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 C<=begin pod> and end with C<=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|      token 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| sub 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`.

```
87|     Str $input-file!,
```

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

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

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

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

Clean the source

Remove unnecessary blank lines

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:

```
                                        # <== unwanted blank lines
                                        # <== unwanted blank lines
    sub foo () {
        { ... }
    } # end of sub foo ()
                                        # <== unwanted blank lines
                                        # <== unwanted blank lines
```

So we'll remove the blank lines immediately outside the beginning and end of the Pod6 sections.

```
90|     my Str $cleaned-source = $source;
91|     $cleaned-source ~~ s:g{\=end (\N*)\n+} =   "\=end$0\n";
92|     $cleaned-source ~~ s:g{\n+\=begin (<hws> pod) [<hws> \d]?} = "\n\=begin$0";
```

The interesting stuff

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

```
93|     my Pair @submatches = Semi::Literate.parse($cleaned-source).caps;
94|
95|     my Str $raku-code = @submatches.map( {
```

Replace Pod6 sections with blank lines

```
96|         when .key eq 'pod' {
97|             my $num-blank-lines =
98|                 .value.hash<blank-line-comment><num-blank-lines>;
99|             "\n" x $num-blank-lines with $num-blank-lines;
```

```
100|              }
```

Add all the `Code` sections.

```
101|          when .key eq 'code' {
102|                  my Str $code = '';
103|                  for $_<code>.keys.reverse -> $key {
104|                      $code ~= $_<code>{$key} if $_<code>{$key};
105|                  }
106|                  $code;
107|          }
108|
```

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

```
109|      }
110|      ).join;
```

Remove the *no-weave* delimiters

```
111|      $raku-code ~~ s:g{
112|                      | <Semi::Literate::begin-no-weave>
113|                      | <Semi::Literate::end-no-weave>
114|              } = '';
115|      $raku-code ~~ s:g{ <Semi::Literate::one-line-no-weave> }
116|              = $<one-line-no-weave><the-code>;
```

remove blank lines at the end

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

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

```
118|      return $raku-code;
119| }
```

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.

```
120| 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`.

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

`$line-numbers`

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

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

`sub weave` returns a Str.

```
123|          --> Str ) is export {
124|
125|      my UInt $line-number = 1;
```

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

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

Remove blank lines at the begining and end of the code

EXPLAIN THIS!

```
127|      my Str $cleaned-source = $source;
128|      $cleaned-source ~~ s:g{\=end (\N*)\n+} =    "\=end$0\n";
129|      $cleaned-source ~~ s:g{\n+\=begin (<hws> pod) [<hws> \d]?} = "\n\=begin$0";
```

Interesting stuff

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

```
130|      my Pair @submatches = Semi::Literate.parse($cleaned-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

```
131|     my token full-line-comment {
132|         $<the-code>=(<leading-ws>)
133|         '#'
134|         <rest-of-line>
135|     }
136|
137|     my regex partial-line-comment {
138|         $<the-code>=(<leading-ws> <optional-chars>)
139|         <!after <opening-quote>>
140|         '#'
141|         $<the-comment>=<-[#]>*
142|         <ws-till-EOL>
143|     }
144|
145|     sub remove-comments (Seq $lines --> Seq) {
146|
147|         my @retval = ();
148|         for $lines.List -> $line {
149|             given $line {
150|                 when /<full-line-comment>/ {;
151|
152|                 when /<partial-line-comment>/ {
153|                     @retval.push: $<partial-line-comment><the-code>;
154|                 }
155|
156|                 default
157|                     { @retval.push: $line; }
158|             }
159|         }
160|
161|
162|         return @retval.Seq;
163|     }
164|
165|     my Str $non-woven-blank-lines = qq:to/EOQ/;
166|         \=end code
167|         \=end pod
168|         \=begin pod
169|         \=begin code :lang<raku>
170|         EOQ
171|
172|     my Regex $full-comment-blank-lines = rx[
173|         '=begin pod'                <ws-till-EOL>
174|         '=begin code :lang<raku>'   <ws-till-EOL>
175|         [<leading-ws> \d+ | '|'?    <ws-till-EOL>]*
176|         '=end code'                 <ws-till-EOL>
177|         '=end pod'                  <ws-till-EOL>
178|     ];
179|
180|     my $fmt = ($line-numbers ?? "%3s| " !! '') ~ "%s\n";
181|
182|     my Str $weave = @submatches.map( {
183|         when .key eq 'pod' {
184|             .value
185|         }
186|
```

```
187|          when .key eq 'code' {
188|              { qq:to/EOCB/ if .<code><woven>; }
189|              \=begin pod
190|              \=begin code :lang<raku>
191|               {
192|                  $_<code><woven>
193|                  ==> lines()
194|                  ==> remove-comments()
195|                  ==> map(
196|                        $line-numbers
197|                          ?? {"%4s| %s\n".sprintf($line-number++, $_) }
198|                          !! {      "%s\n".sprintf(               $_) }
199|                  )
200|                  ==> chomp()
201|               }
202|              \=end code
203|              \=end pod
204|              EOCB
205|          }
206|
207|       }
208|      ).join;
```

Remove unseemly blank lines

```
209|     $weave ~~ s:g{ $non-woven-blank-lines | <$full-comment-blank-lines> } = '';
```

remove blank lines at the end

```
210| $weave ~~ s{\n  <blank-line>* $ } = '';
```

And that's the end of the weave subroutine!

```
211|     "deleteme.rakudoc".IO.spurt: $weave;
212|     return $weave
213| }
```

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