An implementation of Semi-Literate programming for Raku with Pod6

INTRODUCTION

I want to create a semi-literate Raku source file with the extension `.sl`. Then, I will *weave* it to generate a readable file in formats like Markdown, PDF, HTML, and more. Additionally, I will *tangle* it to create source code without any Pod6.

To do this, I need to divide the file into `Pod` and `Code` sections by parsing it. For this purpose, I will create a dedicated Grammar.

(See Useful::Regexes for the definitions of the named regexes used here. (`hws` == Horizontal WhiteSpace))

The Grammar

```
    3| grammar Semi::Literate is export does Useful::Regexes {
```

Our file will exclusively consist of `Pod` or `Code` sections, and nothing else. The `Code` sections are of two types, a) code that is woven into the documentation, and b) code that is not woven into the documentation. The `TOP` token clearly indicates this.

```
    4|     token TOP {
    5|         [
    6|           || <pod>
    7|           || <code>
    8|         ]*
    9|     }
   10|
   11|     token code  {
   12|         [
   13|           || <non-woven>+
   14|           || <woven>+
   15|         ]
   16|     }
```

The Pod6 delimiters

According to the documentation,

Every Pod6 document has to begin with '=begin pod' and end with '=end pod'.

So let's define those tokens.

The `begin-pod` token

```
   17|     token begin-pod {
   18|         <leading-ws>
   19|         '=' begin <hws> pod
   20|         <ws-till-EOL>
   21|     }
```

The `end-pod` token

The `end-pod` token is much simpler.

```
   22|     token end-pod  {
   23|         <leading-ws>
   24|         '=' end <hws> pod
   25|         <ws-till-EOL>
   26|     }
```

Replacing Pod6 sections with blank lines

When we *tangle* the semi-literate code, all the Pod6 will be removed. This would leave a lot of blank lines in the Raku code. So we'll clean it up. We provide the option for users to specify the number of empty lines that should replace a `pod` block. To do this, simply add a Pod6 comment immediately after the `=begin pod` statement. The comment can say anything you like, but must end with a digit specifying the number of blank lines with which to replace the Pod6 section.

```
=begin pod
=comment I want this pod block replaced by only one line 1
...
=end pod
```

Here's the relevant regex:

```
27|     token blank-line-comment {
28|         <leading-ws>
29|         '=' comment
30|         \N*?
31|         $<num-blank-lines> = (\d+)?
32|         <ws-till-EOL>
33|     }
```

The `Pod` token

Within the delimiters, all lines are considered documentation. We will refer to these lines as `plain-lines`. Additionally, it is possible to have nested `Pod` sections. This allows for a hierarchical organization of documentation, allowing for more structured and detailed explanations.

It is also permissible for the block to be empty. Therefore, we will use the 'zero-or-more' quantifier on the lines of documentation, allowing for the possibility of having no lines in the block.

```
34|     token pod {
35|         <.begin-pod>
36|         <blank-line-comment>?
37|             [<pod> | <.plain-line>]*
38|         <.end-pod>
39|     }
```

The `Code` tokens

The `Code` sections are similarly easily defined. There are two types of `Code` sections, depending on whether they will appear in the woven code.

Woven sections

These sections are trivially defined. They are just one or more `plain-lines`.

```
40|     token woven  {
41|         [
42|             || <.plain-line>
43|         ]+
44|     }
```

Non-woven sections

Sometimes there will be code you do not want woven into the documentation, such as boilerplate code like `use v6.d;`. You have two options to mark such code. By individual lines or by a delimited block of code.

```
45|     token non-woven {
46|         [
47|           || <.one-line-no-weave>
48|           || <.delimited-no-weave>
```

```
49|          ]+
50|      }
```

One line of code

Simply append `# no-weave-this-line` at the end of the line!

```
51|      regex one-line-no-weave {
52|          $<the-code>=(<leading-ws> <optional-chars>)
53|          '#' <hws> 'no-weave-this-line'
54|          <ws-till-EOL>
55|      }
```

Delimited blocks of code

Simply add comments `# begin-no-weave` and `#end-no-weave` before and after the code you want ignored in the formatted document.

```
56|      token begin-no-weave {
57|          <leading-ws>
58|          '#' <hws> 'begin-no-weave'
59|          <ws-till-EOL>
60|      }
61|
62|      token end-no-weave {
63|          <leading-ws>
64|          '#' <hws> 'end-no-weave'
65|          <ws-till-EOL>
66|      }
67|
68|      token delimited-no-weave {
69|          <.begin-no-weave>
70|              <.plain-line>*
71|          <.end-no-weave>
72|      }
```

The `plain-line` token

The `plain-line` token is, really, any line at all... ... except for one subtlety. They it can't be one of the begin/end delimiters. We can specify that with a Regex Boolean Condition Check.

```
73|      token plain-line {
74|          :my $*EXCEPTION = False;
75|          [
76|            ||  <.begin-pod>         { $*EXCEPTION = True }
77|            ||  <.end-pod>           { $*EXCEPTION = True }
78|            ||  <.begin-no-weave>    { $*EXCEPTION = True }
79|            ||  <.end-no-weave>      { $*EXCEPTION = True }
80|            ||  <.one-line-no-weave> { $*EXCEPTION = True }
81|            || [^^ <rest-of-line>]
82|          ]
83|          <?{ !$*EXCEPTION }>
84|      }
```

And that concludes the grammar for separating `Pod` from `Code`!

```
85| }
```

The Tangle subroutine

This subroutine will remove all the Pod6 code from a semi-literate file (`.sl`) and keep only the Raku code.

```
 86|
 87| multi tangle (
```

The subroutine has a single parameter, which is the input filename. The filename is required. Typically, this parameter is obtained from the command line or passed from the subroutine `MAIN`.

```
 88|     IO::Path $input-file!,
```

The subroutine will return a `Str`, which will be a working Raku program.

```
 89|         --> Str ) is export {
```

First we will get the entire Semi-Literate `.sl` file...

```
 90|     my Str $source = $input-file.slurp;
```

The interesting stuff

We parse it using the `Semi::Literate` grammar and obtain a list of submatches (that's what the `caps` method does) ...

```
 91|     my Pair @submatches = Semi::Literate.parse(clean $source).caps;
 92|
 93|     my Str $raku-code = @submatches.map( {
```

Replace Pod6 sections with blank lines

```
 94|         when .key eq 'pod' {
 95|             my $num-blank-lines =
 96|                 .value.hash<blank-line-comment><num-blank-lines>;
 97|             "\n" x ($num-blank-lines // 1);
 98|         }
```

Add all the `Code` sections.

```
 99|         when .key eq 'code' {
100|             .value;
101|         }
102|
```

... and we will join all the code sections together...

```
103|     }
104|     ).join;
```

Remove the *no-weave* delimiters

```
105|     $raku-code ~~ s:g{
106|                         | <Semi::Literate::begin-no-weave>
107|                         | <Semi::Literate::end-no-weave>
108|                     } = '';
109|
```

```
110|     $raku-code ~~ s:g{ <Semi::Literate::one-line-no-weave> }
111|                       = "$<Semi::Literate::one-line-no-weave><the-code>\n";
```

remove blank lines at the end

```
112|     $raku-code ~~ s{\n  <blank-line>* $ } = '';
```

And that's the end of the `tangle` subroutine!

```
113|     return $raku-code;
114| }
```

The Weave subroutine

The `Weave` subroutine will *weave* the `.sl` file into a readable Markdown, HTML, or other format. It is a little more complicated than `sub tangle` because it has to include the `code` sections.

```
115| sub weave (
```

The parameters of Weave

`sub weave` will have several parameters.

`$input-file`

The input filename is required. Typically, this parameter is obtained from the command line through a wrapper subroutine `MAIN`.

```
116|     Str $input-file!;
```

`$line-numbers`

It can be useful to print line numbers in the code listing. It currently defaults to True.

```
117|     Bool :l(:$line-numbers) = True;
```

`$verbose`

Use verbose only for debugging

```
118|     Bool :v(:$verbose)      = False;
```

`sub weave` returns a Str.

```
119|         --> Str ) is export {
120|
121|     my UInt $line-number = 1;
```

First we will get the entire `.sl` file...

```
122|     my Str $source = $input-file.IO.slurp;
```

Interesting stuff

...Next, we parse it using the `Semi::Literate` grammar and obtain a list of submatches (that's what the `caps` method does) ...

```
123|     my Pair @submatches = Semi::Literate.parse(clean $source).caps;
```

...And now begins the interesting part. We iterate through the submatches and insert the `code` sections into the Pod6...

This function checks if the line of code is a full line comment. If so, return False, so nothing will be printed for this line.

If it's a line of code with a comment at the end, remove the comment from the line and return True

Otherwise return True

```
124|     sub remove-comments (Seq $lines --> Seq) {
```

```
125|
126|         my token full-line-comment {
127|             $<the-code>=(<leading-ws>)
128|             '#'
129|             <rest-of-line>
130|         }
131|
132|         my regex partial-line-comment {
133|             $<the-code>=(<leading-ws> <optional-chars>)
134|             <!after <opening-quote>>
135|             '#'
136|             $<the-comment>=<-[#]>*
137|             <ws-till-EOL>
138|         }
139|
140|         my @retval = ();
141|         for $lines.List -> $line {
142|             given $line {
143|                 when /<full-line-comment>/ {;
144|
145|                 when /<partial-line-comment>/ {
146|                     @retval.push: $<partial-line-comment><the-code>;
147|                 }
148|
149|                 default
150|                     { @retval.push: $line; }
151|             }
152|         }
153|
154|         return @retval.Seq;
155|     }
156|
157|     my Str $weave = @submatches.map( {
158|         when .key eq 'pod' {
159|             .value
160|         }
161|
162|         when .key eq 'code' {
163|             { qq:to/EOCB/ if .<code><woven>; }
164|             \=begin pod
165|             \=begin code :lang<raku>
166|              {
167|                 $_<code><woven>
168|                 ==> lines()
169|                 ==> remove-comments()
170|                 ==> map(
171|                         $line-numbers
172|                             ?? {"%4s| %s\n".sprintf($line-number++, $_) }
173|                             !! {     "%s\n".sprintf(                $_) }
174|                 )
175|                 ==> chomp()
176|              }
177|             \=end code
178|             \=end pod
179|             EOCB
180|         }
181|
182|     }
```

```
183|     ).join;
```

Remove unseemly blank lines

```
184|     my Str $non-woven-blank-lines = qq:to/EOQ/;
185|         \=end code
186|         \=end pod
187|         \=begin pod
188|         \=begin code :lang<raku>
189|         EOQ
190|
191|     my Regex $full-comment-blank-lines = rx[
192|         '=begin pod'              <ws-till-EOL>
193|         '=begin code :lang<raku>' <ws-till-EOL>
194|         [<leading-ws> \d+ | '|'?  <ws-till-EOL>]*
195|         '=end code'               <ws-till-EOL>
196|         '=end pod'                <ws-till-EOL>
197|     ];
198|
199|     $weave ~~ s:g{ $non-woven-blank-lines | <$full-comment-blank-lines> } = '';
```

And that's the end of the `weave` subroutine!

```
200|     return $weave
201| }
```

Clean the source code of unneccessary blank lines

```
202| sub clean (Str $source is copy --> Str) {
```

Remove blank lines at the begining and end of the code

Very often the `code` section of the Semi-Literate file will have blank lines that you don't want to see in the tangled working code. For example:

```
   ...
   \=end pod
                                                # <== unwanted blank line
                                                # <== unwanted blank line
   sub foo () {
       { ... }
   } # end of sub foo ()
                                                # <== unwanted blank line
                                                # <== unwanted blank line
                                                # <== unwanted blank line
   \=begin pod
   ...
203|     $source ~~ s:g{    \=end (\N*) \n+}      =  "\=end$0\n";
204|     $source ~~ s:g{\n+ \=begin (<hws> pod) } = "\n\=begin$0";
```

Remove blank lines at the end of the code.

```
205|     $source ~~ s{\n  <blank-line>* $ } = '';
206|
207|     return $source;
208| }
```

NAME

`Semi::Literate` - A semi-literate way to weave and tangle Raku/Pod6 source code.

VERSION

This documentation refers to `Semi-Literate` version 0.0.1

SYNOPSIS

```
use Semi::Literate;
# Brief but working code example(s) here showing the most common usage(s)

# This section will be as far as many users bother reading
# so make it as educational and exemplary as possible.
```

DESCRIPTION

Influences

`Semi::Literate` is based on Daniel Sockwell's Pod::Literate.

Also influenced by zyedidia's <Literate|https://zyedidia.github.io/literate/> program. Especially the idea of not weaving some portions of the code.

A full description of the module and its features. May include numerous subsections (i.e. =head2, =head2, etc.)

DEPENDENCIES

```
Useful::Regexes
```

BUGS AND LIMITATIONS

There are no known bugs in this module. Patches are welcome.

AUTHOR

Shimon Bollinger (deoac.bollinger@gmail.com)

LICENSE AND COPYRIGHT