# The Parser 2/parse

*Purpose*

To work through the program read in, assigning each line its category, and noting down other useful information as we go.

---



---

*Definitions*

**¶1.** Like the earlier reading-in stage, the parsing stage always happens in every run of `inweb`, and it happens once only. We use the arrays already built to represent the lines, sections and chapters, and we go on to create new ones, as follows. We know everything about chapters already. However, for the sections we add:

(S5) `$section_sigil[]`, which is the sigil such as `4/fish` identifying the section briefly, and is found on the titling line before the colon. (The hash `$sigil_section{}` provides a convenient inverse to this.) Similarly, `$section_namespace[]` is the optional namespace quoted.

(S6) `$section_toc[]`, which is the text of the brief table of contents of the section, in TEX marked-up form. (The weaver adds this as a headnote when setting the section.)

(S7) `$section_purpose[]`, the text of the purpose given by `@Purpose:`.

(S8) `$section_no_pars[]`, the number of paragraphs in the section.

**¶2.** And for the lines we add:

(L5) `$line_category[]`, which assigns every line one of the `*_LCAT` values (see the Line Categories section for details): when parsing is complete, no line is permitted still to have category `$NO_LCAT`.

(L6) `$line_operand[]`: the type and value of this depends on the line category, but basically it supplies some useful information about what the line constructs.

(L7) `$line_operand_2[]` is a second, even more optional operand.

(L8) `$line_paragraph_number[]` is the paragraph number within the section to which the line belongs (or 0, if it is somewhere at the top before paragraphs have begun). This doesn't entirely specify the paragraph, though, as there might be a paragraph 2 in Definitions above the bar and then another in the code stuff below: so we must also record the

(L9) `$line_paragraph_ornament[]` which is the TEX for the paragraph "ornament", a ¶ or § sign, according to whether we're above or below the bar.

(L10) `$line_starts_paragraph_on_new_page[]` which is simply a flag used with paragraph headings to cause a page throw, or not.

(L11) `$line_is_comment[]` is set if the weaver should treat the line as regular-type commentary, and clear if it should treat it as verbatim program code or other such quoted matter. (We could deduce this from the line category, but it's more convenient not to have to.)

(L12) `$line_occurs_in_CWEB_macro_definition[]` is a flag needed by the tangler.

**¶3.** Command-category lines set their (first) operand to one of the following:

```
define $PAGEBREAK_CMD 1
define $BNF_GRAMMAR_CMD 2
define $THEMATIC_INDEX_CMD 3
define $FIGURE_CMD 4
define $INDEX_UNDER_CMD 5
```

**¶4.** With the parsing done, we discover the remaining layer of structure within the web: the paragraphs. The following count variable exists only for the sake of the statistics line printed out at the end of `inweb`'s run: it counts the number of marked paragraphs in each section, plus a notional 1 for the header part of the section (titling, purpose, interface, grammar, etc.).

```
$no_paragraphs = 0;
```

**¶5.** Each paragraph has a "weight", which is a measure of its typographic prominence. The paragraph weights are:

(0) an ordinary paragraph with no subheading
(1) paragraph with bold label (operand 2: the title)
(2) paragraph with a subheading (operand 2: the title)
(3) paragraph with a chapter heading (operand 2: the title)

**¶6.** Some paragraphs define `CWEB` macros in angled brackets, and those need special handling. We parse the following data on them, using hashes keyed by the name of the macro without angle brackets attached:

(W1) `$cweb_macros_paragraph_number{}` is the paragraph number of the definition, as printed in small type after the name in any usage.
(W2) `$cweb_macros_start{}` is the first line number of the definition body.
(W3) `$cweb_macros_end{}` is the last.
(W4) `$cweb_macros_surplus_bit{}` is the possible little first fragment of macro definition on the same line as the declaration opens, following the equals sign. (It's poor style to write macros this way, I think, but for better compatibility with `CWEB`...)

**¶7.** We also look out for Backus-Naur form grammar in the program, collating it all together. (Obviously, this was written specifically for the Inform documentation, but a lot of programs have grammars of one kind or another, so it might as well be available for any project.) At this parsing stage, collation is all we do. The grammar lines are numbered 0 to `$bnf_grammar_lines` minus 1, and we have arrays:

(G1) `$bnf_grammar[]` holding the text of the line,
(G2) `$bnf_grammar_source[]` holding the line number it came from in the web.

§**1. Sequence of parsing.** As can be seen, we work in two passes. It could all be done in one if we needed to, but this is plenty fast enough, is tidier and makes the code simpler.

```
sub parse_literate_source {
    ⟨Initialise the new arrays created in parsing phase 2⟩;
    determine_line_categories();                                    Pass 1
    establish_canonical_section_names();                            Pass 2
    ⟨Count the number of paragraphs 3⟩;
}
```

§**2.** See Definitions above.

⟨Initialise the new arrays created in parsing phase 2⟩ ≡

```
    my $i;
    for ($i=0; $i<$no_lines; $i++) {
        $line_is_comment[$i] = 0;
        $line_category[$i] = $NO_LCAT;
        $line_paragraph_number[$i] = 0;
        $line_paragraph_ornament[$i] = "";
        $line_starts_paragraph_on_new_page[$i] = 0;
    }
    for ($i=0; $i<$no_sections; $i++) {
        $section_sigil[$i] = "";
        $section_toc[$i] = "";
        $section_purpose[$i] = "";
        $section_no_pars[$i] = 0;
    }
```

This code is used in §1.

§**3.**

⟨Count the number of paragraphs 3⟩ ≡

```
    my $i;
    $no_paragraphs = 0;
    for ($i=0; $i<$no_lines; $i++) {
        if ($line_category[$i] == $PARAGRAPH_START_LCAT) {
            $no_paragraphs++;
        }
    }
```

This code is used in §1.

§4. **First parse.** On the first run through, we assign a category to every line of the source, and set operands as appropriate. We also note down the `$section_purpose[]` and `$section_sigil[]` as we pass.

```
sub determine_line_categories {
    $comment_mode = 1;
    $grammar_mode = 0;
    CATEGORISATION: for ($i=0; $i<$no_lines; $i++) {
        $line_is_comment[$i] = $comment_mode;
        $line_category[$i] = $COMMENT_BODY_LCAT;                    Until set otherwise down below
        my $l = $line_text[$i];
        if (line_is_in_heading_position($i)) {
            language_set($tangle_target_language[$section_tangle_target[$line_sec[$i]]]);
            ⟨Rewrite the heading in CWEB-style paragraph notation but continue 5⟩;
        }
        if ($l =~ m/^\[\[\s*(.*?)\s*\]\]\s*$/)
            ⟨Parse the line as an Inweb command 6⟩;
        if (language_pagebreak_comment($l)) {
            $line_category[$i] = $COMMAND_LCAT;
            $line_operand[$i] = $PAGEBREAK_CMD;
            next CATEGORISATION;
        }
        if ($l =~ m/^\s*\@\<\s*(.*?)\s*\@\>\s*\=\s*(.*?)$/)
            ⟨Note that a CWEB macro is defined here 7⟩;
        if ($l =~ m/^\@(\S*)(.*?)$/) {
            my $command = $1;
            my $remainder = $2;
            my $succeeded = 0;
            ⟨Parse and deal with a structural marker 8⟩;
            if ($succeeded == 0) {
                inweb_error_at_program_line("don't understand @".$command." at:", $i);
            }
        }
        if ($grammar_mode == 1) ⟨Store this line as part of the BNF grammar 15⟩;
        if ($comment_mode == 1) ⟨This is a line destined for the commentary 13⟩;
        if ($comment_mode == 0) ⟨This is a line destined for the verbatim code 14⟩;
    }
}
```

§**5.** In a convenient if faintly sordid manoeuvre, we rewrite a heading line – either a Chapter heading or the titling line of a section – as if it were a CWEB-style paragraph break of a superior kind (`@*`, which is super, or `@**` which is even more so). This code is a residue of the time when `inweb` was essentially a reimplementation of CWEB.

⟨Rewrite the heading in CWEB-style paragraph notation but continue 5⟩ ≡

```
if ($l =~ m/^(([A-Za-z0-9_]+::\s*)*)(\S+\/[a-z][a-z0-9]+)\:\s+(.*)\s*$/) {
    $section_namespace[$line_sec[$i]] = $1;
    $section_sigil[$line_sec[$i]] = $3;
    $sigil_section{$1} = $line_sec[$i];
    $l = '@* '.$4;
    $section_namespace[$line_sec[$i]] =~ s/\s*//g;
} else {
    if (($l =~ m/^Chapter /) || ($l =~ m/^Appendix /)) {
        $l = '@** '.$l;
    }
}
```

This code is used in §4.

§**6.** Note that we report an error if the command isn't one we recognise: we don't simply ignore the squares and let it fall through into the tangler.

⟨Parse the line as an Inweb command 6⟩ ≡

```
my $comm = $1;
$line_category[$i] = $COMMAND_LCAT;
if ($comm =~ m/^(.*?)\s*\:\s*(.*)\s*$/) {
    $comm = $1;
    $line_operand_2[$i] = $2;
}
if ($comm eq "Page Break") { $line_operand[$i] = $PAGEBREAK_CMD; }
elsif ($comm eq "BNF Grammar") { $line_operand[$i] = $BNF_GRAMMAR_CMD; }
elsif ($comm eq "Thematic Index") { $line_operand[$i] = $THEMATIC_INDEX_CMD; }
elsif ($comm =~ m/^Index Under (.*)$/) {
    $thematic_indices{$1} .= $line_operand_2[$i]."...".$i."|";
    $line_operand[$i] = $INDEX_UNDER_CMD;
}
elsif ($comm eq "Figure") { $line_operand[$i] = $FIGURE_CMD; }
else { inweb_error("unknown command $l\n"); }
$line_is_comment[$i] = 1;
```

This code is used in §4.

§**7.** These are woven and tangled much like other comment lines, but we need to notice the macro definition, of course:

⟨Note that a CWEB macro is defined here 7⟩ ≡

```
$line_category[$i] = $MACRO_DEFINITION_LCAT;
$line_operand[$i] = $1;                          The name of the macro
$line_operand_2[$i] = $2;                        Any beginning of its content on the same line
$comment_mode = 0;
$line_is_comment[$i] = 0;
$code_lcat_for_body = $CODE_BODY_LCAT;           Code follows on subsequent lines
next CATEGORISATION;
```

This code is used in §4.

§**8.** A structural marker is introduced by an `@` in column 1, and is a structural division in the current section. There are a number of possibilities. One, where the line is a macro definition, we have already dealt with.

There are some old `CWEB` syntaxes surviving here: `@1` is the same as `@`, `@2` the same as `@p`, `@3` the same as `@*` which is reserved (by us, though not by `CWEB`) for section headings, `@4` the same as `@**` which is reserved (ditto) for chapter headings.

In practice `@*` and `@**` headings do begin on new pages, but that is accomplished by the TEX macros which typeset them, so we don't need to force a page break ourselves: which is why we don't set the `$line_starts_paragraph_on_new_page[]` flag for those.

⟨Parse and deal with a structural marker 8⟩ ≡

```
if ($line_category[$i] == $MACRO_DEFINITION_LCAT) { $succeeded = 1; }
⟨Deal with structural markers above the bar 9⟩;
⟨Deal with the code and extract markers 10⟩;
⟨Deal with the define marker 11⟩;
my $weight = -1;
if ($command eq "") { $weight = 0; }
if ($command eq "p") { $weight = 1; }
if ($command eq "pp") { $weight = 1; $line_starts_paragraph_on_new_page[$i] = 1; }
if ($command eq "*") { $weight = 2; }
if ($command eq "**") { $weight = 2; }
if ($command eq "***") { $weight = 3; }
if ($command =~ m/\*(\d)/) { $weight = eval($1)-1; }
if ($weight >= 0) ⟨Begin a new paragraph of this weight 12⟩;
```

This code is used in §4.

§**9.** The five markers down as far as the bar:

⟨Deal with structural markers above the bar 9⟩ ≡

```
if ($command eq "Purpose:") {
    $line_category[$i] = $PURPOSE_LCAT;
    $line_operand[$i] = $remainder;
    $line_is_comment[$i] = 1;
    $section_purpose[$line_sec[$i]] = $remainder;
    $i2 = $i+1;
    while ($line_text[$i2] =~ m/[a-z]/) {
        $section_purpose[$line_sec[$i]] .= " ".$line_text[$i2];
        $i2++;
    }
    next CATEGORISATION;
}
if ($command eq "Interface:") {
    $line_category[$i] = $INTERFACE_LCAT;
    $line_is_comment[$i] = 1;
    $grammar_mode = 0;
    next CATEGORISATION;
}
if ($command eq "Grammar:") {
    $line_category[$i] = $GRAMMAR_LCAT;
    $line_is_comment[$i] = 1;
    $grammar_mode = 1;
    next CATEGORISATION;
}
```

```
    if ($command eq "Definitions:") {
        $line_category[$i] = $DEFINITIONS_LCAT;
        $line_is_comment[$i] = 1;
        $grammar_mode = 0;
        next CATEGORISATION;
    }
    if ($command =~ m/\-\-\-\-+/) {
        $line_category[$i] = $BAR_LCAT;
        $line_is_comment[$i] = 1;
        $command = "";
        $grammar_mode = 0;
        next CATEGORISATION;
    }
```

This code is used in §8.

§**10.** These have identical behaviour except for whether or not to tangle what follows:

⟨Deal with the code and extract markers 10⟩ ≡

```
    if (($command eq "c") || ($command eq "x")) {
        $line_category[$i] = $BEGIN_VERBATIM_LCAT;
        if ($command eq "x") { $code_lcat_for_body = $TEXT_EXTRACT_LCAT; }
        else { $code_lcat_for_body = $CODE_BODY_LCAT; }
        $comment_mode = 0;
        $succeeded = 1;
        $line_text[$i] = "";
    }
```

This code is used in §8.

§**11.** Definitions are intended to translate to C preprocessor macros, Inform 6 `Constant`s, and so on.

⟨Deal with the define marker 11⟩ ≡

```
    if ($command eq "d") {
        $line_category[$i] = $BEGIN_DEFINITION_LCAT;
        $code_lcat_for_body = $CONT_DEFINITION_LCAT;
        if ($remainder =~ m/^\s*(\S+)\s+(.+?)\s*$/) {
            $line_operand[$i] = $1;                          Name of term defined
            $line_operand_2[$i] = $2;                        Value
        } else {
            $line_operand[$i] = $remainder;                  Name of term defined
            $line_operand_2[$i] = "";                        No value given
        }
        $comment_mode = 0;
        $line_is_comment[$i] = 0;
        $succeeded = 1;
    }
```

This code is used in §8.

§**12.** The noteworthy thing here is the way we fool around with the text on the line of the paragraph opening. This is a hangover from the fact that we are using a line-based system (with significant `@`s in column 1 only) yet imitating `CWEB`, which is based on escape characters and sees only a stream where there's nothing special about line breaks. Anyway,

```
@p The chronology of French weaving. Auguste de Papillon (1734-56) soon
```

is split so that "The chronology of French weaving" is stored as operand 2 (the title) while the line is rewritten to read simply

```
Auguste de Papillon (1734-56) soon
```

so that it can go on to by woven or tangled exactly as the succeeding lines will be.

⟨Begin a new paragraph of this weight 12⟩ ≡

```
$grammar_mode = 0;
$comment_mode = 1;
$line_is_comment[$i] = 1;
$line_category[$i] = $PARAGRAPH_START_LCAT;
$line_operand[$i] = $weight;                                   Weight
$line_operand_2[$i] = "";                                      Title
if (($weight > 0) && ($remainder =~ m/\s+(.*?)\.\s*(.*)$/)) {
    $line_operand_2[$i] = $1;                                  Title up to the full stop
    $line_text[$i] = $2;                                       And then some regular material
} else {
    if ($remainder =~ m/\s+(.+?)$/) { $line_text[$i] = $1; }
    else { $line_text[$i] = ""; }
}
$succeeded = 1;
```

This code is used in §8.

§**13. Any last special rules for lines.** In commentary mode, we look for any fancy display requests, and also look to see if there are interface lines under an `@Interface:` heading (because that happens in commentary mode, though this isn't obvious).

⟨This is a line destined for the commentary 13⟩ ≡

```
if ($line_text[$i] =~ m/^\>\>\s+(.*?)\s*$/) {
    $line_category[$i] = $SOURCE_DISPLAY_LCAT;
    $line_operand[$i] = $1;
}
if (($web_language == $C_LANGUAGE) || ($web_language == $C_FOR_INFORM_LANGUAGE)) {
    if (scan_line_for_interface($i)) {
        $line_category[$i] = $INTERFACE_BODY_LCAT;
    }
}
```

This code is used in §4.

§**14.** Note that in an `@d` definition, a blank line is treated as the end of the definition. (This is unnecessary for C, and is a point of difference with `CWEB`, but is needed for languages which don't allow multi-line definitions.)

⟨This is a line destined for the verbatim code 14⟩ ≡

```
if (($web_language == $C_LANGUAGE) || ($web_language == $C_FOR_INFORM_LANGUAGE)) {
    scan_line_for_interface($i);
}
if (($line_category[$i] != $BEGIN_DEFINITION_LCAT) &&
    ($line_category[$i] != $COMMAND_LCAT)) {
    $line_category[$i] = $code_lcat_for_body;
}
if (($line_category[$i] == $CONT_DEFINITION_LCAT) &&
    ($line_text[$i] =~ m/^\s*$/)) {
    $line_category[$i] = $COMMENT_BODY_LCAT;
    $code_lcat_for_body = $COMMENT_BODY_LCAT;
}
if (language_and_so_on($line_text[$i])) {
    $line_category[$i] = $TOGGLE_WEAVING_LCAT;
}
if ($web_language == $C_FOR_INFORM_LANGUAGE)
    ⟨Detect some NI-specific oddities and throw them into the BNF grammar 16⟩;
```

This code is used in §4.

§**15. The BNF grammar.** Material underneath the optional `@Grammar:` heading is stored away thus for the time being:

⟨Store this line as part of the BNF grammar 15⟩ ≡

```
$line_category[$i] = $GRAMMAR_BODY_LCAT;
$bnf_grammar[$bnf_grammar_lines] = $l;
$bnf_grammar_source[$bnf_grammar_lines] = $i;
$bnf_grammar_lines++;
next CATEGORISATION;
```

This code is used in §4.

§**16.** It's convenient to auto-detect some grammar from specific code patterns in the source code to `NI`, though of course this mustn't be done for any other project:

⟨Detect some NI-specific oddities and throw them into the BNF grammar 16⟩ ≡

```
if ($l =~ m/the_debugging_aspects\[\] =/) {
    $bnf_grammar_source[$bnf_grammar_lines] = $i;
    $bnf_grammar[$bnf_grammar_lines++] = "<debugging-aspect>";
    $da_baseline = $i;
}
if (($l =~ m/^\s+\{\s+\"(.+?)\"\,\s+\"(.*?)\"\,\s+\"(.*?)\"/)
    && ($i < $da_baseline+200)) {
    $bnf_grammar[$bnf_grammar_lines] = "    := $1";
    if ($2 ne "") { $bnf_grammar[$bnf_grammar_lines] .= " $2"; }
    if ($3 ne "") { $bnf_grammar[$bnf_grammar_lines] .= " $3"; }
    $bnf_grammar_lines++;
}
```

This code is used in §14.

§**17. Second parse.** We work out the values of `$line_paragraph_number[]`, `$line_paragraph_ornament[]`, `$section_toc[]`, `$section_no_pars[]`; and we also create the hashes of details for the `CWEB` macro definitions.

```
sub establish_canonical_section_names {
    my $par_number_counter = 0;
    my $pnum = 0;
    my $ornament = "\\S";
    my $i;
    my $toc = "";
    my $toc_range_start = 0;
    my $toc_name = "";
    ⟨Begin a new section TOC 19⟩;
    for ($i=0; $i<$no_lines; $i++) {
        if (($line_category[$i] == $DEFINITIONS_LCAT) ||
            ($line_category[$i] == $PURPOSE_LCAT)) {
            $ornament = "\\P";
            $par_number_counter = 0;                          Start counting paras from 1
        }
        if ($line_category[$i] == $BAR_LCAT) {
            $ornament = "\\S";
            $par_number_counter = 0;                          Start counting paras from 1 again
            ⟨Complete any named paragraph range now half-done in the TOC 21⟩;
        }
        if ($line_category[$i] == $PARAGRAPH_START_LCAT) {
            $par_number_counter++;
            if ($line_operand[$i] == 2) {                     weight 2, so a section heading
                if ($line_sec[$i] > 0)                        i.e., if this isn't the first section
                    ⟨Complete the TOC from the section just ended 22⟩;
                ⟨Begin a new section TOC 19⟩;
            }
            $section_no_pars[$line_sec[$i]]++;
            if ($line_operand[$i] == 1)                       weight 1: a named @p or @pp paragraph
                ⟨Begin a new named paragraph range in the TOC 20⟩;
        }
        $line_paragraph_ornament[$i] = $ornament;
        $line_paragraph_number[$i] = $par_number_counter;
        if ($line_category[$i] == $MACRO_DEFINITION_LCAT)
            ⟨Record details of this CWEB macro 18⟩;
        $line_csn[$i] = $section_sigil[$line_sec[$i]].
            '.$'.$line_paragraph_ornament[$i].'$'.
            $line_paragraph_number[$i];
    }
    ⟨Complete the TOC from the very last section 23⟩;
}
```

§**18.** We now record the four hashes of data about the macro just found:

⟨Record details of this CWEB macro 18⟩ ≡
```
my $j = $i;
my $name = $line_operand[$i];
$cweb_macros_paragraph_number{$name} = $par_number_counter;
$cweb_macros_start{$name} = $j+1;
$cweb_macros_surplus_bit{$name} = $line_operand_2[$j];
$j++;
while (($j<$no_lines) && ($line_category[$j] == $CODE_BODY_LCAT)) {
    $line_occurs_in_CWEB_macro_definition[$j] = 1;
    $j++;
}
$cweb_macros_end{$name} = $j;
```

This code is used in §17.

§**19.** The table of contents (TOC) for a section is abbreviated, and we generate it sequentially, with the following mildly convoluted logic. The idea is to end up with something like "¶2-5. Blue cheese; ¶6. Danish blue; ¶7-11. Gouda; §1-3. Slicing the cheese; §4. Wrapping in greaseproof paper." We start writing a TOC entry when hitting a new named paragraph, and write just the "¶7" part; only when we hit the next named paragraph, or the bar dividing the section, or the end of the section, or the end of the whole web, do we complete it by adding "-11. Gouda".

⟨Begin a new section TOC 19⟩ ≡
```
$toc = ""; $toc_range_start = 0;
```

This code is used in §17.

§**20.** The first of the four ways we might have to complete a part-made entry.

⟨Begin a new named paragraph range in the TOC 20⟩ ≡
```
$toc = complete_toc_chunk($toc, $toc_range_start, $par_number_counter-1, $toc_name);
$toc_range_start = $par_number_counter; $toc_name = $line_operand_2[$i];
if ($toc eq "") { $toc = $section_sigil[$line_sec[$i]]."."; }
else { $toc .= "; "; }
$toc .= '$'.$ornament.'$'.$par_number_counter;
```

This code is used in §17.

§**21.** This one is when we hit the bar.

⟨Complete any named paragraph range now half-done in the TOC 21⟩ ≡
```
$toc = complete_toc_chunk($toc, $toc_range_start, $par_number_counter, $toc_name);
$toc_range_start = 0;
```

This code is used in §17.

§**22.** And here we run into the end of section.

⟨Complete the TOC from the section just ended 22⟩ ≡
```
$toc = complete_toc_chunk($toc, $toc_range_start, $par_number_counter-1, $toc_name);
set_toc_for_section($line_sec[$i]-1, $toc);
```

This code is used in §17.

§**23.** And here, the end of the whole web.

⟨Complete the TOC from the very last section 23⟩ ≡

```
    $toc = complete_toc_chunk($toc, $toc_range_start, $par_number_counter, $toc_name);
    set_toc_for_section($line_sec[$no_lines-1], $toc);
```

This code is used in §17.

§**24.** Note the TEX tie ~ to prevent line breaks between the paragraph number and names, and that we use a number range only when there are multiple paragraphs.

```
sub complete_toc_chunk {
    my $toc = $_[0];
    my $toc_range_start = $_[1];
    my $pnum = $_[2];
    my $toc_name = $_[3];
    if ($toc_range_start != 0) {
        if ($pnum != $toc_range_start) { $toc .= "-".$pnum; }
        $toc .= "~".$toc_name;
    }
    return $toc;
}
```

§**25.** Once we have determined the table of contents for a section, we call the following:

```
sub set_toc_for_section {
    my $sect_no = $_[0];
    my $toc = $_[1];
    $section_toc[$sect_no] = $toc;
    return;                                          delete this line for a debugging trace, viz...
    if ($toc eq "") {
        print "Warning: ", $section_sigil[$sect_no], " has no TOC.\n";
    } else {
        print $section_sigil[$sect_no], " has TOC ", $toc, "\n";
    }
}
```