Programming Languages 3/plan

Purpose

To characterise the relevant differences in behaviour between the various programming languages we support: for instance, how comments are represented.

Definitions

¶1. At any given time, there's a current programming language, and it is always one of the following possibilities:

```
define $C_LANGUAGE 1                          C or C++
define $C_FOR_INFORM_LANGUAGE 2               Ditto, but with NI extensions
define $PERL_LANGUAGE 3                       Perl
define $I6_LANGUAGE 4                         Inform 6
define $I7_LANGUAGE 5                         Inform 7 extension
define $PLAIN_LANGUAGE 6                      Plain text
define $NO_LANGUAGE 100                       Nothing to tangle, by fiat

$web_language = $C_LANGUAGE;                  The default
$tangled_extension = ".c";
```

---

§**1.** Setting the current language from a textual form:

```
sub language_set {
    my $lname = $_[0];
    $web_language = -1;
    if ($lname eq "C") { $web_language = $C_LANGUAGE; $tangled_extension = ".c"; }
    if ($lname eq "C++") { $web_language = $C_LANGUAGE; $tangled_extension = ".cpp"; }
    if ($lname eq "C for Inform") {
        $web_language = $C_FOR_INFORM_LANGUAGE; $tangled_extension = ".c";
        ⟨Flag some identifiers as unusual in this form of C 2⟩;
    }
    if ($lname eq "Perl") { $web_language = $PERL_LANGUAGE;  $tangled_extension = ".pl"; }
    if ($lname eq "Inform 6") { $web_language = $I6_LANGUAGE;  $tangled_extension = ".i6"; }
    if ($lname eq "Inform 7") { $web_language = $I7_LANGUAGE;  $tangled_extension = ".i7x"; }
    if ($lname eq "Plain Text") { $web_language = $PLAIN_LANGUAGE;  $tangled_extension = ".txt"; }
    if ($lname eq "None") { $web_language = $NO_LANGUAGE;  $tangled_extension = ""; }
    if ($web_language == -1) {
        inweb_fatal_error("unsupported programming language ".$lname);
    }
}
```

§**2.** The term "blacklisting" is a little melodramatic. A function which has been blacklisted is allowed to be defined more than once in the source code; Inform needs this for `isdigit`, of all things, because the Windows library's implementation is deficient.

Similarly, a member name which has been blacklisted is allowed to be present in more than one structure. This of course is perfectly legal in C, but the Inform source code tries to avoid doing it, since it leads to confusion and makes it harder to nail down where the structure is used. A few exemptions are made for members deliberately used in common across wide ranges of structures.

⟨Flag some identifiers as unusual in this form of C 2⟩ ≡

```
    $blacklisted_functions{"isdigit"} = 1;
    $blacklisted_members{"word_ref1"} = 1;
    $blacklisted_members{"word_ref2"} = 1;
    $blacklisted_members{"next"} = 1;
    $blacklisted_members{"down"} = 1;
    $blacklisted_members{"allocation_id"} = 1;
    $members_allowed_to_be_unused{"handling_routine"} = 1;
```

This code is used in §1.

§**3.** The file extension generally used by files of code in this language:

```
sub language_file_extension { return $tangled_extension; }
```

§**4.** Any compulsory heading that must occur on line 1, really:

```
sub language_shebang {
    my $text = $_[0];
    if ($web_language == $PERL_LANGUAGE) { return "#!/usr/bin/perl\n\n"; }
    return "";
}
```

§**5.** Now a routine to write a comment. Languages without comment should write nothing.

```
sub language_comment {
    my $text = $_[0];
    if ($web_language == $C_LANGUAGE) { return "/* ".$text." */\n"; }
    if ($web_language == $C_FOR_INFORM_LANGUAGE) { return "/* ".$text." */\n"; }
    if ($web_language == $PERL_LANGUAGE) { return "# ".$text."\n"; }
    if ($web_language == $I6_LANGUAGE) { return "! ".$text."\n"; }
    if ($web_language == $I7_LANGUAGE) { return "[".$text."]\n"; }
    return "";
}
```

§**6.** And we need to spot and so on... comments:

```
sub language_and_so_on {
    my $text = $_[0];
    if (($web_language == $C_LANGUAGE) || ($web_language == $C_FOR_INFORM_LANGUAGE)) {
        if ($text =~ m/^\s*\/\* ...and so on... \*\/\s*/) { return 1; }
    }
    if ($web_language == $I6_LANGUAGE) {
        if ($text =~ m/^\s*\!\s*...and so on...\s*$/) { return 1; }
    }
    if ($web_language == $I7_LANGUAGE) {
        if ($text =~ m/^\s*\[\s*...and so on...\s*\]\s*$/) { return 1; }
    }
    return 0;
}
```

§**7.** And in general to see when a line ends in a comment:

```
sub line_ends_with_comment {
    my $matter = $_[0];
    if (($web_language == $C_LANGUAGE) || ($web_language == $C_FOR_INFORM_LANGUAGE)) {
        if ($matter =~ m/^(.*)\/\*\s*(.*?)\s*\*\/\s*$/) {
            $part_before_comment = $1; $part_within_comment = $2; return 1;
        }
    }
    if ($web_language == $PERL_LANGUAGE) {
        if ($matter =~ m/^\#\s+(.*?)\s*$/) {
            $part_before_comment = ""; $part_within_comment = $1; return 1;
        }
        if ($matter =~ m/^(.*)\s+\#\s+(.*?)\s*$/) {
            $part_before_comment = $1; $part_within_comment = $2; return 1;
        }
    }
    return 0;
}
```

§**8.** To place page breaks strategically within code:

```
sub language_pagebreak_comment {
    my $l = $_[0];
    if (($web_language == $C_LANGUAGE) || ($web_language == $C_FOR_INFORM_LANGUAGE)) {
        if ($l =~ m/^\s*\/\* PAGEBREAK \*\/\s*$/) { return 1; }
    }
    if ($web_language == $PERL_LANGUAGE) {
        if ($l =~ m/^\s*\#\s+PAGEBREAK\s*$/) { return 1; }
    }
    if ($web_language == $I6_LANGUAGE) {
        if ($l =~ m/^\s*\!\s+PAGEBREAK\s*$/) { return 1; }
    }
    if ($web_language == $I7_LANGUAGE) {
        if ($l =~ m/^\s*\[\s*PAGEBREAK\s*\]\s*$/) { return 1; }
    }
    return 0;
}
```

§**9.** Does the currently selected language allow tangling? (Yes, unless this is purely a documentation project.)

```
sub language_tangles {
    if ($web_language == $NO_LANGUAGE) { return 0; }
    return 1;
}
```

§**10.** The following routines handle the `@d` escape, writing a definition of the constant named `$term` as the value given. If the value spans multiple lines, the first-line part is supplied to `language_start_definition` and then subsequent lines are fed in order to `language_prolong_definition`. At the end, `language_end_definition` is called.

```
sub language_start_definition {
    my $term = $_[0];
    my $startval = $_[1];
    if (($web_language == $C_LANGUAGE) || ($web_language == $C_FOR_INFORM_LANGUAGE)) {
        if ($web_language == $C_FOR_INFORM_LANGUAGE) { $startval =~ s/::/__/g; }
        print TANGLEOUT "#define ", $term, " ", $startval;
        return;
    }
    if ($web_language == $PERL_LANGUAGE) {
        print TANGLEOUT $term, " = ", $startval;
        return;
    }
    inweb_error("programming language $web_language does not support \@d");
}
sub language_prolong_definition {
    my $more = $_[0];
    if (($web_language == $C_LANGUAGE) || ($web_language == $C_FOR_INFORM_LANGUAGE)) {
        if ($web_language == $C_FOR_INFORM_LANGUAGE) { $more =~ s/::/__/g; }
        print TANGLEOUT "\\\n    ", $more;
        return;
```

```
    }
    inweb_error("programming language $web_language does not support multiline \@d");
}
sub language_end_definition {
    if (($web_language == $C_LANGUAGE) || ($web_language == $C_FOR_INFORM_LANGUAGE)) {
        print TANGLEOUT "\n";
    }
    if ($web_language == $PERL_LANGUAGE) {
        print TANGLEOUT "\n;\n";
    }
}
```