_c=input_buffer[index_to_buffer];
    c_s = curr_c & '\x7f';
    next_c=input_buffer[index_to_buffer+1];
    c_after_next=input_buffer[index_to_buffer+2];
}
/*case 1*/
/*disambiguate periods*/
if(curr_c == ".")
{
    if(set_curr_column("report") == -1) /* if it is a dot command*/
        index_to_buffer=
            parse_dot_command(
                f_out, input_buffer. index_to_buffer);
    else /*if it is only a period*/
    {
        send_char_to_html(f_out, curr_c);
        set_curr_column("incr");
    }
}
/*
WordStar "dot-commands" are user visible,
and are fully described in the user manual.
*/
/*case 2*/
else if((curr_c == ' ') || (curr_c == '\xa0')) /*SPACES*/
    index_to_buffer=
        parse_spaces(
            f_out, input_buffer. index_to_buffer);
}

```

```

/*case 3*/
else if( isprint(curr_c))          /*PRINTABLE*/
{
    send_char_to_html(f_out, curr_c);
    set_curr_column("incr");
}
/*case 3a*/
else if( isprint(c_s))            /* PRINTABLE WITH HIGH BIT */
{
    send_char_to_html(f_out, c_s);
    set_curr_column("incr");
}
/*case 4*/
else if( curr_c == '\r')          /*Cntrl-M HARD CARRIAGE RETURN*/
{
    write_cr_according_to_ruler(f_out);
    if(next_c == '\n') index_to_buffer++;
    set_curr_column("hard");
    manage_flip_flop_char_attr(f_out, curr_c, "reset");
    /* makes the assumption that special
       attributes dont persist across a
       paragraph break */
}
/*case 5*/
else if( curr_curr_c == '\x8d')   /*SOFT CARRIAGE RETURN*/
{
    set_curr_column("soft");
    write_s(f_out, "\r\n");
    if(next_c == '\n') index_to_buffer++;
}
/*case 6*/
else if( curr_c == '\x0c')        /*Cntrl-L-- Page Break*/
{
    write_cr_according_to_ruler(f_out);
    if(next_c == '\n') index_to_buffer++;
    set_curr_column("hard");
    manage_flip_flop_char_attr(f_out, curr_c, "reset");
    /* makes the assumption that special
       attributes dont persist across a
       page break */
}
/*case 7*/
/*
In Wordstar, a number of Cntrl-bytes toggle the print
attribute of text. IN HTML, the attributes have different tags
to turn them off and on. So we have to keep track of the
state of the Wordstar document,
and select the appropriate HTML tag.
*/
else if(
    (curr_c == '\x02' /* Cntrl-B Bold */)
    || (curr_c == '\x04' /* Cntrl-D Double */)
    || (curr_c == '\x0b' /* Cntrl-K Indexed */)
    || (curr_c == '\x13' /* Cntrl-S Underline */)
    || (curr_c == '\x14' /* Cntrl-T SuperScript */)
    || (curr_c == '\x16' /* Cntrl-V Subscript */)
    || (curr_c == '\x18' /* Cntrl-X Strikeout */)
    || (curr_c == '\x19' /* Cntrl-Y Italics */)
) manage_flip_flop_char_attr(f_out, curr_c, "toggle");

/*
case 8
if it is an extended character,
that is: [hex 1b][ext_char][hex 1c]
*/
else if( curr_c == '\x1b')
{
    if(c_after_next == '\x1c')
    {
        set_curr_column("incr");
        send_ext_char_to_html(f_out, next_c);
        index_to_buffer+=2;
    }
    else write_hex_code(f_out, '\x1b');
}
/*
case 9
Cntrl-H Overprint
A common ad-hoc method of forming extended characters
*/
else if( curr_c == '\x08');
    write_s(
        f_out,
        "[Overprint]"
    );
}

/*
case 10
Cntrl-I Tab
PROBLEM
*/
else if( curr_c == '\x09')
{
    set_curr_column("incr");
    write_s(f_out, "[TAB]");
}
/*
case 11
Unpaired Line Feed
*/
else if( curr_c == '\x0a')
    write_s(f_out, "<P>[LINE FEED]<P>");

/* case 12
Custom Print Controls--
only decodable by reference to printer manual
and dot-X commands, wordstar configuration settings
*/
else if( curr_c == '\x05')
    write_s(
        f_out,
        "<P>/r/n[Cntrl-E -- Custom Print Control]/r/n<P>"
    );
else if( curr_c == '\x11')
    write_s(
        f_out,
        "<P>/r/n[Cntrl-Q -- Custom Print Control]/r/n<P>"
    );
else if( curr_c == '\x12' )
    write_s(
        f_out,
        "<P>/r/n[Cntrl-R -- Custom Print Control]/r/n<P>"
    );
else if( curr_c == '\x17')
    write_s(
        f_out,
        "<P>/r/n[Cntrl-W -- Custom Print Control]/r/n<P>"
    );
/*case 13 Reserved Characters*/
else if( curr_c == '\x10')
    write_s(
        f_out,
        "<P>/r/n[Cntrl-P -- Reserved Char]/r/n<P>"
    );
else if( curr_c == '\x15')
    write_s(
        f_out,
        "<P>/r/n[Cntrl-U -- Reserved Char]/r/n<P>"
    );
}
/*
case 14
Cntrl-At-- Obsolete--fix Print Position
*/
else if( curr_c == '\x00')
    write_s(
        f_out,
        "[Cntrl-At -- fix Print Position ]/r/n<P>"
    );
/* case 15
old font change mechnism,
superseded by symetric sequence 15h,
and should have been automatically striped out
if edited under version 5 and subsequent
*/
else if( curr_c == '\x01')
    write_s(
        f_out,
        "<P>/r/n[Cntrl-A -- Altrenative Font]/r/n<P>"
    );
else if( curr_c == '\x0e')
    write_s(
        f_out,
        "<P>/r/n[Cntrl-N -- Return to Normal Font]/r/n<P>"
    );
/*
case 16
Cntrl-C -- Obsolete--
Pause For User Response During Print.
In other words, the user was expected to look at
what the printer had printed off,
insert a special daisywheel print element
or change a printer ribbon,
and resume.
*/
else if( curr_c == '\x03')
    write_s(
        f_out,
        "<P>/r/n[Cntrl-C -- Meaning Unrecoverable]/r/n<P>"
    );
}

```

```

/* case 17
Daisywheel Phantom characters
Meaningful only if you know which daisywheel was being used.
*/
else if( curr_c == '\x06' /* Cntrl-F */)
    write_s(
        f_out,
        "[Daisywheel Phantom Space]>"
    );
else if( curr_c == '\x07' /* Cntrl-G */)
    write_s(
        f_out,
        "[Daisywheel Phantom Rubout]"
    );
/*case 18*/
else if( curr_c == '\x0f' /* Cntrl-O -- Binding space*/)
    ;

/*
case 19
Cntrl-Z -- End of File used as padding at the end
of the file to bring it up to a round length
under C/PM and MS-DOS v.1. However, if it
occurs in the middle of the file, print the bytecode.
*/
else if( curr_c == '\x1a')
    write_s(f_out, "[hex code 1a]");
/*
case 20
Symetric Sequences
*/
else if( curr_c == '\xd1' /* Cntrl-Right-Bracket -- */)
    index_to_buffer=
    parse_symetric_sequence(
        f_out, input_buffer. index_to_buffer);
/*
case 21
Soft Hyphen
*/
else if( curr_c == '\x1e' /* Cntrl-Caret -- */)
    ;
else if( curr_c == '\x1f' /* Cntrl-Under_Hyphen -- */)
    ;
/*
case 22
Any Other Byte
put out the hex code and let the user figure out what it means
*/
else write_hex_code(f_out, curr_c);
}
}

```

```

/*-----*/
/*
In Wordstar, a number of Cntrl-bytes toggle the print
attribute of text. IN HTML, the attributes have different tags
to turn them off and on. So we have to keep track of the
state of the Wordstar document,
and select the appropriate HTML tag.
*/
manage_flip_flop_char_attr(int f_out, char curr_c, char *mode )
{
    int static is_attr[8], save_attr[8];

    int i;

    /* define the names of the text state counters */
    int bold_attr = 0;
    int doub_attr = 1;
    int undl_attr = 2;
    int sup_attr = 3;
    int sub_attr = 4;
    int ital_attr = 5;
    int strikeout_attr = 6;
    int indexed_attr =7;

    if (strcmp(mode, "toggle") == 0)
    {
        if( curr_c == '\x02' /*Cntrl-B*/)
            flip_attr(&is_attr, bold_attr, f_out, "<B>", "</B>", "toggle");
        else if( curr_c == '\x04' /*Cntrl-D*/)
            flip_attr(&is_attr, doub_attr, f_out, "<B>", "</B>", "toggle");
        else if( curr_c == '\x0b' /*Cntrl-D*/)
            flip_attr(&is_attr, indexed_attr, f_out, "<?>", "</?>",
"toggle");
        else if( curr_c == '\x13' /*Cntrl-S*/)
            flip_attr(&is_attr, undl_attr, f_out, "<U>", "</U>", "toggle");
        else if( curr_c == '\x14' /*Cntrl-T*/)
            flip_attr(&is_attr, sup_attr, f_out, "<SUP>", "</SUB>",
"toggle");
        else if( curr_c == '\x16' /*Cntrl-V*/)
            flip_attr(&is_attr, sub_attr, f_out, "<SUB>", "</SUB>",
"toggle");
        else if( curr_c == '\x18' /*Cntrl-X*/)
            flip_attr(&is_attr,
                strikeout_attr, f_out, "<S>", "</S>", "toggle");
        else if( curr_c == '\x19' /*Cntrl-Y*/)
            flip_attr(&is_attr, ital_attr, f_out, "<EM>", "</EM>", "toggle");
    }

    else if (strcmp(mode, "restore") == 0)
    {
        if( is_attr[bold_attr] != save_attr[bold_attr] /*Cntrl-B*/)
            flip_attr(&is_attr, bold_attr, f_out, "<B>", "</B>", "toggle");
        if( is_attr[doub_attr] != save_attr[doub_attr] /*Cntrl-D*/)
            flip_attr(&is_attr, doub_attr, f_out, "<B>", "</B>", "toggle");
        if( is_attr[indexed_attr] != save_attr[indexed_attr] /*Cntrl-D*/)
            flip_attr(&is_attr, indexed_attr, f_out, "<?>", "</?>",
"toggle");
        if( is_attr[undl_attr] != save_attr[undl_attr] /*Cntrl-S*/)
            flip_attr(&is_attr, undl_attr, f_out, "<U>", "</U>", "toggle");
        if( is_attr[sup_attr] != save_attr[sup_attr] /*Cntrl-T*/)
            flip_attr(&is_attr, sup_attr, f_out, "<SUP>", "</SUB>",
"toggle");
        if( is_attr[sub_attr] != save_attr[sub_attr] /*Cntrl-V*/)
            flip_attr(&is_attr, sub_attr, f_out, "<SUB>", "</SUB>",
"toggle");
        if( is_attr[strikeout_attr] != save_attr[strikeout_attr] /*Cntrl-X*/)
            flip_attr(&is_attr,
                strikeout_attr, f_out, "<S>", "</S>", "toggle");
        if( is_attr[ital_attr] != save_attr[ital_attr] /*Cntrl-Y*/)
            flip_attr(&is_attr, ital_attr, f_out, "<EM>", "</EM>", "toggle");
    }

    else if(strcmp(mode,"initialize" ) == 0)
    {
        int i;
        for(i=0;i < 8; i++)
        {
            is_attr[i] = 0;
        }
    }
    else if(strcmp(mode,"save" ) == 0)
    {
        int i;
        for(i=0;i < 8; i++)
        {
            save_attr[i] = is_attr[i];
        }
    }
}

```

```

else if(strcmp(mode,"reset") == 0)
{
    flip_attr(&is_attr, bold_attr, f_out, "<B>", "</B>", "reset");
    flip_attr(&is_attr, doub_attr, f_out, "<B>", "</B>", "reset");
    flip_attr(&is_attr, indexed_attr, f_out, "<?>", "</?>", "reset");
    flip_attr(&is_attr, undl_attr, f_out, "<U>", "</U>", "reset");
    flip_attr(&is_attr, sup_attr, f_out, "<SUP>", "</SUB>", "reset");
    flip_attr(&is_attr, sub_attr, f_out, "<SUB>", "</SUB>", "reset");
    flip_attr(&is_attr, strikeout_attr, f_out, "<S>", "</S>", "reset");
    flip_attr(&is_attr, ital_attr, f_out, "<EM>", "</EM>", "reset");
}
}
/*-----*/
flip_attr(int *is_attr, int this_attr, int f_out, char *beg_str, char
*end_str, char *mode)
{
    if(is_attr[this_attr])
    {
        is_attr[this_attr] = 0;
        write_s(f_out, end_str);
    }
    else if(strcmp(mode, "reset") == 0)
    {
        is_attr[this_attr] = 1;
        write_s(f_out, beg_str);
    }
}
/* rewrite to localize this as far as possible. a; the main program needs
to know is that the following characters are flip attributes, and that
something appropriate is being put in the output stream. there are three
possible cases, initialize, reset, and encounter a bytecode*/
/*-----*/
int set_curr_column(char *mode)
{
    static int curr_coll,save_coll;
    if(strcmp(mode,"hard") == 0) curr_coll = -1;
    else if(strcmp(mode,"soft") == 0) curr_coll = 0;
    else if(strcmp(mode,"incr") == 0)
    {
        if(curr_coll == -1) curr_coll = 1;
        else curr_coll++;
    }
    else if(strcmp(mode,"save") == 0) save_coll = curr_coll;
    else if(strcmp(mode,"restore") == 0) curr_coll = save_coll;
    else if(strcmp(mode,"report") == 0);
    return(curr_coll);
}
/*
returns -1 after a hard carriage return,
0 after a soft carriage return,
otherwise the number of printable characters
now read into the line
*/
/*-----*/
write_hex_code(int f_out, char c)
{
    char c1, c2, c1c, c2c;
    char[16] char_rep= "[hex code bb]";
    c2=c%16; /*modulo off the high order nybble*/
    c1=(c-c2)/16; /* and subtract to obtain the remainder*/
    if(c1 <= 9) c1c = c1 + '0';
        else c1c = c1 + 'A'-10;
    if(c2 <= 9) c2c = c2 + '0';
        else c2c = c2 + 'A'-10;
    char_rep[11] = c1c;
    char_rep[12] = c2c;
    write_s(f_out, &char_rep);
}
/*-----*/
send_char_to_html(int f_out, char c)
{
    char send_str[2];
    if(c == "\"") write_s(f_out, "&quot;");
    else if(c == "&") write_s(f_out, "&amp;");
    else if(c == "<") write_s(f_out, "&lt;");
    else if(c == ">") write_s(f_out, "&gt;");
    else
    {
        send_str[0] = c;
        send_str[1] = '\0';
        write_s(f_out, &send_str);
    }
}
/*provides aliasing for source characters which are reserved in HTML*/
/*-----*/
write_html_s(int f_out, char *s)
{
    char *s1;
    for(s1=s; *s1 != '\0'; s1++) send_char_to_html(f_out, *s1);
}
/*-----*/
write_s(int f_out, char *s)
{
    write(f_out, s, strlen(s));
}
/* the general output point*/
/*-----*/
send_ext_char_to_html(int f_out, char c)
{
    switch(c)
    {
        case '\x00': write_s(f_out, " "); break;/*blank*/
        case '\x01': write_s(f_out, "[happy_face]"); break;/*happy face*/
        case '\x02': write_s(f_out, "[inv_happy_face]"); break;
        /*inverse happy face*/
        case '\x03': write_s(f_out, "&heartsuit;"); break;/*heart*/
        case '\x04': write_s(f_out, "&diamondsuit;"); break;/*diamond*/
        case '\x05': write_s(f_out, "&clubsuit;"); break;/*club*/
        case '\x06': write_s(f_out, "&spadesuit;"); break;/*spade*/
        case '\x07': write_s(f_out, "&bullet;"); break;/*bullet*/
        case '\x08': write_s(f_out, "[inv_bullet]"); break;/*inverse bullet*/
        case '\x09': write_s(f_out, "&xcirc;"); break;/*circle*/
        case '\x0a': write_s(f_out, "[inv_circle]"); break;/*inverse circle*/
        case '\x0b': write_s(f_out, "&male;"); break;/*male sign*/
        case '\x0c': write_s(f_out, "&female;"); break;/*female sign*/
        case '\x0d': write_s(f_out, "&sung;"); break;/*single note*/
        case '\x0e': write_s(f_out, "&sung&sung;"); break;/*double note*/
        case '\x0f': write_s(f_out, "[sun]"); break;/*sun*/

        case '\x10': write_s(f_out, "&rtrif;"); break;/*right triangle*/
        case '\x11': write_s(f_out, "&ltrif;"); break;/*left triangle*/
        case '\x12': write_s(f_out, "&updownarrow;"); break;/*up-down arrow*/
        case '\x13': write_s(f_out, "!!"); break;/*double exclamation*/
        case '\x14': write_s(f_out, "&para;"); break;/*paragraph sign*/
        case '\x15': write_s(f_out, "&sect;"); break;/*section sign*/
        case '\x16': write_s(f_out, "&hybull;"); break;/*rectangular bullet*/
        case '\x17': write_s(f_out, "&UpArrowBar;&DownArrowBar;"); break;
        /*up/down to line*/
        case '\x18': write_s(f_out, "&uparrow;"); break;/*up arrow*/
        case '\x19': write_s(f_out, "&downarrow;"); break;/*down arrow*/
        case '\x1a': write_s(f_out, "&rightrightarrow;"); break;/*right arrow*/
        case '\x1b': write_s(f_out, "&leftleftarrow;"); break;/*left arrow*/
        case '\x1c': write_s(f_out, "&boxur;"); break;/*lower left box*/
        case '\x1d': write_s(f_out, "&leftrightarrow;"); break;
        /*left-right arrow*/
        case '\x1e': write_s(f_out, "&blacktriangle;"); break;/*up triangle*/
        case '\x1f': write_s(f_out, "&blacktriangledown;"); break;
        /*down triangle*/

        case '\x80': write_s(f_out, "&Ccedil;"); break;
        case '\x81': write_s(f_out, "&uuml;"); break;
        case '\x82': write_s(f_out, "&acute;"); break;
        case '\x83': write_s(f_out, "&acirc;"); break;
        case '\x84': write_s(f_out, "&auml;"); break;
        case '\x85': write_s(f_out, "&agrave;"); break;
        case '\x86': write_s(f_out, "&aring;"); break;
        case '\x87': write_s(f_out, "&ccedil;"); break;

        case '\x88': write_s(f_out, "&ecirc;"); break;
        case '\x89': write_s(f_out, "&euml;"); break;
        case '\x8a': write_s(f_out, "&egrave;"); break;
        case '\x8b': write_s(f_out, "&iuml;"); break;
        case '\x8c': write_s(f_out, "&icirc;"); break;
        case '\x8d': write_s(f_out, "&igrave;"); break;
        case '\x8e': write_s(f_out, "&ouml;"); break;
        case '\x8f': write_s(f_out, "&aring;"); break;

        case '\x90': write_s(f_out, "&Eacute;"); break;
        case '\x91': write_s(f_out, "&aelig;"); break;
        case '\x92': write_s(f_out, "&Aelig;"); break;

        case '\x93': write_s(f_out, "&ocirc;"); break;
        case '\x94': write_s(f_out, "&ouml;"); break;
        case '\x95': write_s(f_out, "&ograve;"); break;
        case '\x96': write_s(f_out, "&ucirc;"); break;
        case '\x97': write_s(f_out, "&ugrave;"); break;
        case '\x98': write_s(f_out, "&yuml;"); break;
        case '\x99': write_s(f_out, "&Ouml;"); break;
        case '\x9a': write_s(f_out, "&Uuml;"); break;

        case '\x9b': write_s(f_out, "&cent;"); break;
        case '\x9c': write_s(f_out, "&pound;"); break;
        case '\x9d': write_s(f_out, "&yen;"); break;
        case '\x9e': write_s(f_out, "[Pt]"); break;/*Pt sign*/

        case '\x9f': write_s(f_out, "[funct]"); break;/*the function mark*/

        case '\xa0': write_s(f_out, "&acute;"); break;
        case '\xa1': write_s(f_out, "&iacute;"); break;
        case '\xa2': write_s(f_out, "&oacute;"); break;
        case '\xa3': write_s(f_out, "&uacute;"); break;
        case '\xa4': write_s(f_out, "&ntilde;"); break;
        case '\xa5': write_s(f_out, "&Ntilde;"); break;
        case '\xa6': write_s(f_out, "&amacr;"); break;
        case '\xa7': write_s(f_out, "&omacr;"); break;
    }
}

```

```

case '\xa8': write_s(f_out, "&iquest;"); break;
case '\xa9': write_s(f_out, "&boxdr;"); break; /*upper left box*/
case '\xaa': write_s(f_out, "&boxdl;"); break; /*upper right box*/
case '\xab': write_s(f_out, "&frac12;"); break;
case '\xac': write_s(f_out, "&frac14;"); break;
case '\xad': write_s(f_out, "!"); break; /*opening exclamation point*/
case '\xae': write_s(f_out, "&laquo;"); break; /*<< mark*/
case '\xaf': write_s(f_out, "&raquo;"); break; /*>> mark*/
case '\xb0': write_s(f_out, "&blk14;"); break; /*light block*/
case '\xb1': write_s(f_out, "&blk12;"); break; /*medium block*/
case '\xb2': write_s(f_out, "&blk34;"); break; /*dark block*/

case '\xb3': write_s(f_out, "&boxv;"); break; /*single vertical*/
case '\xb4': write_s(f_out, "&boxv1;"); break; /*single right junction*/ }
case '\xb5': write_s(f_out, "&boxvL;"); break; /**2 to 1 right junction*/ /*
case '\xb6': write_s(f_out, "&boxv1;"); break; /**1 to 2 right junction*/
case '\xb7': write_s(f_out, "&boxDL;"); break; /**1 to 2 upper right*/
case '\xb8': write_s(f_out, "&boxDL;"); break; /**2 to 1 upper right*/
case '\xb9': write_s(f_out, "&boxVL;"); break; /*double right junction*/
case '\xba': write_s(f_out, "&boxv;"); break; /*double vertical*/
case '\xbb': write_s(f_out, "&boxDL;"); break; /*double upper right*/
case '\xbc': write_s(f_out, "&boxUL;"); break; /*double lower right*/
case '\xbd': write_s(f_out, "&boxUL;"); break; /**1 to 2 lower right*/
case '\xbe': write_s(f_out, "&boxUL;"); break; /**2 to 1 lower right*/
case '\xbf': write_s(f_out, "&boxdl;"); break; /*single upper right*/
case '\xc0': write_s(f_out, "&boxur;"); break; /*single lower left*/
case '\xc1': write_s(f_out, "&boxhu;"); break; /*single lower junction*/
case '\xc2': write_s(f_out, "&boxhd;"); break; /*single upper junction*/
case '\xc3': write_s(f_out, "&boxvr;"); break; /*single left junction*/
case '\xc4': write_s(f_out, "&boxh;"); break; /*single horizontal*/
case '\xc5': write_s(f_out, "&boxvh;"); break; /*single intersection*/
case '\xc6': write_s(f_out, "&boxvR;"); break; /**2 to 1 left junction*/
case '\xc7': write_s(f_out, "&boxvR;"); break; /**1 to 2 left junction*/
case '\xc8': write_s(f_out, "&boxUR;"); break; /*double lower left*/
case '\xc9': write_s(f_out, "&boxDR;"); break; /*double upper left*/
case '\xca': write_s(f_out, "&boxHU;"); break; /*double lower junction*/
case '\xcb': write_s(f_out, "&boxHD;"); break; /*double upper junction*/
case '\xcc': write_s(f_out, "&boxVR;"); break; /*double left junction*/
case '\xcd': write_s(f_out, "&boxH;"); break; /*double horizontal*/
case '\xce': write_s(f_out, "&boxVH;"); break; /*double intersection*/
case '\xcf': write_s(f_out, "&boxHu;"); break; /**1 to 2 lower junction*/
case '\xd0': write_s(f_out, "&boxHu;"); break; /**2 to 1 lower junction*/
case '\xd1': write_s(f_out, "&boxHd;"); break; /**1 to 2 upper junction*/
case '\xd2': write_s(f_out, "&boxHd;"); break; /**2 to 1 upper junction*/
case '\xd3': write_s(f_out, "&boxUr;"); break; /**1 to 2 lower left*/
case '\xd4': write_s(f_out, "&boxUr;"); break; /**2 to 1 lower left*/
case '\xd5': write_s(f_out, "&boxDr;"); break; /**2 to 1 upper left*/
case '\xd6': write_s(f_out, "&boxDr;"); break; /**1 to 2 upper left*/
case '\xd7': write_s(f_out, "&boxVh;"); break; /**2 to 1 intersection*/
case '\xd8': write_s(f_out, "&boxVh;"); break; /**1 to 2 intersection*/
case '\xd9': write_s(f_out, "&boxul;"); break; /*single lower right*/
case '\xda': write_s(f_out, "&boxdr;"); break; /*single upper left*/

case '\xdb': write_s(f_out, "&block;"); break; /*inverse space*/
case '\xdc': write_s(f_out, "&lhlbk;"); break; /*lower inverse*/
case '\xdd': write_s(f_out, "&marker;"); break; /*left inverse*/
case '\xde': write_s(f_out, "&marker;"); break; /*right inverse*/
case '\xdf': write_s(f_out, "&uhblk;"); break; /*upper inverse*/

case '\xe0': write_s(f_out, "&alpha;"); break;
case '\xe1': write_s(f_out, "&beta;"); break;
case '\xe2': write_s(f_out, "&Gamma;"); break;
case '\xe3': write_s(f_out, "&pi;"); break;
case '\xe4': write_s(f_out, "&Sigma;"); break;
case '\xe5': write_s(f_out, "&sigma;"); break;
case '\xe6': write_s(f_out, "&mu;"); break;
case '\xe7': write_s(f_out, "&tau;"); break;
case '\xe8': write_s(f_out, "&Phi;"); break;
case '\xe9': write_s(f_out, "&Theta;"); break;
case '\xea': write_s(f_out, "&Omega;"); break;
case '\xeb': write_s(f_out, "&delta;"); break;
case '\xec': write_s(f_out, "[infinity]"); break; /*infinity*/
case '\xed': write_s(f_out, "&straightphi;"); break;
case '\xee': write_s(f_out, "&epsilon;"); break;

case '\xef': write_s(f_out, "&Intersection;"); break;
/*intersection of sets*/
case '\xf0': write_s(f_out, "&Congruent;"); break; /*is identical to*/
case '\xf1': write_s(f_out, "&PlusMinus;"); break;
case '\xf2': write_s(f_out, "&ge;"); break; /*greater than or equal*/
case '\xf3': write_s(f_out, "&leq;"); break; /*less than or equal to*/
case '\xf4': write_s(f_out, "&Integral;"); break; /*top half integral*/
case '\xf5': write_s(f_out, "&Integral;"); break; /*lower half
integral*/
case '\xf6': write_s(f_out, "&divide;"); break; /*divide by*/
case '\xf7': write_s(f_out, "&approx;"); break; /*approximately*/
case '\xf8': write_s(f_out, "&deg;"); break; /*degree sign*/
case '\xf9': write_s(f_out, "[filled_deg]"); break;

```

```

/*filled in degree sign*/
case '\xfa': write_s(f_out, "&sqf;"); break; /*small bullet*/
case '\xfb': write_s(f_out, "[sqrt]"); break; /*square root n*/
case '\xfc': write_s(f_out, "[sup n]"); break; /*superscript n*/
case '\xfd': write_s(f_out, "&sup2;"); break; /*superscript 2*/
case '\xfe': write_s(f_out, "[small box]"); break; /*small box*/
case '\xff': write_s(f_out, " "); break; /*phantom space*/
default:
{
if(isprint(c))send_char_to_html(f_out, c);
else write_hex_code(f_out, c);
break;
}
}

```

Converts extended PC bytecodes into HTML aliased characters. While the HTML alias character set is quite large, it is not entirely a superset of the PC byescodes, and I have been forced to take some liberties.

Sources:
Thom Hogan, The Programmer's PC Sourcebook, Microsoft Press, 1988,
Table 1.22, IBM ASCII Character Set
Character Entity Table,
<https://dev.w3.org/html5/html-author.charref>

```

int parse_symetric_sequence(
    int f_out,
    unsigned char input_buffer[],
    int index_to_first_byte_in_sequence
)
{
    int index_to_last_byte_in_sequence;
    unsigned char opcode, c_1;

    if(
        index_to_last_byte_in_sequence=
        symetric_sequence_is_valid(input_buffer, index_to_buffer)
        ==
        -1)
    {
        write_s(f_out, "[hex code 1d]");
        return(index_to_first_byte_in_sequence);
    }
    else
    {
        opcode = input_buffer[index_to_buffer+3];
        switch (opcode)
        {
            case '\x03':
            case '\x04':
            case '\x05':
            case '\x06':
            {
                /* save and reset toggles and flags */
                manage_flip_flop_char_attr(f_out, c_1, "save");
                manage_flip_flop_char_attr(f_out, c_1, "reset");
                set_curr_column("save");
                set_curr_column("hard");
                bytes_added_to_stream=
                parse_note(
                    f_out,
                    input_buffer,
                    index_to_first_byte_in_sequence
                    index_to_last_byte_in_sequence
                );
                return(
                    index_to_last_byte_in_sequence
                    -bytes_added_to_stream);
            }
            break;
        }
    }
    /*
    HTML </aside>
    encoded in Wordstar format by function parse_note
    The sequence will be of the form:
    <1d><04><00><ff><04><00><1d>
    */
    case '\xff' :
    {
        /*restore toggles and flags */
        manage_flip_flop_char_attr(f_out, c_1, "restore");
        set_curr_column("restore");
        /*output */
        write_s(f_out, "</ASIDE>");
        return(index_to_last_byte_in_sequence);
        break;
    }
    /*
    Converts Wordstar image file tag into HTML image file tag
    */
    case '\x10' :
    {
        /* make the file name into a C string */
        input_buffer[index_to_last_byte_in_sequence-2]='\x00'
        write_s(f_out, "<img src=\"");
        write_s(f_out,
            &input_buffer[index_to_first_byte_in_sequence+3]
        );
        write_s(f_out, "\">");
        return(index_to_last_byte_in_sequence);
        break;
    }
    /*
    Header
    */
    case '\x00' :
    {
        return(index_to_last_byte_in_sequence);
        break;
    }
    /*
    Color
    */
    case '\x01' :
    {
        return(index_to_last_byte_in_sequence);
        break;
    }
}

/*
Font
*/
case '\x02' :
{
    return(index_to_last_byte_in_sequence);
    break;
}
/*
Tabs and Dot Leaders
*/
case '\x09' :
{
    return(index_to_last_byte_in_sequence);
    break;
}
/*
End of Page-- Wordstar reccomends ignoring this
*/
case '\x0b' :
{
    return(index_to_last_byte_in_sequence);
    break;
}
/*
Page Offset-- Wordstar says this should not occur in files,
only in printer drivers
*/
case '\x0c' :
{
    return(index_to_last_byte_in_sequence);
    break;
}
/*
Paragraph Number
*/
case '\x0d' :
{
    return(index_to_last_byte_in_sequence);
    break;
}
/*
Index Item
*/
case '\x0e' :
{
    return(index_to_last_byte_in_sequence);
    break;
}
/*
User Print Control
*/
case '\x0f' :
{
    return(index_to_last_byte_in_sequence);
    break;
}
/*
Paragraph Style
*/
case '\x11' :
{
    return(index_to_last_byte_in_sequence);
    break;
}
/*
Alternate/Normal Font Change
*/
case '\x15' :
{
    return(index_to_last_byte_in_sequence);
    break;
}
/*
Truncation
*/
case '\x16' :
{
    return(index_to_last_byte_in_sequence);
    break;
}
/*
*/
default:
{
    return(index_to_last_byte_in_sequence);
    break;
}
} /* end switch */
}
/*
returns index of last byte which is still in the buffer,
and has been processed, so normal incrementing of the index
by the for-loopwill get one to a new byte,
*/

```

```

/*-----*/
int parse_note(
    int f_out,
    unsigned char input_buffer[],
    int index_to_first_byte_in_sequence,
    int index_to_last_byte_in_sequence
);
{
    unsigned char opcode,
        conversion_flag, inner_conversion_flag;
    int internal_sequence_offset,
        line_count, internal_sequence_line_count,
        note_number, length_of_internal_sequence,
        fst_loc_inside_seq_index,
        last_loc_inside_seq_index,
        inside_length_of_sequence;
    char temp_digit_str[20];
    /*Break out the header fields*/
    opcode = input_buffer[index_to_first_byte_in_sequence+3];
    line_count=get_int16_from_byte_array(
        input_buffer,
        index_to_first_byte_in_sequence+4
    );
    internal_sequence_offset=get_int16_from_byte_array(
        input_buffer,
        index_to_first_byte_in_sequence+6
    );
    conversion_flag = input_buffer[index_to_first_byte_in_sequence+8];
    /* ... and the boundaries of the data section */
    fst_loc_inside_seq_index =
        index_to_first_byte_in_sequence+9;
    last_loc_inside_seq_index =
        index_to_last_byte_in_sequence-3;
    inside_length_of_sequence = last_loc_inside_seq_index
        -fst_loc_inside_seq_index
        +1;
    /*locate the internal sequence lf it exists */
    if(
        start_of_internal_sequence_index=is_there_an_internal_sequence(
            input_buffer,
            fst_loc_inside_seq_index,
            last_loc_inside_seq_index,
            &length_of_internal_sequence
        )
        != -1
    )
    {
        /*and break out its fields*/
        internal_sequence_line_count=
            get_int16_from_byte_array(
                input_buffer,
                start_of_internal_sequence_index+4
            );
        note_number=
            get_int16_from_byte_array(
                input_buffer,
                start_of_internal_sequence_index+6
            );
    }
}

```

```

inner_conversion_flag =
    input_buffer[start_of_internal_sequence_index+8];
if(length_of_internal_sequence > 11)
{
    for(i=0; i < length_of_internal_sequence-11 ; i++)
        internal_sequence_data[i]
            =input_buffer[
                start_of_internal_sequence_index
                +8+i];
    internal_sequence_data[length_of_internal_sequence-11]=0;
}
else internal_sequence_data[0]=0;
/*
and remove the internal sequence
*/
inside_length_of_sequence
= mem_xcise(
    &input_buffer[fst_loc_inside_seq_index],
    inside_length_of_sequence,
    &input_buffer[start_of_internal_sequence_index],
    length_of_internal_sequence
);
}
/* ...and move the sequence to just before the terminal tag*/
memmove(
    &input_buffer[
        index_to_last_byte_in_sequence
        -inside_length_of_sequence
        - 6
    ],
    &input_buffer[fst_loc_inside_seq_index],
    inside_length_of_sequence
);
}

```

```

/* ...and add the terminal tag, <1d><04><00><ff><04><00><1d>*/
input_buffer[index_to_last_byte_in_sequence-6]=
input_buffer[index_to_last_byte_in_sequence]='\x1d';
input_buffer[index_to_last_byte_in_sequence-4]=
input_buffer[index_to_last_byte_in_sequence-1]='\x00';
input_buffer[index_to_last_byte_in_sequence-5]=
input_buffer[index_to_last_byte_in_sequence-2]='\x04';
input_buffer[index_to_last_byte_in_sequence-2]='\xff';
/*Build a front tag, and sent it to the output*/
write_s(f_out, "<ASIDE>")
switch(opcode)
{
case 3: write_s(f_out, "Footnote"); break;
case 4: write_s(f_out, "Endnote"); break;
case 5: write_s(f_out, "Annotation"); break;
case 6: write_s(f_out, "Comment"); break;
}
if(start_of_internal_sequence_index == -1)
if(internal_sequence_offset > 0)
{
sprintf(&temp_digit_str, "%d",
internal_sequence_offset);
write_s(f_out, &temp_digit_str);
}
else if(
(opcode == 3)
||
(opcode != 4)
)
{
sprintf(&temp_digit_str, "%d",note_number);
write_s(f_out, &temp_digit_str);
}
write_s(f_out, ";<P>")
/* ...and return*/
return(inside_length_of_sequence+7);
}
/*
returns the number of bytes inserted
each note will be converted to an "aside," "<aside>," and "</aside>"
*/
/*-----*/
int mem_xcise(void *dst_addr, size_t dst_len,void *src_addr, size_t
src_len)
{
memmove(
src_addr,
src_addr+src_len,
dst_len -(src_addr-dst_addr)-src_len
);
return(dst_len-src_len);
}
/*
excises source region [address-length]
from destination region [address-length],
and returns the new length of destination region
*/

```

```

/*-----*/
int is_there_an_internal_sequence(
unsigned char input_buffer[],
int fst_loc_inside_seq_index,
int last_loc_inside_seq_index,
int *length
)
{
int i,j, last_possible_index_for_inner_sequence;
last_possible_index_for_inner_sequence =
last_loc_inside_seq_index-6;
for(
i=fst_loc_inside_seq_index;
i >= last_possible_index_for_inner_sequence;
i++
)
{
if(
(input_buffer[i] == '\x1d')
&&
(input_buffer[i-1] != '\x1b')
)
{
if(
(j= symmetric_sequence_is_valid(input_buffer, i)
==
-1
)
return(-1)
else
{
*length=j-i;
return(i);
}
}
}
return(-1)
}
/*
Returns index of initiating [Hex 1D]
if there is a valid internal sequence, and sets length,
else returns -1

The Wordstar manual did not seem quite sure of itself,
where the location of internal sequences was concerned,
possibly commingling differenes between versions 5,6,
and 7. I therefore opt for a brute-force approach,
which is comparatively foolproof
*/
/*-----*/
int symmetric_sequence_is_valid(
unsigned char input_buffer[],
int index_to_first_byte_in_sequence
)
{
int offset, index_to_last_byte_in_sequence;
offset=get_int16_from_byte_array(
input_buffer,
index_to_first_byte_in_sequence+1
);

index_to_last_byte_in_sequence=
index_to_first_byte_in_sequence+offset+2;
if(
(input_buffer[index_to_first_byte_in_sequence] == '\x1d' )
&&
(input_buffer[index_to_last_byte_in_sequence] == '\x1d' )
&&
(
get_int16_from_byte_array(
input_buffer,
index_to_last_byte_in_sequence-2
)
==
offset
)
&&
(
offset > 0
)
)
return(index_to_last_byte_in_sequence);
else return(-1);
}
/*returns index of terminating byte [Hex 1D] if valid, else -1*/

```

```

/*-----*/
int get_int16_from_byte_array(
    unsigned char byte_array[],
    int index_to_array
    );
{
    unsigned char high_byte_of_integer, low_byte_of_integer;
    int integer_16;
    high_byte_of_integer=
        input_buffer[index_to_first_byte_in_sequence+1];
    low_byte_of_integer=
        input_buffer[index_to_first_byte_in_sequence];
    integer_16 =
        (int)low_byte_of_offset + (256 * (int) high_byte_of_offset );
    return(integer_16);
}
/*
    Obtains a 16-bit integer,
    stored in a byte array in orthodox PC (little-endian),
    fashion, given the index of the first byte
*/
/*-----*/
int parse_dot_command(
    int f_out,
    unsigned char input_buffer[],
    int index_to_buffer
    )
{
    int index_of_last_byte_of_dot_command, index_of_cr;
    char *addr_of_cr;
    /*find next carriage return*/
    if(
        addr_of_cr
        =memchr(&input_buffer[index_to_buffer], '\r', 256)
        == NULL
        )
    {
        index_of_last_byte_of_dot_command=index_to_buffer;
        send_char_to_html(f_out, input_buffer[index_to_buffer]);
        set_curr_column("incr");
    }
    /*
        if there is not a hard carriage return
        in a reasonable distance, best assume
        it is a period after all
    */
    else
    {
        index_of_cr=addr_of_cr-input_buffer;
        /*and the following linefeed*/
        if(input_buffer[index_of_cr+1] == '\n')
            index_of_last_byte_of_dot_command = index_of_cr+1;
        else
            index_of_last_byte_of_dot_command = index_of_cr;
        /*
            which gives you the bounds of the dot command line,
            and the index to return.
        */
        set_curr_column("hard");
        manage_flip_flop_char_attr(f_out, curr_c, "reset");
        /*
            makes the assumption that special
            attributes dont persist across
            a dot command
        */
        input_buffer[index_of_cr] == '\0';
        process_dot_command(f_out, &input_buffer[index_to_buffer+1]);
    }
    return(index_of_last_byte_of_dot_command);
}
/*
    returns index of last byte of the terminaing CR-LF,
    so normal incrementing of the index
    by the for-loopwill get one to a new byte,
*/
/*-----*/
process_dot_command(int f_out,char *command_string)
{
    if(strncmp(command_string, "rr", 2) == 0)
        process_dot_ruler_command(f_out,command_string);
    else if(strncmp(command_string, "fi", 2) == 0)
    {
        write_s(f_out, "<A href=\"");
        write_s(f_out, command_string+3);
        write_s(f_out, "\">");
        write_html_s(f_out, command_string);
        write_s(f_out, "</A>\r\n<P>");
    }
    else
    {
        write_s(f_out, "WORDSTAR DOT COMMAND: ");
        write_html_s(f_out, command_string);
        write_s(f_out, "\r\n<P>");
    }
}
}

```

```

/*-----*/
process_dot_ruler_command(int f_out,char *command_string)
{
    char ruler_type[30];
    int is_embedded_ruler_line;
    if(isdigit(command_string[2]))
    {
        ruler_type[0] = command_string[2];
        is_embedded_ruler_line = 0;
    }
    else if(
        (command_string[2] == ' ')
        &&
        isdigit(command_string[3])
        )
    {
        ruler_type[0] = command_string[3];
        is_embedded_ruler_line = 0;
    }
    else
    {
        ruler_type[0] = 'L';
        is_embedded_ruler_line = 1;
    }
    ruler_type[1] = '\0';
    set_ruler_mode(f_out,ruler_type);
    if(is_embedded_ruler_line)
    {
        write_html_s(f_out, command_string);
        write_s(f_out, "\r\n<P>");
    }
}
/*-----*/
set_ruler_mode(int f_out,char *ruler_type)
{
    static char current_mode;
    if(strcmp(ruler_type,"CR") == 0)
    {
        if(current_mode == '2')
            write_s(f_out, "\r\n<LI>\r\n<LI>");
        else write_s(f_out, "\r\n<P>");
    }
    else if(strcmp(ruler_type,"Reset") == 0)
    {}
    else if(strcmp(ruler_type,"Save") == 0)
    {}
    else if(strcmp(ruler_type,"Restore") == 0)
    {}
    else if(
        (strcmp(ruler_type,"0") == 0)
        || (strcmp(ruler_type,"3") == 0)
        || (strcmp(ruler_type,"4") == 0)
        || (strcmp(ruler_type,"5") == 0)
        || (strcmp(ruler_type,"6") == 0)
        || (strcmp(ruler_type,"7") == 0)
        || (strcmp(ruler_type,"8") == 0)
        || (strcmp(ruler_type,"9") == 0)
        ) /*Normal Mode*/
    {
        if(current_mode == '1')
            write_s(f_out, "\r\n<BLOCKQUOTE>");
        else if(current_mode == '2')
            write_s(f_out, "\r\n<LI></UL>");
        else if(current_mode == 'L')
            write_s(
                f_out,
                "\r\n<P>[End of Embedded Ruler Line]"
                );
        current_mode='0';
        if(strcmp(ruler_type,"0") != 0)
        { write_s(f_out, "\r\n<P> [Prewdefined Ruler type: ");
          write_s(f_out, ruler_type);
          write_s(f_out, "-- Format Unrecoverable\r\n<P>");
        }
    }
    else if(strcmp(ruler_type,"1") == 0) /*Block Quote */
    {
        if(current_mode == '0')
            write_s(f_out, "\r\n<BLOCKQUOTE>");
        else if(current_mode == '2')
            write_s(f_out, "\r\n<LI></UL><BLOCKQUOTE>");
        else if(current_mode == 'L')
            write_s(
                f_out,
                "\r\n<P>[End of Embedded Ruler Line]\r\n<BLOCKQUOTE>"
                );
        current_mode='1';
    }
}

```

```

else if(strcmp(ruler_type,"2") == 0) /* Outline */
{
    if(current_mode == '0')
        write_s(f_out, "\r\n<UL><LI>");
    else if(current_mode == 'L')
        write_s(f_out, "\r\n</BLOCKQUOTE><UL><LI>");
    else if(current_mode == '1')
        write_s(
            f_out,
            "\r\n<P>[End of Embedded Ruler Line]\r\n<UL><LI>"
        );
    current_mode='2';
}
else if(strcmp(ruler_type,"L") == 0)
{
    if(current_mode == '0')
        write_s(
            f_out,
            "\r\n<P>[Start Embedded Ruler Line]<P>\r\n"
        );
    else if(current_mode == '1')
        write_s(
            f_out,
            "\r\n</BLOCKQUOTE><P>[Start Embedded Ruler Line]<P>\r\n"
        );
    else if(current_mode == '2')
        write_s(
            f_out,
            "\r\n</LI></UL><P>[Start Embedded Ruler Line]<P>\r\n"
        );
    current_mode='L';
}
}
/*
Command Codes:0-9, L, Reset, CR, Save, Restore,
expressed as strings
Internal States: 0-9, L, expressed as a character

Predefined ruler lines can be set within the wordstar program,
which means they cannot be recovered from a wordstar file.
However, even if the user may have tinkered with the settings,
he will probably not have changed the
underlying intentions.
That said:
Predefined Ruler line 0: Normal.
1: Block Quote.
2: Outline,
which I gloss as Unordered List.
3-9: Purely User-Defined
*/
/*
*/
write_cr_according_to_ruler(int f_out)
{
    set_ruler_mode(f_out,"CR")
}
/*-----*/
parse_spaces(
    int f_out,
    unsigned char input_buffer[],
    int index_to_buffer
)
{
    int i;
    for(
        i=0;
        (input_buffer[index_to_buffer+i] == ' ')
        ||
        (input_buffer[index_to_buffer+i] == '\xa0')
        ;i++
    );
    for(j=0;j<i;j++) send_char_to_html(f_out, ' ');
}
/*
returns index of last byte of the group of spaces,
so normal incrementing of the index
by the for-loop will get one to a new byte,
*/

```