```
: # use perl                                          -*- mode: Perl; -*-

eval 'exec perl -S $0 "$@"'


if $runnning_under_some_shell;

# JS 12-6-95
# Moved a bunch of the subs to this file so that this file could be opened
# and modified by MacPerl because it was >32k before.
require "MacWebLint-lib.pl";

#
# weblint - pick fluff off WWW pages (html).
#
# Copyright (C) 1994, 1995 Neil Bowers.   All rights reserved.
#
# See README for additional blurb.
# Bugs, comments, suggestions welcome: neilb@khoral.com
#
# Latest version is available as:
#
ftp://ftp.khoral.com/pub/perl/www/weblint.tar.gz
#

$VERSION    = '1.011';
($PROGRAM = $0) =~ s@.*/@@;

# JS 12-6-95 Changed line below.
$TMPDIR     = 'tmp';

# JS 12-6-95
# Removed usage and todo because it is not really applicable to this port
# of the mac version.
$usage="lots of usage";
$todo="lots to do";

*WARNING = *STDOUT;

# obsolete tags
$obsoleteTags = 'PLAINTEXT|XMP|LISTING|COMMENT';

$maybePaired    = 'LI|DT|DD|P|ROW|TD|TH|TR';

$pairElements = 'A|ABBREV|ABOVE|ACRONYM|ADDRESS|ARRAY|AU|'.
                    'HTML|HEAD|BANNER|BAR|BELOW|BIG|BLOCKQUOTE|BODY|BOX|
```

```
BQ|BT|'.
                    'CAPTION|CREDIT|DDOT|DEL|DIV|DOT|'.
                    'FIG|FN|H1|H2|H3|H4|H5|H6|HAT|INS|LH|OVERLAY|'.


'B|I|U|TT|STRONG|EM|CODE|KBD|VAR|DFN|CITE|SAMP|Q|LANG|'.


'UL|OL|DL|'.
                    'MATH|MENU|DIR|FORM|NOTE|PERSON|ROOT|'.
                    'S|SELECT|SMALL|SQRT|STRIKE|STYLE|'.
                    'SUB|SUP|T|TABLE|TEXT|TEXTAREA|TILDE|TITLE|VEC|CODE|PRE|'.
                    $maybePaired.'|'.
                    $obsoleteTags;

# expect to see these tags only once
%onceOnly = ('HTML', 1, 'HEAD', 1, 'BODY', 1, 'TITLE', 1);

%physicalFontElements =
(
 'B',   'STRONG',
 'I',   'EM',
 'TT', 'CODE, SAMP, KBD, or VAR'
 );

# expect these tags to have attributes
# these are elements which have no required attributes, but we expect to
# see at least one of the attributes
$expectArgsRE = 'A';

# these tags can only appear in the head element
$headTagsRE = 'TITLE|NEXTID|LINK|BASE|META';

%requiredContext =
(
 'ABOVE',       'MATH',
 'ARRAY',       'MATH',
 'ATOP',        'BOX',
 'BAR',         'MATH',
 'BELOW',       'MATH',
 'BOX',         'MATH',
 'BT',          'MATH',
 'CAPTION',     'TABLE|FIG',
 'CHOOSE',      'BOX',
 'DD',          'DL',
 'DDOT',        'MATH',
```

```
 'DOT',          'MATH',
 'DT',           'DL',
 'HAT',          'MATH',
 'INPUT',        'FORM',
 'ITEM',         'ROW',
 'LEFT',         'BOX',
 'LH',           'DL|OL|UL',
 'LI',           'DIR|MENU|OL|UL',
 'OF',           'ROOT',
 'OPTION',       'SELECT',
 'OVER',         'BOX',
 'OVERLAY',      'FIG',
 'RIGHT',        'BOX',
 'ROOT',         'MATH',
 'ROW',          'ARRAY',
 'SELECT',       'FORM',
 'SQRT',         'MATH',
 'T',            'MATH',
 'TD',           'TR',
 'TEXT',         'MATH',
 'TEXTAREA',     'FORM',
 'TH',           'TR',
 'TILDE',        'MATH',
 'TR',           'TABLE',
 'VEC',          'MATH'
 );

# these tags are allowed to appear in the head element
%okInHead = ('ISINDEX', 1, 'TITLE', 1, 'NEXTID', 1, 'LINK', 1,

       'BASE', 1, 'META', 1, 'RANGE', 1, 'STYLE', 1, '!--', 1);

# expect to see these at least once.
# html-outer covers the HTML element
@expectedTags = ('HEAD', 'TITLE', 'BODY');

# elements which cannot be nested
$nonNest = 'A|FORM';

$netscapeElements = 'NOBR|WBR|FONT|BASEFONT|BLINK|CENTER';

#
# This is a regular expression for all legal elements
# UPDATE: need to remove duplication in legalElements and pairElements
#
$legalElements =
```

```
    'A|ABBREV|ABOVE|ACRONYM|ADDRESS|ARRAY|ATOP|AU|'.
    'B|BANNER|BAR|BASE|BELOW|BIG|BLOCKQUOTE|BODY|BOX|BQ|BR|BT|'.
    'CAPTION|CHOOSE|CITE|CODE|CREDIT|'.
    'DD|DDOT|DFN|DEL|DIR|DIV|DL|DOT|DT|'.
    'EM|FIG|FN|FORM|H1|H2|H3|H4|H5|H6|HAT|HEAD|HR|HTML|'.
    'I|IMG|INPUT|INS|ISINDEX|ITEM|KBD|'.
    'LANG|LEFT|LH|LI|LINK|MATH|MENU|META|NEXTID|NOTE|'.
    'OF|OL|OPTION|OVER|OVERLAY|P|PERSON|PRE|Q|RANGE|RIGHT|ROOT|
ROW|'.
    'SAMP|SELECT|S|SMALL|SQRT|STRIKE|STRONG|STYLE|SUB|SUP|'.
    'T|TAB|TABLE|TD|TEXT|TEXTAREA|TH|TILDE|TITLE|TR|TT|U|UL|VAR|VEC|'.
    $obsoleteTags;

# This table holds the valid attributes for elements
# Where an element does not have an entry, this implies that the element
# does not take any attributes
%validAttributes =
    (
    'A',              'ID|LANG|CLASS|HREF|MD|NAME|SHAPE|TITLE|REL|REV',
    'ABOVE',          'SYM',
    'ADDRESS',        'ID|LANG|CLASS|CLEAR|NOWRAP',
    'ARRAY',          'ALIGN|COLDEF|LDELIM|RDELIM|LABELS',
    'BANNER',         'ID|LANG|CLASS',
    'BASE',           'HREF',
    'BR',             'ID|LANG|CLASS|CLEAR',
    'BLOCKQUOTE', 'ID|LANG|CLASS|CLEAR|NOWRAP',
    'BODY',           'ID|LANG|CLASS|BACKGROUND',
    'BOX',            'SIZE',
    'BQ',             'ID|LANG|CLASS|CLEAR|NOWRAP',
    'BELOW',          'SYM',
    'CAPTION',        'ID|LANG|CLASS|ALIGN',
    'CREDIT',         'ID|LANG|CLASS',
    'DD',             'ID|LANG|CLASS|CLEAR',
    'DIV',            'ID|LANG|CLASS|ALIGN|NOWRAP|CLEAR',
    'DL',             'ID|LANG|CLASS|CLEAR|COMPACT',
    'DT',             'ID|LANG|CLASS|CLEAR',
    'FIG',            'ID|LANG|CLASS|CLEAR|NOFLOW|SRC|MD|ALIGN|WIDTH|HEIGHT|'.
                      'UNITS|IMAGEMAP',
    'FN',             'ID|LANG|CLASS',
    'FORM',           'ACTION|METHOD|ENCTYPE|SCRIPT',
    'H1',             'ID|LANG|CLASS|ALIGN|CLEAR|SEQNUM|SKIP|DINGBAT|SRC|MD|
NOWRAP',
    'H2',             'ID|LANG|CLASS|ALIGN|CLEAR|SEQNUM|SKIP|DINGBAT|SRC|MD|
NOWRAP',
    'H3',             'ID|LANG|CLASS|ALIGN|CLEAR|SEQNUM|SKIP|DINGBAT|SRC|MD|
NOWRAP',
```

```
    'H4',           'ID|LANG|CLASS|ALIGN|CLEAR|SEQNUM|SKIP|DINGBAT|SRC|MD|
NOWRAP',
    'H5',           'ID|LANG|CLASS|ALIGN|CLEAR|SEQNUM|SKIP|DINGBAT|SRC|MD|
NOWRAP',
    'H6',           'ID|LANG|CLASS|ALIGN|CLEAR|SEQNUM|SKIP|DINGBAT|SRC|MD|
NOWRAP',
    'HR',           'ID|CLASS|CLEAR|SRC|MD',
    'HTML',          'VERSION|URN|ROLE',
    'IMG',          'ID|LANG|CLASS|SRC|MD|WIDTH|HEIGHT|UNITS|ALIGN|ALT|
ISMAP',
    'INPUT',        'ID|LANG|CLASS|TYPE|NAME|VALUE|DISABLED|ERROR|
CHECKED|SIZE|'.
                    'MAXLENGTH|MIN|MAX|ACCEPT|SRC|MD|ALIGN',
    'ITEM',         'ALIGN|COLSPAN|ROWSPAN',
    'LH',           'ID|LANG|CLASS',
    'LI',           'ID|LANG|CLASS|CLEAR|SRC|MD|DINGBAT|SKIP',
    'LINK',         'HREF|REL|REV|URN|TITLE|METHODS',
    'MATH',          'ID|CLASS|BOX',
    'META',          'HTTP-EQUIV|NAME|CONTENT',
    'NEXTID',       'N',
    'NOTE',          'ID|LANG|CLASS|CLEAR|SRC|MD',
    'OL',           'ID|LANG|CLASS|CLEAR|CONTINUE|SEQNUM|COMPACT',
    'OPTION',       'ID|LANG|CLASS|DISABLED|ERROR|VALUE|SELECTED|SHAPE',
    'OVERLAY',      'SRC|MD|UNITS|X|Y|WIDTH|HEIGHT',
    'P',            'ID|LANG|CLASS|ALIGN|CLEAR|NOWRAP',
    'PRE',          'ID|LANG|CLASS|CLEAR|WIDTH',
    'RANGE',         'ID|CLASS|FROM|UNTIL',
    'ROW',          'ALIGN|COLSPAN|ROWSPAN',
    'SELECT',       'ID|LANG|CLASS|NAME|MULTIPLE|DISABLED|ERROR|SRC|MD|
WIDTH|'.
                    'HEIGHT|UNITS|ALIGN',
    'STYLE',        'NOTATION',
    'TAB',          'ID|INDENT|TO|ALIGN|DP',
    'TABLE',        'ID|LANG|CLASS|CLEAR|NOFLOW|ALIGN|UNITS|COLSPEC|DP|
WIDTH|'.
                    'BORDER|NOWRAP',
    'TD',           'ID|LANG|CLASS|COLSPAN|ROWSPAN|ALIGN|DP|VALIGN|
NOWRAP|'.
                    'AXIS|AXES',
    'TEXTAREA',     'ID|LANG|CLASS|NAME|ROWS|COLS|DISABLED|ERROR|ALIGN',
    'TH',           'ID|LANG|CLASS|COLSPAN|ROWSPAN|ALIGN|DP|VALIGN|
NOWRAP|'.
                    'AXIS|AXES',
    'TR',           'ID|LANG|CLASS|ALIGN|DP|VALIGN|NOWRAP',
    'UL',           'ID|LANG|CLASS|CLEAR|PLAIN|SRC|MD|DINGBAT|WRAP|
COMPACT',
```

```
    );

%requiredAttributes =
    (
    'BASE',        'HREF',
    'FORM',        'ACTION',
    'IMG',         'SRC',
    'LINK',        'HREF',
    'NEXTID',      'N',
    'SELECT',      'NAME',
    'STYLE',       'NOTATION',
    'TEXTAREA', 'NAME|ROWS|COLS'
    );

%validNetscapeAttributes =
    (
    'ISINDEX',   'PROMPT',
    'HR',          'SIZE|WIDTH|ALIGN|NOSHADE',
    'UL',          'TYPE',
    'OL',          'TYPE|START',
    'LI',          'TYPE|VALUE',
    'IMG',         'BORDER|VSPACE|HSPACE',
    'BODY',        'BGCOLOR|TEXT|LINK|VLINK|ALINK',
    'TABLE',       'CELLSPACING|CELLPADDING',
    'TD',          'WIDTH',
    'TH',          'WIDTH'
    );

%mustFollow =
(
 'LH',          'UL|OL|DL',
 'OVERLAY',   'FIG',
 'HEAD',        'HTML',
 'BODY',        '/HEAD',
 '/HTML',       '/BODY',
 );

## JS 12-6-95
## changed to default.html from index.html because mac people
## usually use default.html :)
%variable =
(
 'directory-index',
'default.html',
 'url-get',
```

```
'lynx -source',
 'message-style',
'lint'
);

@options = ('d=s', 'e=s', 'stderr', 'help', 'i', 'l', 's', 't', 'todo', 'U',

     'urlget=s', 'v', 'version', 'warnings', 'x=s');

$exit_status = 0;

require 'newgetopt.pl';
require 'find.pl';

# JS 12-6-95
# Line below crashes MacPerl for some reason.
# die "$usage" if @ARGV == 0;

&ReadDefaults();
&GetConfigFile();

# escape the `-' command-line switch (for stdin), so NGetOpt don't mess wi' it
grep(s/^-$/\tstdin\t/, @ARGV);

&NGetOpt(@options) || die "use -U switch to display usage statement\n";

# put back the `-' command-line switch, if it was there
grep(s/^\tstdin\t$/-/, @ARGV);

die "$PROGRAM v$VERSION\n"                  if $opt_v || $opt_version;
die "$usage"                                if $opt_u || $opt_help;
die "$todo"                                 if $opt_todo;
&AddExtension($opt_x)                       if $opt_x;
$variable'message-style' = 'short'    if $opt_s;
$variable'message-style' = 'terse'    if $opt_t;
$variable'url-get' = $opt_urlget     if $opt_urlget;
*WARNING = *STDERR                          if $opt_stderr;
&ListWarnings()

                    if $opt_warnings;

# WARNING file handle is default
select(WARNING);

$opt_l = 1                        if $ignore'SYMLINKS';
```

```
# -d to disable warnings
if ($opt_d)

    for (split(/,/,$opt_d))

        &enableWarning($_, 0);



# -e to enable warnings
if ($opt_e)

    for (split(/,/,$opt_e))

        &enableWarning($_, 1) || next;



# -i option to ignore case in element tags
if ($opt_i)

    $enabled'lower-case' = $enabled'upper-case' = 0;


while (@ARGV > 0)

    $arg = shift(@ARGV);

    &CheckURL($arg), next if $arg =~ m!^(http|gopher|ftp)://!;

    &find($arg), next if -d $arg;

    &WebLint($arg), next if (-f $arg && -r $arg) || $arg eq '-';

    print "$PROGRAM: could not read $arg: $!\n";


exit $exit_status;

#==============================================================
=======
# Function:
WebLint
# Purpose:
This is the high-level interface to the checker.    It takes
#
```

```
a file and checks for fluff.
#====================================================================
=======
sub WebLint

    local($filename,$relpath) = @_;
    local(@tags) = ();
    local($tagRE) = '';
    local(@taglines) = ();
    local(@orphans) = ();
    local(@orphanlines) = ();
    local(%seenPage);
    local(%seenTag);
    local(%whined);
    local(*PAGE);
    local($line) = '';
    local($id, $ID);
    local($tag);
    local($closing);
    local($tail);
    local(%args);
    local($arg);
    local($rest);
    local($lastNonTag);
    local(@notSeen);
    local($seenMailtoLink) = (0);
    local($matched);
    local($matchedLine);
    local($novalue);
    local($heading);
    local($headingLine);
    local($commentline);
    local($_);


    if ($filename eq '-')

        *PAGE = *STDIN;
        $filename = 'stdin';

    else

        return if defined $seenPage$filename;
        if (-d $filename)
```

```
print "$PROGRAM: $filename is a directory.\n";

$exit_status = 0;

return;

        $seenPage$filename++;
        open(PAGE,"<$filename") || do


print "$PROGRAM: could not read file $filename: $!\n";

$exit_status = 0;

return;
        ;
        $filename = $relpath if defined $relpath;


   undef $heading;

READLINE:
   while (<PAGE>)

        $line .= $_;
        $line =~ s/\n/ /g;

        while ($line =~ /</o)


$tail = $'; #'

undef $lastNonTag;

$lastNonTag = $` if $` !~ /^\s*$/o;


#-------------------------------------------------------

#== SGML comment: <!-- ... blah blah ... -->

#-------------------------------------------------------

if ($tail =~ /^!--/o)
```

```
    $commentline = $. unless defined $commentline;

    # push lastNonTag onto word list for spell checking

    $ct = $';
    next READLINE unless $ct =~ /--\s*>/o;

    undef $commentline;

    $comment = $`;
    $line = $';

    # markup embedded in comment can confuse some (most? :-) browsers
    &whine($., 'markup-in-comment') if $comment =~ /<\s*[^>]+>/o;
    next;


undef $commentline;

next READLINE unless $tail =~ /^(\s*)([^>]*)>/;


&whine($., 'leading-whitespace', $2) if $1 ne '';

        $id = $tag = $2;
        $line = $';

        &whine($., 'unknown-element', $id),next if $id =~ /^\s*$/;

# push lastNonTag onto word list for spell checking
```

```
            undef $tail;
            undef $closing;
            undef %args;

            #-- <!DOCTYPE ... > is ignored for now.
            next if $id =~ /^!doctype/io;


$closing = 0;
            if ($id =~ m@^/@o)

                $id =~ s@^/@@;

      $ID = "\U$id";
                $closing = 1;


            #---------------------------------------------------------
            #== some seriously ugly code to handle attributes ...
            #---------------------------------------------------------

if ($closing == 0 && $tag =~ m|^(\S+)\s+(.*)|)

                ($id,$tail) = ($1,$2);

      $ID = "\U$id";

      $tail =~ s/\n/ /g;

                # check for odd number of quote characters
                ($quotes = $tail) =~ s/[^"]//g;
                &whine($., 'odd-quotes', $tag) if length($quotes) % 2 == 1;


      $novalue = 0;

      $valid = $validAttributes$ID;

      while ($tail =~ /^\s*([^=\s]+)\s*=\s*(.*)$/


    # catch attributes like ISMAP for IMG, with no arg


    || ($tail =~ /^\s*(\S+)(.*)/ && ($novalue = 1)))
```

```
        $arg = "\U$1";
        $rest = $2;
                &whine($., 'unexpected-open', $tag) if $arg =~ /</;
        if ($arg !~ /^($valid)$/i && $ID =~ /^($legalElements)$/o)

if ($arg =~ /^($validNetscapeAttributes$ID)$/i)

    &whine($., 'netscape-attribute', $arg, $id);

else

    &whine($., 'unknown-attribute', $id, $arg);

                #-- catch repeated attributes.   for example:
                #--        <IMG SRC="foo.gif" SRC="bar.gif">
                if (defined $args$arg)
                    &whine($., 'repeated-attribute', $arg, $id);
```

```
        if ($novalue)



$args$arg = '';

$tail = $rest;



        elsif ($rest =~ /^'([^']+)'(.*)$/)



&whine($., 'attribute-delimiter', $arg, $ID);
                        $args$arg = $1;
                        $tail = $2;


        elsif ($rest =~ /^"([^"]+)"(.*)$/


      || $rest =~ /^'([^']+)'(.*)$/


      || $rest =~ /^(\S+)(.*)$/)

                        $args$arg = $1;
                        $tail = $2;

                else

                        $args$arg = $rest;
                        $tail = '';


        $novalue = 0;


   &whine($., 'unexpected-open', $tag) if $tail =~ /</o;
```

```
else

                if ($closing && $id =~ m|^(\S+)\s+(.*)|)

        &whine($., 'closing-attribute', $tag);
        $id = $1;

    $ID = "\U$id";


$TAG = ($closing ? "/" : "").$ID;
if (defined $mustFollow$TAG)

    $ok = 0;
    foreach $pre (split(/\|/, $mustFollow$TAG))

        ($ok=1),last if $pre eq $lastTAG;

    if (!$ok || $lastNonTag !~ /^\s*$/)

        &whine($., 'must-follow', $TAG, $mustFollow$TAG);



#-- catch empty container elements
if ($closing && $ID eq $lastTAG && $lastNonTag =~ /^\s*$/
```

```
      && $ID ne 'TEXTAREA')

    &whine($., 'empty-container', $ID);

#-- special case for empty optional container elements
if (!$closing && $ID eq $tags[$#tags] && $lastTAG eq $ID
      && $ID =~ /^($maybePaired)$/
      && $lastNonTag =~ /^\s*$/)

    $t = pop @tags;
    $tline = pop @taglines;
    &whine($tline, 'empty-container', $ID);
    $tagRE = join('|',@tags);

        #-- whine about unrecognized element, and do no more checks ----
        if ($id !~ /^($legalElements)$/io)

    if ($id =~ /^($netscapeElements)$/io)

        &whine($., 'netscape-markup', ($closing ? "/$id" : "$id"));

    else
```

```
        &whine($., 'unknown-element', ($closing ? "/$id" : "$id"));


    next;


            if ($closing == 0 && defined $requiredAttributes$ID)


    @argkeys = keys %args;

    foreach $attr (split(/\|/,$requiredAttributes$ID))


        unless (defined $args$attr)


&whine($., 'required-attribute', $attr, $id);




            elsif ($closing == 0 && $id =~ /^($expectArgsRE)$/io)

                &whine($., 'expected-attribute', $id) unless defined %args;


            #-----------------------------------------------------
            #== check case of tags
            #-----------------------------------------------------
            &whine($., 'upper-case', $id) if $id ne $ID;
            &whine($., 'lower-case', $id) if $id ne "\L$id";


            #-----------------------------------------------------
            #== if tag id is /foo, then strip slash, and mark as a closer
            #-----------------------------------------------------
            if ($closing)
```

```
if ($ID !~ /^($pairElements)$/o)

    &whine($., 'illegal-closing', $id);

        if ($ID eq 'A' && $lastNonTag =~ /^\s*here\s*$/io)
            &whine($., 'here-anchor');

#-- end of HEAD, did we see a TITLE in the HEAD element? ----
&whine($., 'require-head') if $ID eq 'HEAD' && !$seenTag'TITLE';

#-- was there a <LINK REV=MADE HREF="mailto:..."> element in HEAD?
&whine($., 'mailto-link') if $ID eq 'HEAD' && $seenMailtoLink == 0;
      else
          #-----------------------------------------------------
          # do context checks.   Should really be a state machine.
          #-----------------------------------------------------

if (defined $physicalFontElements$ID)

    &whine($., 'physical-font', $ID, $physicalFontElements$ID);

        if ($ID eq 'A' && defined $args'HREF')

    $target = $args'HREF';
            if ($target =~ /([^:]+):\/\/([^\/]+)(.*)$/
```

```
    || $target =~ /^(news|mailto):/

    || $target =~ /^\//)

                    else


   $target =~ s/#.*$//;

   if ($target !~ /^\s*$/ && ! -f $target && ! -d $target)



       &whine($., 'bad-link', $target);



                if ($ID =~ /^H(\d)$/o)

                    if (defined $heading && $1 - $heading > 1)

              &whine($., 'heading-order', $ID, $heading, $headingLine);

                    $heading     = $1;
                    $headingLine = $.;



      #-- check for mailto: LINK ----------------------------

      if ($ID eq 'LINK' && $args'REV' =~ /^made$/io

&& $args'HREF' =~ /^mailto:/io)
```

```
    $seenMailtoLink = 1;



if (defined $onceOnly$ID)

    &whine($., 'once-only', $ID, $seenTag$ID) if $seenTag$ID;

        $seenTag$ID = $.;

        &whine($., 'body-no-head') if $ID eq 'BODY' && !$seenTag'HEAD';

        if ($ID ne 'HTML' && $ID ne '!DOCTYPE' && !$seenTag'HTML'
            && !$whined'outer-html')

            &whine($., 'html-outer');
            $whined'outer-html' = 1;

#-- check for illegally nested elements ----------

if ($ID =~ /^($nonNest)$/o && $ID =~ /^($tagRE)$/)

    for ($i=$#tags; $tags[$i] ne $ID; --$i)



    &whine($., 'nested-element', $ID, $taglines[$i]);

&whine($., 'unknown-element', $ID) unless $ID =~ /^($legalElements)$/o;
```

```
    #-----------------------------------------------------
    # check for tags which have a required context
    #-----------------------------------------------------
    if (defined ($reqCon = $requiredContext$ID))

        $ok = 0;
        foreach $context (split(/\|/, $requiredContext$ID))

($ok=1),last if $context =~ /^($tagRE)$/;

        unless ($ok)
                    &whine($., 'required-context', $ID, $requiredContext$ID);


    #-----------------------------------------------------
    # check for tags which can only appear in the HEAD element
    #-----------------------------------------------------
    if ($ID =~ /^($headTagsRE)$/o && 'HEAD' !~ /^($tagRE)$/)
                &whine($., 'head-element', $ID);
```

```
  if (! defined $okInHead$ID && 'HEAD' =~ /^($tagRE)$/)

                &whine($., 'non-head-element', $ID);



  #-----------------------------------------------------

  # check for tags which have been deprecated (now obsolete)

  #-----------------------------------------------------

  &whine($., 'obsolete', $ID) if $ID =~ /^($obsoleteTags)$/o;


      #-----------------------------------------------------
      #== was tag of type <TAG> ... </TAG>?
      #== welcome to kludgeville, population seems to be on the increase!
      #-----------------------------------------------------
      if ($ID =~ /^($pairElements)$/o)

  #-- if we have a closing tag, and the tag(s) on top of the stack

  #-- are optional closing tag elements, pop the tag off the stack,

  #-- unless it matches the current closing tag

  if ($closing)


     while (@tags > 0 && $tags[$#tags] ne $ID

    && $tags[$#tags] =~ /^($maybePaired)$/o)



pop @tags;

pop @taglines;
```

```
        $tagRE = join('|',@tags);


              if ($closing && $tags[$#tags] eq $ID)

                  $matched     = pop @tags;
                  $matchedLine = pop @taglines;


        #-- does top of stack match top of orphans stack? --------

        while (@orphans > 0 && @tags > 0

    && $orphans[$#orphans] eq $tags[$#tags])



  &whine($., 'element-overlap', $orphans[$#orphans],


$orphanlines[$#orphanlines], $matched, $matchedLine);

  pop @orphans;

  pop @orphanlines;

  pop @tags;

  pop @taglines;

                  $tagRE = join('|',@tags);

              elsif ($closing && $tags[$#tags] ne $ID)
```

```
        #-- closing tag does not match opening tag on top of stack
        if ($ID =~ /^($tagRE)$/)


   # If we saw </HTML>, </HEAD>, or </BODY>, then we try
   # and resolve anything inbetween on the tag stack
   if ($ID =~ /^(HTML|HEAD|BODY)$/o)


       while ($tags[$#tags] ne $ID)


$ttag = pop @tags;

$ttagline = pop @taglines;

if ($ttag !~ /^($maybePaired)$/)


    &whine($., 'unclosed-element', $ttag, $ttagline);
```

```
#-- does top of stack match top of orphans stack? --

while (@orphans > 0 && @tags > 0

        && $orphans[$#orphans] eq $tags[$#tags])



    pop @orphans;

    pop @orphanlines;

    pop @tags;

    pop @taglines;



      #-- pop off the HTML, HEAD, or BODY tag ----------
```

```
    pop @tags;

    pop @taglines;

    $tagRE = join('|',@tags);



else



    #-- matched opening tag lower down on stack

    push(@orphans, $ID);

    push(@orphanlines, $.);



      else



&whine($., 'mis-match', $ID);


            else

                push(@tags,$ID);
                $tagRE = join('|',@tags);
```

```
                push(@taglines,$.);


            #-----------------------------------------------------
            #== inline images (IMG) should have an ALT argument :-)
            #-----------------------------------------------------
            &whine($., 'img-alt') if ($ID eq 'IMG'



    && !defined $args'ALT'



    && !$closing);

         continue
            $lastTAG = $TAG;

        $lastNonTag = $line;

    close PAGE;

    if (defined $commentline)

        &whine($commentline, 'unclosed-comment');
        return;


    while (@tags > 0)

        $tag = shift(@tags);
        $line = shift(@taglines);
        if ($tag !~ /^($maybePaired)$/)


&whine($., 'unclosed-element', $tag, $line);



    for (@expectedTags)

        # if we haven't seen TITLE but have seen HEAD
```

```
        # then we'll have already whined about the lack of a TITLE element
        next if $_ eq 'TITLE' && !$seenTag$_ && $seenTag'HEAD';
        push(@notSeen,$_) unless $seenTag$_;

    if (@notSeen > 0)

        printf ("%sexpected tag(s) not seen: @notSeen\n",


        ($opt_s ? "" : "$filename(-): "));
        $exit_status = 1;




#=================================================================
=======
# Function:
ReadDefaults
# Purpose:
Read the built-in defaults.   These are stored at the end
#                   of the script, after the __END__, and read from the
#                   DATA filehandle.
#=================================================================
=======
sub ReadDefaults

    local(@elements);


    while (<DATA>)

        chop;
        s/^\s*//;
        next if /^$/;

        push(@elements, $_);

        next unless @elements == 3;

        ($id, $default, $message) = @elements;
        $enabled$id = ($default eq 'ENABLE');
        ($message$id = $message) =~ s/"/\\"/g;
        undef @elements;
```

__END__
upper-case

DISABLE

tag <$argv[0]> is not in upper case.
lower-case

DISABLE

tag <$argv[0]> is not in lower case.
mixed-case

ENABLE

tag case is ignored
here-anchor

ENABLE

bad form to use `here' as an anchor!
require-head

ENABLE

no <TITLE> in HEAD element.
once-only

ENABLE

tag <$argv[0]> should only appear once. I saw one on line $argv[1]!
body-no-head

ENABLE

<BODY> but no <HEAD>.
html-outer

ENABLE

outer tags should be <HTML> .. </HTML>.
head-element

ENABLE

<$argv[0]> can only appear in the HEAD element.
non-head-element

ENABLE

<$argv[0]> cannot appear in the HEAD element.
obsolete

ENABLE

<$argv[0]> is obsolete.
mis-match

ENABLE

unmatched </$argv[0]> (no matching <$argv[0]> seen).
img-alt

ENABLE

IMG does not have ALT text defined.
nested-element

ENABLE

<$argv[0]> cannot be nested -- </$argv[0]> not yet seen for <$argv[0]> on line $argv[1].
mailto-link

DISABLE

did not see <LINK REV=MADE HREF="mailto..."> in HEAD.
element-overlap

ENABLE

</$argv[0]> on line $argv[1] seems to overlap <$argv[2]>, opened on line $argv[3].
unclosed-element

ENABLE

no closing </$argv[0]> seen for <$argv[0]> on line $argv[1].
markup-in-comment

ENABLE

markup embedded in a comment can confuse some browsers.

unknown-attribute

ENABLE

unknown attribute "$argv[1]" for element <$argv[0]>.
leading-whitespace

ENABLE

should not have whitespace between "<" and "$argv[0]>".
required-attribute

ENABLE

the $argv[0] attribute is required for the <$argv[1]> element.
unknown-element

ENABLE

unknown element <$argv[0]>.
odd-quotes

ENABLE

odd number of quotes in element <$argv[0]>.
heading-order

ENABLE

bad style - heading <$argv[0]> follows <H$argv[1]> on line $argv[2].
bad-link

DISABLE

target for anchor "$argv[0]" not found.
expected-attribute

ENABLE

expected an attribute for <$argv[0]>.
unexpected-open

ENABLE

unexpected < in <$argv[0]> -- potentially unclosed element.
required-context

ENABLE

illegal context for <$argv[0]> - must appear in <$argv[1]> element.
unclosed-comment

ENABLE

unclosed comment (comment should be: <!-- ... -->).
illegal-closing

ENABLE

element <$argv[0]> is not a container -- </$argv[0]> not legal.
netscape-markup

ENABLE

<$argv[0]> is netscape specific (use "-x netscape" to allow this).
netscape-attribute

ENABLE

attribute `$argv[0]' for <$argv[1]> is netscape specific (use "-x netscape" to allow this).
physical-font

DISABLE

<$argv[0]> is physical font markup -- use logical (such as $argv[1]).
repeated-attribute

ENABLE

attribute $argv[0] is repeated in element <$argv[1]>
must-follow

ENABLE

<$argv[0]> must immediately follow <$argv[1]>
empty-container

ENABLE

empty container element <$argv[0]>.
directory-index

ENABLE

directory $argv[0] does not have an index file ($argv[1])
closing-attribute

ENABLE

closing tag <$argv[0]> should not have any attributes specified.
attribute-delimiter

ENABLE

use of ' for attribute value delimiter is not supported by all browsers (attribute $argv[0] of tag $argv[1])