The Weaver 3/weave

Purpose

To weave a portion of the code into instructions for TeX.

§**1. The Master Weaver.** Here's what has happened so far, on a `-weave` run of `inweb`: on any other sort of run, of course, we would never be in this section of code. The web was read completely into memory, and then fully parsed, with all of the arrays and hashes populated. A request was then made either to swarm a mass of individual weaves, or to make just a single weave, with the target in each case being identified by its sigil. A further decoding layer then translated each sigil into rather more basic details of what to weave and where to put the result: and so we arrive at the front door of the routine `weave_source` below.

```
sub weave_source {
    my $filename = $_[0];
    my $with_cover_sheet = $_[1];                    Should we add a cover sheet at the front?
    my $weave_section_match = $_[2];                 Regular expression which sigils for the lines must match
    open(WEAVEOUT, ">".$filename) or die "inweb: can't open weave file '$filename' for output";
    print WEAVEOUT "% Weave ", $filename, " generated by ", $INWEB_BUILD, "\n";
    ⟨Incorporate suitable TEX macro definitions into the woven output 3⟩;
    if ($with_cover_sheet == 1) { weave_cover_sheet(); }
    ⟨Start the weaver with a clean slate 2⟩;
    OUTLOOP:
    for ($i=0; $i<$no_lines; $i++) {
        ⟨Skip material from any sections which are not part of this target 4⟩;
        $lines_woven++;                              In principle, then, what remains is in the target; but...
        ⟨Material between ...and so on... markers is not visible 5⟩;
        ⟨Grammar and index entries are collated elsewhere, not woven in situ 6⟩;
        ⟨Respond to any commands aimed at the weaver, and otherwise skip commands 7⟩;
        Some of the more baroque front matter of a section...
        ⟨Weave the Purpose marker as a little heading 9⟩;
        ⟨If we need to work in a section table of contents and this is a blank line, do it now 10⟩;
        ⟨Deal with the Interface passage 11⟩;
        ⟨Weave the Definitions marker as a little heading 12⟩;
        ⟨Weave the section bar as a horizontal rule 13⟩;
        The crucial junction point between modes...
        ⟨Deal with the marker for the start of a new paragraph, section or chapter 32⟩;
        With all exotica dealt with, we now just have material to weave verbatim...
        my $matter = $line_text[$i];
        if ($line_is_comment[$i] == 1) ⟨Weave verbatim matter in commentary style 14⟩
        else ⟨Weave verbatim matter in code style 19⟩;
    }
    ⟨Complete any started but not-fully-woven paragraph 36⟩;
    print WEAVEOUT "% End of weave: ", $lines_woven, " lines from ", $no_lines, "\n";
    print WEAVEOUT '\end', "\n";
    close(WEAVEOUT);
}
```

§**2.** We can now begin on a clean page, by initialising the state of the weaver. It's convenient for these to be global variables since the weaver is not recursively called, and it avoids some nuisance over scope caused by the braces implicit in the `inweb` macro below:

⟨Start the weaver with a clean slate 2⟩ ≡

```
$within_TeX_beginlines = 0;                 Currently setting copy between \beginlines and \endlines?
$weaving_suspended = 0;
$interface_table_pending = 0;
$functions_in_par = ""; $structs_in_par = "";
$current_macro_definition = "";
$next_heading_without_vertical_skip = 0;
$show_section_toc_soon = 0;                 Is a table of contents for the section imminent?
$horizontal_rule_just_drawn = 0;
$chaptermark = ""; $sectionmark = "";
begin_making_pdf_links();                   Be ready to set PDF hyperlinks from now on
$lines_woven = 0;                           Number of lines in the target to be woven
```

This code is used in §1.

§**3. The TEX macros.** We don't use TEX's `\input` mechanism for macros because it is so prone to failures when searching directories (especially those with spaces in the names) and then locking TEX into a repeated prompt for help from `stdin` which is rather hard to escape from.

Instead we paste the entire text of our macros file into the woven TEX:

⟨Incorporate suitable TEX macro definitions into the woven output 3⟩ ≡

```
my $ml;
open(MACROS, $path_to_inweb_setting.'inweb/Materials/inweb-macros.tex')
    or die "inweb: can't open file of TeX macros";
while ($ml = <MACROS>) {
    $ml =~ m/^(.*)\s*$/; $ml = $1;
    print WEAVEOUT $ml, "\n";
}
close MACROS;
```

This code is used in §1.

§**4. Reasons to skip things.** We skip any material in files not chosen at the command line (or by the swarmer) for weave output:

⟨Skip material from any sections which are not part of this target 4⟩ ≡

```
if ($weave_section_match ne "") {
    if (not ($section_sigil[$line_sec[$i]] =~ m/$weave_section_match/)) { next OUTLOOP; }
}
```

This code is used in §1.

§**5.** We skip material between "...and so on..." markers as being even more tedious than the rest of the program:

⟨Material between ...and so on... markers is not visible 5⟩ ≡
```
if ($line_category[$i] == $TOGGLE_WEAVING_LCAT) {
    if ($weaving_suspended == 0) {
        print WEAVEOUT "\\smallskip\\par\\noindent";
        print WEAVEOUT "{\\ttninepoint\\it ...and so on...}\\smallskip\n";
        $weaving_suspended = 1;
        next OUTLOOP;
    }
    $weaving_suspended = 0;
    next OUTLOOP;
}
if ($weaving_suspended == 1) { next OUTLOOP; }
```
This code is used in §1.

§**6.** And we skip some material used only for compiling tables:

⟨Grammar and index entries are collated elsewhere, not woven in situ 6⟩ ≡
```
if ($line_category[$i] == $GRAMMAR_LCAT) { next OUTLOOP; }
if ($line_category[$i] == $GRAMMAR_BODY_LCAT) { next OUTLOOP; }
if ($line_category[$i] == $INDEX_ENTRY_LCAT) { next OUTLOOP; }
```
This code is used in §1.

§**7.** And lastly we ignore commands, or act on them if they happen to be aimed at us; but we don't weave them into the output, that's for sure.

⟨Respond to any commands aimed at the weaver, and otherwise skip commands 7⟩ ≡
```
if ($line_category[$i] == $COMMAND_LCAT) {
    my $argument = $line_operand_2[$i];
    if ($line_operand[$i] == $PAGEBREAK_CMD) {
        print WEAVEOUT "\\vfill\\eject\n";
    }
    if ($line_operand[$i] == $BNF_GRAMMAR_CMD) {
        weave_bnf_grammar($argument);
        print WEAVEOUT "\\smallbreak\n";
        print WEAVEOUT "\\hrule\\smallbreak\n";
    }
    if ($line_operand[$i] == $THEMATIC_INDEX_CMD) {
        weave_thematic_index($argument);
        print WEAVEOUT "\\smallbreak\n";
        print WEAVEOUT "\\hrule\\smallbreak\n";
    }
    if ($line_operand[$i] == $FIGURE_CMD) {
        ⟨Weave a figure 8⟩;
    }
    Otherwise assume it was a tangler command, and ignore it here
    next OUTLOOP;
}
```
This code is used in §1.

§**8.** TEX itself has an almost defiant lack of support for anything pictorial, which is one reason it didn't live up to its hope of being the definitive basis for typography; even today the loose confederation of TEX-like programs and extensions lack standard approaches. Here we're going to use `pdftex` features, having nothing better. All we're trying for is to insert a picture, scaled to a given width, into the text at the current position.

⟨Weave a figure 8⟩ ≡

```
my $figname = $argument;
my $width = "";
if ($figname =~ m/^(\d+)cm\: (.*)$/) {
    $figname = $2;
    $width = " width ".$1."cm";
}
print WEAVEOUT "\\pdfximage".$width."{../Figures/".$figname, "}\n";
print WEAVEOUT "\\smallskip\\noindent",
    "\\hbox to\\hsize{\\hfill\\pdfrefximage \\pdflastximage\\hfill}",
    "\\smallskip\n";
```

This code is used in §7.

§**9. Headings.** The purpose is set with a little heading. Its operand is that part of the purpose-text which is on the opening line; the rest follows on subsequent lines until the next blank.

⟨Weave the Purpose marker as a little heading 9⟩ ≡

```
if ($line_category[$i] == $PURPOSE_LCAT) {
    print WEAVEOUT "\\smallskip\\par\\noindent{\\it Purpose}\\smallskip\\noindent\n";
    print WEAVEOUT $line_operand[$i], "\n";
    $show_section_toc_soon = 1;
    $horizontal_rule_just_drawn = 0;
    next OUTLOOP;
}
```

This code is used in §1.

§**10.** This normally appears just after the Purpose subheading:

⟨If we need to work in a section table of contents and this is a blank line, do it now 10⟩ ≡

```
if (($show_section_toc_soon == 1) && ($line_text[$i] =~ m/^\s*$/)) {
    print WEAVEOUT "\\medskip";
    $show_section_toc_soon = 0;
    if ($section_toc[$line_sec[$i]] ne "") {
        print WEAVEOUT "\\smallskip\\hrule\\smallskip\\par\\noindent{\\usagefont ",
            $section_toc[$line_sec[$i]], "}\\par\\medskip\\hrule\\bigskip\n";
        $horizontal_rule_just_drawn = 1;
    } else {
        $horizontal_rule_just_drawn = 0;
    }
}
```

This code is used in §1.

§**11.** After which we have the Interface, except that this is skipped entirely, as far as weaving is concerned, unless (a) there's a body to it, and (b) we are in C-for-Inform mode, in which case the body is ignored anyway but a table of I6 template invocations of the section's functions is set instead:

⟨Deal with the Interface passage 11⟩ ≡

```
if ($line_category[$i] == $INTERFACE_LCAT) {
    $interface_table_pending = 1;
    next OUTLOOP;
}
if ($line_category[$i] == $INTERFACE_BODY_LCAT) {
    if ($interface_table_pending) {
        $interface_table_pending = 0;
        if ($web_language == $C_FOR_INFORM_LANGUAGE) {
            $horizontal_rule_just_drawn = 0;
            weave_interface_table_for_section($line_sec[$i]);
        }
    }
    next OUTLOOP;
}
```

This code is used in §1.

§**12.** And another little heading...

⟨Weave the Definitions marker as a little heading 12⟩ ≡

```
if ($line_category[$i] == $DEFINITIONS_LCAT) {
    print WEAVEOUT "\\smallskip\\par\\noindent{\\it Definitions}\\smallskip\\noindent\n";
    $next_heading_without_vertical_skip = 1;
    $horizontal_rule_just_drawn = 0;
    next OUTLOOP;
}
```

This code is used in §1.

§**13.** ...with the section bar to follow. The bar line completes any half-finished paragraph and is set as a horizontal rule:

⟨Weave the section bar as a horizontal rule 13⟩ ≡

```
if ($line_category[$i] == $BAR_LCAT) {
    ⟨Complete any started but not-fully-woven paragraph 36⟩;
    $functions_in_par = "";
    $structs_in_par = "";
    $current_macro_definition = "";
    $within_TeX_beginlines = 0;
    $next_heading_without_vertical_skip = 1;
    if ($horizontal_rule_just_drawn == 0) {
        print WEAVEOUT "\\par\\medskip\\noindent\\hrule\\medskip\\noindent\n";
    }
    next OUTLOOP;
}
```

This code is used in §1.

§14. Commentary matter. Typographically this is a fairly simple business, since the commentary is already by definition written in TEX format: it's almost the case that we only have to transcribe it. But not quite! This is where we implement the convenient additions `inweb` makes to TEX syntax.

⟨Weave verbatim matter in commentary style 14⟩ ≡

```
⟨Weave displayed source in its own special style 15⟩;
⟨Weave a blank line as a thin vertical skip and paragraph break 16⟩;
⟨Weave bracketed list indications at start of line into indentation 17⟩;
⟨Weave tabbed code material as a new indented paragraph 18⟩;
print WEAVEOUT $matter, "\n";
next OUTLOOP;
```

This code is used in §1.

§15. Displayed source is the material marked with >> arrows in column 1.

⟨Weave displayed source in its own special style 15⟩ ≡

```
if ($line_category[$i] == $SOURCE_DISPLAY_LCAT) {
    print WEAVEOUT "\\quotesource{", $line_operand[$i], "}\n";
    next OUTLOOP;
}
```

This code is used in §14.

§16. Our style is to use paragraphs without initial-line indentation, so we add a vertical skip between them to show the division more clearly.

⟨Weave a blank line as a thin vertical skip and paragraph break 16⟩ ≡

```
if ($matter =~ m/^\s*$/) {
    print WEAVEOUT "\\smallskip\\par\\noindent%\n";
    next OUTLOOP;
}
```

This code is used in §14.

§17. Here our extension is simply to provide a tidier way to use TEX's standard `\item` and `\itemitem` macros for indented list items.

⟨Weave bracketed list indications at start of line into indentation 17⟩ ≡

```
if ($matter =~ m/^\(\.\.\.\)\s+(.*?)$/) {                          continue single
    $matter = '\item{}'.$1;
} elsif ($matter =~ m/^\(-\.\.\.\)\s+(.*?)$/) {                    continue double
    $matter = '\itemitem{}'.$1;
} elsif ($matter =~ m/^\(([a-z0-9A-Z\.]+)\)\s+(.*?)$/) {            begin single
    $matter = '\item{('.$1.')}'.$2;
} elsif ($matter =~ m/^\(-([a-z0-9A-Z\.]+)\)\s+(.*?)$/) {          begin double
    $matter = '\itemitem{('.$1.')}'.$2;
}
```

This code is used in §14.

§**18.** Finally, matter encased in vertical strokes one tab stop in from column 1 in the source is set indented in code style.

⟨Weave tabbed code material as a new indented paragraph 18⟩ ≡
```
if ($matter =~ m/^\t\|/) {
    $matter = '\par\noindent\qquad '.$matter;
}
```
This code is used in §14.

§**19. Code-like matter.** Even though `inweb`'s approach, unlike `CWEB`'s, is to respect the layout of the original, this is still quite typographically complex: commentary and macro usage is rendered differently.

⟨Weave verbatim matter in code style 19⟩ ≡
```
⟨Enter beginlines/endlines mode if necessary 20⟩;
⟨Weave a blank line as a thin vertical skip 21⟩;
my $tab_stops_of_indentation = 0;
⟨Convert leading space in line matter to a number of tab stops 22⟩;
⟨Weave a suitable horizontal advance for that many tab stops 23⟩;
my $concluding_comment = "";
⟨Extract any comment matter ending the line to be set in italic 24⟩;
⟨Encase the code matter within vertical strokes 25⟩;
⟨Give constant definition lines slightly fancier openings 26⟩;
⟨Detect any structure or function definitions being woven in this paragraph 27⟩;
⟨Typeset the CWEB-style macros with cute highlighting and PDF links 28⟩;
⟨Insert continuation line breaks to cope with very long lines 29⟩;
print WEAVEOUT $matter, $concluding_comment, "\n";
next OUTLOOP;
```
This code is used in §1.

§**20.** Code is typeset between the `\beginlines` and `\endlines` macros in TEX, so:

⟨Enter beginlines/endlines mode if necessary 20⟩ ≡
```
if ($within_TeX_beginlines == 0) {
    print WEAVEOUT "\\beginlines\n"; $within_TeX_beginlines = 1;
}
```
This code is used in §19.

§**21.** A blank line is typeset as a thin vertical skip (no TEX paragraph break is needed):

⟨Weave a blank line as a thin vertical skip 21⟩ ≡
```
if ($matter =~ m/^\s*$/) {
    print WEAVEOUT "\\smallskip\n";
    next OUTLOOP;
}
```
This code is used in §19.

§**22.** Examine the white space at the start of the code line, and count the number of tab steps of indentation, rating 1 tab = 4 spaces:

⟨Convert leading space in line matter to a number of tab stops 22⟩ ≡
```
my $spaces_in = 0;
while ($matter =~ m/^(\s)(.*)$/) {
    $matter = $2;
    $whitespace = $1;
    if ($whitespace eq "\t") {
        $spaces_in = 0;
        $tab_stops_of_indentation++;
    } else {
        $spaces_in++;
        if ($spaces_in == 4) {
            $tab_stops_of_indentation++;
            $spaces_in = 0;
        }
    }
}
```

This code is used in §19.

§**23.** We actually use horizontal spaces rather than risk using TEX's messy alignment system:

⟨Weave a suitable horizontal advance for that many tab stops 23⟩ ≡
```
my $i;
for ($i=0; $i<$tab_stops_of_indentation; $i++) {
    print WEAVEOUT "\\qquad";
}
```

This code is used in §19.

§**24.** Comments which run to the end of a line are set in italic type. If the only item on their lines, they are presented at the code tab stop; otherwise, they are set flush right.

⟨Extract any comment matter ending the line to be set in italic 24⟩ ≡
```
if (line_ends_with_comment($matter)) {
    if ($part_before_comment eq "") {
        $matter = $part_before_comment; my $commentary = $part_within_comment;
        $concluding_comment = "{\\ttninepoint\\it ".$commentary."}";
    } else {
        $matter = $part_before_comment; my $commentary = $part_within_comment;
        if ($commentary =~ m/^C\d+\S+$/) {
            $commentary = "Test with |".$commentary.".txt|";
        }
        $concluding_comment = "\\hfill\\quad {\\ttninepoint\\it ".$commentary."}";
    }
}
```

This code is used in §19.

§**25.** Code is typeset by TEX within vertical strokes; these switch a sort of typewriter-type verbatim mode on and off. To get an actual stroke, we must escape from code mode, escape it using a backslash `|`, then re-enter code mode once again:

⟨Encase the code matter within vertical strokes 25⟩ ≡
```
$matter =~ s/\|/\|\\\|\|/g;
$matter = '|'.$matter.'|';
```
This code is used in §19.

§**26.** Set the `@d` definition escape very slightly more fancily (remembering that we are now encased in verticals):

⟨Give constant definition lines slightly fancier openings 26⟩ ≡
```
$matter =~ s/^\|\@d /{\\ninebf define}| /;
```
This code is used in §19.

§**27.** We note any structure typedefs, and also any functions which are called from outside this section, whose names we typeset in red. (We do this so that the endnotes can be added at the foot of the paragraph.)

⟨Detect any structure or function definitions being woven in this paragraph 27⟩ ≡
```
if ($matter =~ m/^\|\s*typedef\s+struct\s+(.*?)\s+\{/) {
    $structs_in_par .= $1.",";
}
if ($matter =~ m/^\|\/(\*+)\/\s*(.*?\**)(\S+?)\((.*)$/) {
    my $fstars = $1;
    my $ftype = $2;
    my $fname = $3;
    my $frest = $4;
    $matter = '|'.$ftype.'|\pdfliteral direct{0 1 1 0 k}|'.$fname.'|\special{PDF:0 g}|'.
        '('.$frest;
    $functions_in_par .= $fname.",";
}
if (($tab_stops_of_indentation == 0) &&
    ($matter =~ m/^\|(\S.*?\**)(\S+?)\((.*)$/)) {
    my $ftype = $1;
    my $fname = $2;
    my $frest = $3;
    my $unamended = $fname;
    $fname =~ s/::/__/g;
    if (exists($functions_line{$fname})) {
        $matter = '|'.$ftype.'|\pdfliteral direct{0 1 1 0 k}|'.$unamended.
            '|\special{PDF:0 g}|'.'('.$frest;
        $functions_in_par .= $fname.",";
    }
}
```
This code is used in §19.

§**28.** Any usage of angle-macros is highlighted in several cute ways: first, we make use of colour and we drop in the paragraph number of the definition of the macro in small type –

⟨Typeset the CWEB-style macros with cute highlighting and PDF links 28⟩ ≡

```
while ($matter =~ m/^(.*?)\@\<(.*?)\@\>(.*)$/) {
    my $left = $1;
    my $right = $3;
    my $href = $2.' {\sevenss '.$cweb_macros_paragraph_number{$2}.'}';
    my $angled_sname = $2;
    my $xrefcol = '\pdfliteral direct{1 1 0 0 k}';
    my $blackcol = '\special{PDF:0 g}';
    my $tweaked = '$\langle${\xreffont'.$xrefcol.$href.'}'.$blackcol.'$\rangle$';
    if ($right =~ m/^\s*\=(.*)$/) ⟨This is where the angle-macro is defined 30⟩
    else ⟨This is a reference to an angle-macro, not its definition 31⟩;
    $matter = $left.'|'.$tweaked.'|'.$right;
}
```

This code is used in §19.

§**29.** Overlong lines of code are liable to cause overfull hbox errors, the bane of all TEX users.

⟨Insert continuation line breaks to cope with very long lines 29⟩ ≡

```
my $count = 0;
my $code_matter = 0;
my $done = "";
my $number_of_splits = 0;
my $last_char = "";
while ($matter =~ m/^(.)(.*?)$/) {
    my $next_char = $1; $matter = $2;
    if (($next_char eq "|") && ($last_char ne "\\")) { $code_matter = 1 - $code_matter; }
    elsif ($code_matter == 1) {
        $count++;
        if ($count > 99) {
            $done = $done."| \n\\quad ...| "; $number_of_splits++; $count = 10;
        }
    }
    $done = $done.$next_char;
    $last_char = $next_char;
}
$matter = $done;
```

This code is used in §19.

§**30.** We mark this as a hyperlink destination (using `pdftex` extensions again):

⟨This is where the angle-macro is defined 30⟩ ≡

```
$right = $1;
$tweaked = '\pdfdest num '.link_for_par_name($angled_sname).' fit '.$tweaked;
$tweaked = $tweaked.' $\equiv$ ';
$current_macro_definition = $angled_sname;
```

This code is used in §28.

§**31.** And this is a link:

⟨This is a reference to an angle-macro, not its definition 31⟩ ≡

```
my $new_p = $line_paragraph_number[$i];
if (not($angled_paragraph_usage{$angled_sname} =~ m/$new_p,$/)) {
    $angled_paragraph_usage{$angled_sname} .= $new_p.',';
}
$tweaked =
    '\pdfstartlink attr{/C [0.9 0 0] /Border [0 0 0]} '.
    'goto num '.link_for_par_name($angled_sname).' '.
    $tweaked.'\pdfendlink';
```

This code is used in §28.

§32. How paragraphs begin.

⟨Deal with the marker for the start of a new paragraph, section or chapter 32⟩ ≡

```
if ($line_category[$i] == $PARAGRAPH_START_LCAT) {
    ⟨Complete any started but not-fully-woven paragraph 36⟩;
    ⟨If this is a paragraph break forced onto a new page, then throw a page 33⟩;
    my $weight = $line_operand[$i];
    my $para_title = $line_operand_2[$i];
    my $new_chap_num = $section_chap[$line_sec[$i]];
    my $secnum = $line_sec[$i];
    my $parnum = $line_paragraph_number[$i];
    my $ornament = $line_paragraph_ornament[$i];
    my $mark = "";
    ⟨Work out the next mark to place into the TEX vertical list 34⟩;
    $functions_in_par = "";
    $structs_in_par = "";
    $current_macro_definition = "";
    my $TeX_macro = "";
    ⟨Choose which TEX macro to use in order to typeset the new paragraph heading 35⟩;
    print WEAVEOUT "\\".$TeX_macro, "{", $parnum, "}{", $para_title, "}{",
        $mark, "}{", $ornament, "}{", $section_sigil[$secnum], "}%\n";
    There's quite likely ordinary text on the line following the paragraph
    start indication, too, so we need to weave this out:
    if ($line_text[$i] ne "") { print WEAVEOUT $line_text[$i], "\n"; }
    Chapter headings get a chapter title page, or possibly pages, too:
    if ($weight == 3) {
        my $sigil = $chapter_sigil[$section_chap[$line_sec[$i]]];
        print WEAVEOUT $chapter_rubric{$sigil}, "\\medskip\n";
        my $sn;
        for ($sn=0; $sn<$no_sections; $sn++) {
            if (not($section_sigil[$sn] =~ m/^$sigil\//)) { next; }
            print WEAVEOUT "\\smallskip\\noindent |", $section_sigil[$sn], "|: ",
                "{\\it ", $section_leafname[$sn], "}\\quad\n";
            print WEAVEOUT $section_purpose[$sn];
        }
    }
    And that completes the new paragraph opening.
    next OUTLOOP;
}
```

This code is used in §1.

§33. A few paragraphs are marked `@pp` and forced to begin on a new page:

⟨If this is a paragraph break forced onto a new page, then throw a page 33⟩ ≡

```
if ($line_starts_paragraph_on_new_page[$i]) {
    print WEAVEOUT "\\vfill\\eject\n";
}
```

This code is used in §32.

§**34.** "Marks" are the vile contrivance by which TEX produces running heads on pages which follow the material on those pages: so that the running head for a page can show the paragraph sigil for the material which tops it, for instance.

The ornament has to be set in math mode, even in the mark. § and ¶ only work in math mode because they abbreviate characters found in math fonts but not regular ones, in TEX's slightly peculiar font encoding system.

define $DEBUG_MARKING 0

⟨Work out the next mark to place into the TEX vertical list 34⟩ ≡

```
if ($weight == 3) { $chaptermark = $para_title; $sectionmark = ""; }
if ($weight == 2) {
    $sectionmark = $para_title;
    if ($section_sigil[$secnum] ne "") { $chaptermark = $section_sigil[$secnum]; }
    if ($chaptermark ne "") { $sectionmark = " - ".$sectionmark; }
}
$mark = $chaptermark.$sectionmark."\\quad\$".$ornament."\$".$parnum;
if ($DEBUG_MARKING == 1) {
    print "Start par $i: weight $weight, title <$para_title>, ",
        "chap $new_chap_num, sec $secnum, par $parnum, ornament $ornament\n";
    print "Mark: $mark\n";
}
```

This code is used in §32.

§**35.** We want to have different heading styles for different weights, and TEX is horrible at using macro parameters as function arguments, so we don't want to pass the weight that way. Instead we use

```
\weavesection
\weavesections
\weavesectionss
\weavesectionsss
```

where the weight is the number of terminal ss, 0 to 3. (TEX macros, lamentably, are not allowed digits in their name.) In the cases 0 and 1, we also have variants `\nsweavesection` and `\nsweavesections` which are the same, but with the initial vertical spacing removed; these allow us to prevent unsightly excess white space in certain configurations of a section.

⟨Choose which TEX macro to use in order to typeset the new paragraph heading 35⟩ ≡

```
my $j;
$TeX_macro = "weavesection";
for ($j=0; $j<$weight; $j++) { $TeX_macro = $TeX_macro."s"; }
if (($next_heading_without_vertical_skip == 1) && ($weight < 2)) {
    $next_heading_without_vertical_skip = 0;
    $TeX_macro = "ns".$TeX_macro;
}
if ($weight == 3) {                                    a chapter heading: note the inch-high sigil...
    my $brief_title = $para_title;
    $brief_title =~ s/^.*?\: //;
    $para_title = "{\\sinchhigh ".$chapter_sigil[$section_chap[$line_sec[$i]]].
        "}\\quad ".$brief_title;
}
```

This code is used in §32.

§**36. How paragraphs end.** At the end of a paragraph, on the other hand, we do this:

⟨Complete any started but not-fully-woven paragraph 36⟩ ≡

```
    if ($within_TeX_beginlines > 0) {
        print WEAVEOUT '\endlines', "\n"; $within_TeX_beginlines = 0;
    }
    show_endnotes_on_previous_paragraph($functions_in_par, $structs_in_par,
        $current_macro_definition);
```

This code is used in §1,13,32.

§**37.** The endnotes describe function calls from far away, or unexpected structure usage, or how CWEB-style code substitutions were made.

```
sub show_endnotes_on_previous_paragraph {
    my $functions_in_par = $_[0];
    my $structs_in_par = $_[1];
    my $current_macro_definition = $_[2];
    if ($current_macro_definition ne "")
        ⟨Weave endnote saying which other paragraphs use this one 38⟩
    else {
        my %fnames_done = ();
        while ($functions_in_par =~ m/^(\S+?)\,(.*)$/) {
            my $fname = $1;
            $functions_in_par = $2;
            if ($fnames_done{$fname}++ > 0) { next; }
            if ($fname eq "isdigit") { next; }
            ⟨Weave endnote saying where this function is called from 39⟩;
        }
        while ($structs_in_par =~ m/^(\S+?)\,(.*)$/) {
            my $sname = $1;
            my $sbilling;
            $structs_in_par = $2;
            if ($structure_ownership_summary{$sname} ne "") {
                $sbilling = $structure_ownership_summary{$sname};
                $sbilling =~ s/ /, /g;
                $sbilling =~ s/, $//;
                $sbilling =~ s/, (\S+)$/ and $1/;
                $sbilling = "shared with ".$sbilling;
            } else {
                $sbilling = "private to this section";
            }
            $sname =~ s/_/\\_/g;
            print WEAVEOUT "\\par\\noindent";
            print WEAVEOUT "\\penalty10000\n";
            $sname =~ s/\#/\\#/g;
            print WEAVEOUT "{\\usagefont The structure $sname is $sbilling.}\n";
        }
        print WEAVEOUT "\\smallskip\n";
    }
}
```

§**38.** We try hard to prevent a page break between paragraph and endnote (hence the high `\penalty` value). Sometimes this succeeds.

⟨Weave endnote saying which other paragraphs use this one 38⟩ ≡

```
my $used_in = $angled_paragraph_usage{$current_macro_definition};
$used_in =~ s/\,$//;
print WEAVEOUT "\\penalty1000\n";
print WEAVEOUT "\\noindent{\\usagefont This code is used in \$\\S\$",
    $used_in, ".}\\smallskip\n";
```

This code is used in §37.

§**39.** And similarly:

⟨Weave endnote saying where this function is called from 39⟩ ≡

```
my $scope = $functions_declared_scope{$fname};
my @usages;
my $no_sections_using = 0;
⟨Work out which sections to mention calls from 40⟩;
my $fname_with_underscores_escaped = $fname;
$fname_with_underscores_escaped =~ s/__/::/g;
$fname_with_underscores_escaped =~ s/_/\\_/g;
print WEAVEOUT "\\par\\noindent";
print WEAVEOUT "\\penalty10000\n";
print WEAVEOUT "{\\usagefont The function $fname_with_underscores_escaped is";
my $clause_begins = 0;
if ($scope eq "*****") {
    print WEAVEOUT " where execution begins"; $clause_begins = 1;
}
if ($scope eq "****") {
    print WEAVEOUT " invoked by a command in a |.i6t| template file"; $clause_begins = 1;
}
if (($clause_begins == 1) || (($scope eq "***") || ($scope eq "**"))) {
    if ($no_sections_using > 0) {
        if ($clause_begins) { print WEAVEOUT " and"; }
        print WEAVEOUT " called from ";
        my $x;
        for ($x=0; $x<$no_sections; $x++) {
            if ($usages[$x] == 1) {
                print WEAVEOUT $section_sigil[$x];
                if ($no_sections_using > 2) { print WEAVEOUT ", "; }
                if ($no_sections_using == 2) { print WEAVEOUT " and "; }
                $no_sections_using--;
            }
        }
    }
}
print WEAVEOUT ".}\n";
```

This code is used in §37.

§**40.** Loop through the concise string holding this information:

⟨Work out which sections to mention calls from 40⟩ ≡

```
my $x;
for ($x=0; $x<$no_sections; $x++) { $usages[$x] = 0; }
my $cp = $functions_usage_concisely_described{$fname};
while ($cp =~ m/^\:(\d+)(.*?)$/) { $usages[eval($1)] = 1; $cp = $2; }
for ($x=0; $x<$no_sections; $x++) { if ($usages[$x] == 1) { $no_sections_using++; } }
```

This code is used in §39.

§**41. The cover sheet.** This is added to the front of larger PDFs; whole chapters and the complete work. People will probably want to customise this, so it's implemented as a TEX file which we substitute bibliographic data into.

```
sub weave_cover_sheet {
    my $cover_sheet = $path_to_inweb_setting.'inweb/Materials/cover-sheet.tex';
    if (exists ($bibliographic_data{"Cover Sheet"})) {
        $cover_sheet = $web_setting.'Materials/'.$bibliographic_data{"Cover Sheet"};
    }
    $bibliographic_data{"Capitalized Title"} = $bibliographic_data{"Title"};
    $bibliographic_data{"Capitalized Title"} =~ tr/a-z/A-Z/;
    $bibliographic_data{"Booklet Title"} = $booklet_title;
    weave_cover_from($cover_sheet);
}
```

§**42.** Note that the substitution `[[Cover Sheet]]` embeds the whole default cover sheet; this is so that we can just add a continuation of that, if we want to.

```
sub weave_cover_from {
    my $cs_filename = $_[0];
    local *COVER;
    open(COVER, $cs_filename) or die "inweb: cannot find cover sheet file";
    my $csl;
    while ($csl = <COVER>) {
        $csl =~ m/^(.*?)\s*$/; $csl = $1;
        if ($csl =~ m/^\%/) { next; }                    a TeX comment begins with a percent sign
        while ($csl =~ m/^(.*?)\[\[(.*?)\]\](.*)$/) {
            print WEAVEOUT $1; my $expander = $2; $csl = $3;
            if ($expander eq "Cover Sheet") {
                my $insert_filename =
                    $path_to_inweb_setting.'inweb/Materials/cover-sheet.tex';
                if ($insert_filename ne $cs_filename) {
                    weave_cover_from($insert_filename);
                }
            } elsif (exists($bibliographic_data{$expander})) {
                print WEAVEOUT $bibliographic_data{$expander};
            } else {
                print WEAVEOUT $expander;
            }
        }
        print WEAVEOUT $csl, "\n";
    }
```

```
    close (COVER);
}
```

§**43. Table of I6 template interpreter invocations.** This is only used in C-for-Inform mode, and shows four-star functions – the ones called by the I6T interpreter.

```
sub weave_interface_table_for_section {
    my $sect_no = $_[0];
    my $j;
    my $red_text = "\|\\pdfliteral direct\{0 1 1 0 k\}\|";
    my $blue_text = "\|\\pdfliteral direct\{1 1 0 0 k\}\|";
    my $black_text = "\|\\special{PDF:0 g}\|";
    my $fname;
    my %functions_in_order = ();                          a device used to sort functions in definition order
    foreach $fname (keys %functions_line) {
        $functions_in_order{sprintf("%07d", $functions_line{$fname})} = $fname;
    }
    my $heading_made = 0;
    foreach $fnamed (sort keys %functions_in_order) {
        $fname = $functions_in_order{$fnamed};
        my $j = $functions_line{$fname};
        if ($line_sec[$j] != $sect_no) { next; }
        if ($functions_declared_scope{$fname} ne "****") { next; }
        if ($heading_made == 0) {
            $heading_made = 1;
            print WEAVEOUT "\\bigbreak\\par\\noindent",
                "{\\it Template interpreter commands}\\smallskip\n";
        }
        print WEAVEOUT "\\par\\noindent\n";
        print WEAVEOUT "{\\sevenss", $line_paragraph_number[$j], "}|  |\n";
        $spc = $functions_arglist{$fname};
        $spc =~ s/\,\s*/\,\n\t\t/g;
        $access = 'callv';
        if ($spc =~ m/FILE \*/) { $access = 'call'; }
        if ($fname =~ m/^(.*)_array$/) { $fname = $1; $access = 'array'; }
        if ($fname =~ m/^(.*)_routine$/) { $fname = $1; $access = 'routine'; }
        $fname =~ s/__/::/g;
        print WEAVEOUT '|{-', $access, ':', $blue_text, $fname, $black_text, "}|\n";
    }
    if ($heading_made == 1) {
        print WEAVEOUT "\\bigbreak\\noindent\n";
    }
}
```

§**44. Thematic Index.** Produced in response to explicit weaver commands (and otherwise never visible, so in a web which doesn't use indexing, this will never have any effect).

```
sub weave_thematic_index {
    my $wanted= $_[0];
    foreach $subject (sort keys %thematic_indices) {
        my $chunk;
        if (($wanted ne "Remainder") && ($wanted ne $subject)) { next; }
        if ($index_subject_done{$subject}) { next; }
        $index_subject_done{$subject} = 1;
        print WEAVEOUT "{\\bf{", $subject, "}}\\par\n\\smallbreak\\hrule\\smallbreak\n";
        print WEAVEOUT ;
        $chunk = $thematic_indices{$subject};
        while ($chunk =~ m/^(.*?)\.\.\.(\d+)\|(.*)$/) {
            my $entry = $1;
            my $at = $2;
            $chunk = $3;
            print WEAVEOUT "\\line{", $entry,
                "\\leaders\\hbox to 1em{\\hss.\\hss}\\hfill {\\xreffont ",
                $line_csn[eval($at)], "}}\n";
        }
    }
}
```

§**45. PDF links.** Hypertext links within a PDF document are now a pretty well-established part of the PDF specification: we don't set any that point outside it. TEX of course cannot generate such details, but `pdftex` and other more modern forms can.

```
sub begin_making_pdf_links {
    %link_number_for_par_name = ();
    $no_par_name_links = 1;
}
sub link_for_par_name {
    my $par_name = $_[0];
    if (not(exists $link_number_for_par_name{$par_name})) {
        $link_number_for_par_name{$par_name} = $no_par_name_links++;
    }
    return $link_number_for_par_name{$par_name};
}
```