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|           || <non-woven-code>
    8|           || <woven-code>
    9|         ]*
   10|
```

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

```
   11|     token begin-pod {
   12|         <leading-ws>
   13|         '=' begin <hws> pod
   14|         <ws-till-EOL>
   15|
```

The `end-pod` token

The `end-pod` token is much simpler.

```
   16|     token end-pod  {
   17|         <leading-ws>
   18|         '=' end <hws> pod
   19|         <ws-till-EOL>
   20|
```

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:

```
    token blank-line-comment {
        <leading-ws>
        '=' comment
        \N*?
        $<num-blank-lines> = (\d+)?
        <ws-till-EOL>

```

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.

```
    token pod {
        <begin-pod>
        <blank-line-comment>?
            [<pod> | <plain-line>]*
        <end-pod>

```

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`.

```
    token woven-code  {
        [
            || <plain-line>
        ]+

```

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.

```
    token non-woven-code {
        [
          || <one-line-no-weave>
          || <delimited-no-weave>
        ]+

```

One line of code

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

```
45|     token one-line-no-weave {
46|         <leading-ws> \N*?
47|         '#' <hws> 'no-weave-this-line'
48|         <ws-till-EOL>
49|
```

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.

```
50|     token begin-no-weave {
51|         <leading-ws>
52|         '#' <hws> 'begin-no-weave'
53|         <ws-till-EOL>
54|
55|
56|     token end-no-weave {
57|         <leading-ws>
58|         '#' <hws> 'end-no-weave'
59|         <ws-till-EOL>
60|
61|
62|     token delimited-no-weave {
63|         <begin-no-weave>
64|             <plain-line>*
65|         <end-no-weave>
66|
```

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.

```
67|     token plain-line {
68|         :my $*EXCEPTION = False;
69|         [
70|           ||  <begin-pod>         { $*EXCEPTION = True }
71|           ||  <end-pod>           { $*EXCEPTION = True }
72|           ||  <begin-no-weave>    { $*EXCEPTION = True }
73|           ||  <end-no-weave>      { $*EXCEPTION = True }
74|           ||  <one-line-no-weave> { $*EXCEPTION = True }
75|           || [^^ <rest-of-line>]
76|         ]
77|         <?{ !$*EXCEPTION }>
78|
```

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

The Tangle subroutine

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

```
80| 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`.

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

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

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

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

```
83|     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.

```
84|     my Str $cleaned-source = $source;
85|     $cleaned-source ~~ s:g{\=end (\N*)\n+} =   "\=end$0\n";
86|     $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) ...

```
87|     my Pair @submatches = Semi::Literate.parse($cleaned-source).caps;
```

...and iterate through the submatches and keep only the `code` sections...

```
88|     my Str $raku-code = @submatches.map( {
89|         when .key eq 'woven-code'|'non-woven-code' {
90|             .value;
91|         }
```

Replace Pod6 sections with blank lines

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

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

```
98|
99|      ).join;
```

Remove the *no-weave* delimiters

```
100|     $raku-code ~~ s:g{ ^^ <hws> '#' <hws> 'begin-no-weave'     <rest-of-line> }
101|         = '';
102|     $raku-code ~~ s:g{ ^^ <hws> '#' <hws> 'no-weave-this-line' <rest-of-line> }
103|         = "$0\n";
104|     $raku-code ~~ s:g{ ^^ <hws> '#' <hws> 'end-no-weave'       <rest-of-line> }
105|         = '';
```

remove blank lines at the end

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

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

```
107|     return $raku-code;
108|
```

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.

```
  109| 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`.

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

`$line-numbers`

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

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

`sub weave` returns a Str.

```
  112|         --> Str ) is export {
  113|
  114|     my UInt $line-number = 1;
```

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

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

Remove blank lines at the begining and end of the code

EXPLAIN THIS!

```
  116|     my Str $cleaned-source = $source;
  117|     $cleaned-source ~~ s:g{\=end (\N*)\n+} =   "\=end$0\n";
  118|     $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) ...

```
  119|     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

```
120|     my token full-line-comment {
121|         $<the-code>=(<leading-ws>)
122|         '#'
123|         <rest-of-line>
124|
125|
126|     my regex partial-line-comment {
127|
128|
129|
130|
131|         <ws-till-EOL>
132|
133|
134|     sub remove-comments (Seq $lines --> Seq) {
135|
136|         my @retval = ();
137|         for $lines.List -> $line {
138|             given $line {
139|                 note $++,": retval has {@retval.elems} elems";
140|
141|
142|                 when /<partial-line-comment>/ {
143|
144|                     note '-1> ', @retval.join("\t\n");
145|                     note '-the-code>', $<the-code>;
146|                     @retval.push: $<the-code>;
147|                     note '-2> ', @retval.join("\t\n");
148|                     note "\n";
149|                 }
150|
151|                 default
152|                     {#`[[note ">> normal line";]] @retval.push: $line}
153|
154|
155|
156|
157|         return @retval.Seq;
158|
159|
160|     my Str $non-woven-blank-lines = qq:to/EOQ/;
161|         \=end code
162|         \=end pod
163|         \=begin pod
164|         \=begin code :lang<raku>
165|         EOQ
166|
167|     my Regex $full-comment-blank-lines = rx[
168|         '=begin pod'              <ws-till-EOL>
169|         '=begin code :lang<raku>' <ws-till-EOL>
170|         [<leading-ws> \d+ | '|'?  <ws-till-EOL>]*
171|         '=end code'               <ws-till-EOL>
172|         '=end pod'                <ws-till-EOL>
173|     ];
174|
175|     my $fmt = ($line-numbers ?? "%3s| " !! '') ~ "%s\n";
```

```
176|
177|     my Str $weave = @submatches.map( {
178|         when .key eq 'pod' {
179|             .value
180|
181|
182|         when .key eq 'woven-code' { qq:to/EOCB/; }
183|             \=begin pod
184|             \=begin code :lang<raku>
185|              {
186|                 .value
187|                 ==> lines()
188|                 ==> remove-comments()
189|                 ==> map(
190|                         $line-numbers
191|                             ?? {"%4s| %s\n".sprintf($line-number++, $_) }
192|                             !! {     "%s\n".sprintf(                $_) }
193|                 )
194|
195|              }
196|             \=end code
197|             \=end pod
198|             EOCB
199|
200|         when .key eq 'non-woven-code' {
201|
202|
203|
204|
205|     ).join;
```

Remove unseemly blank lines

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

remove blank lines at the end

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

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

```
208|     "deleteme.rakudoc".IO.spurt: $weave;
209|     return $weave
210|
```

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