

Is this quine?

```

0 use crate::{HEIGHT, HELP_TXT, WIDTH};
1 use printpdf::*;
2
3 pub fn add_new_page(
4     doc: &mut PdfDocumentReference,
5     fname: &str,
6 ) -> (PdfPageReference, PdfLayerReference) {
7     let (page, layer) = doc.add_page(Mm(WIDTH), Mm(HEIGHT), fname);
8     (doc.get_page(page), doc.get_page(page).get_layer(layer))
9 }
10
11 pub fn init_doc(title: &str, layer_name: &str) -> (PdfDocumentReference,
IndirectFontRef) {
12     let (mut doc, title_page, title_layer) =
13         PdfDocument::new(title, Mm(WIDTH), Mm(HEIGHT), layer_name);
14     doc = doc.with_conformance(PdfConformance::Custom(CustomPdfConformance {
15         requires_icc_profile: false,
16         requires_xmp_metadata: false,
17         ..Default::default()
18     }));
19     let title_layer = doc.get_page(title_page).get_layer(title_layer);
20
21     let mut font_reader =
22         std::io::Cursor::new(include_bytes!("../assets/JetBrainsMono-
Regular.ttf").as_ref());
23     let font = doc.add_external_font(&mut font_reader).unwrap();
24
25     title_layer.use_text(title, 50.0, Mm(0.0), Mm(HEIGHT / 2.0), &font);
26     doc.add_bookmark("title page", title_page);
27     (doc, font)
28 }
29 pub fn exit() -> ! {
30     std::process::exit(1)
31 }
32
33 pub fn parse_cli() -> CliOpts {
34     use std::env::args;
35     let mut cmd_args_tmp = args().collect::<Vec<String>>();
36     cmd_args_tmp.remove(0);
37     let mut inputs = Vec::new();
38     let mut output_file = None;
39     let mut abort_on_binary = false;
40     let mut it = cmd_args_tmp.iter();
41     let mut opt_t: Option<String> = None;
42     while let Some(i) = it.next() {
43         match i.as_str() {
44             "-o" => {
45                 output_file = match it.next() {

```

```

46         Some(x) => Some(x.clone()),
47         None => {
48             eprintln!("expected an output file after \"-o\"\n{} ",
HELP_TXT);
49             exit();
50         }
51     }
52 }
53 "--title" | "-t" => {
54     opt_t = match it.next() {
55         Some(x) => Some(x.clone()),
56         None => {
57             eprintln!("expected an argument after \"--title\"\n{} ",
HELP_TXT);
58             exit();
59         }
60     }
61 }
62 "--stop-on-bad-file" | "-s" => abort_on_binary = true,
63 n => {
64     if n.starts_with("-") {
65         eprintln!("unexpected option: {}\n{}", n, HELP_TXT);
66         exit();
67     }
68     inputs.push(n.to_string());
69 }
70 }
71 }
72 if inputs.len() < 1 {
73     eprintln!("");
74     eprintln!("{}", HELP_TXT);
75     exit();
76 }
77 if output_file.is_none() {
78     eprintln!("printpdf needs one output file");
79     eprintln!("");
80     eprintln!("{}", HELP_TXT);
81     exit();
82 }
83
84 let title = match opt_t {
85     Some(t) => t,
86     None => "TITLE".to_string()
87 };
88 CliOpts {
89     inputs,
90     output_file: output_file.unwrap(),
91     title,

```

```
92         abort_on_binary,  
93     }  
94 }  
95  
96 pub struct CliOpts {  
97     pub inputs: Vec<String>,  
98     pub output_file: String,  
99     pub title: String,  
100     pub abort_on_binary: bool,  
101 }
```

```

0 use printpdf::*;
1 use std::fs::File;
2 use std::io::BufWriter;
3 use util::*;
4 use walkdir::WalkDir;
5 mod util;
6
7 const WIDTH: f64 = 200.0;
8 const HEIGHT: f64 = 264.0;
9 const MAX_HEIGHT_TEXT: usize = 48;
10
11 const HELP_TXT: &'static str = "pdfcr version 1.0
12 usage:
13 pdfcr [files]... [directories]... [--stop-on-bad-file | -s] [--title | -t TITLE] -
output-file.pdf
14
15 file: an optional list of files to render
16 directories: an optional list of directories to render
17 NOTE: at least one file or directory must be provided
18
19 --stop-on-bad-file | -s: if pdfcr finds a file such as a binary file, it will not
skip it (default), but stop and not render an output file
20
21 --title | -t: specify the title of the document, default is TITLE
22
23 -o: the output pdf file to render to, required
24
25 examples:
26
27 pdfcr src -o code.pdf # classic example
28 pdfcr src Cargo.toml -o code.pdf -t \"is this a quine?\" # this renders the src
directory and a Cargo.toml file to code.pdf, with a title of \"is this a quine?\"
29 pdfcr cmd -o test.pdf --stop-on-bad-file # renders every file in cmd to test.pdf,
if it encounters binary files, it aborts the rendering
30 ";
31
32 fn main() {
33     let opts = parse_cli();
34     let (mut doc, font) = init_doc(opts.title.as_str(), opts.title.as_str());
35
36     for input in opts.inputs {
37         for e in WalkDir::new(input) {
38             match e {
39                 Ok(x) => {
40                     if x.path().is_dir() {
41                         continue;
42                     }
43                     let path = x.path().to_str().unwrap();

```

```

44         let c = match CodeFile::from_file(path, font.clone()) {
45             Ok(z) => z,
46             Err(e) => {
47                 if !opts.abort_on_binary {
48                     eprintln!(
49                         "Could not render file '{}' because {}, skipping
it",
50                         path, e
51                     );
52                     continue;
53                 } else {
54                     eprintln!(
55                         "Could not render file '{}' because {},
aborting",
56                         path, e
57                     );
58                     exit();
59                 }
60             }
61         };
62         c.print_page(&mut doc);
63         println!("Rendered: {}", path);
64         drop(c);
65     }
66     Err(e) => {
67         eprintln!("Could not render: {}", e);
68         exit();
69     }
70 }
71 }
72 }
73
74 println!("saving document...");
75 match doc.save(&mut BufWriter::new(match File::create(&opts.output_file) {
76     Ok(x) => x,
77     Err(e) => {
78         eprintln!("could not write file: {}", e);
79         exit();
80     }
81 }))) {
82     Ok(_) => {
83         println!("Saved into: {}", opts.output_file);
84     }
85     Err(e) => {
86         eprintln!("could not save doc: {}", e);
87         exit();
88     }
89 }

```

```

90 }
91
92 struct CodeFile {
93     text: String,
94     name: String,
95     font: IndirectFontRef,
96 }
97
98 impl CodeFile {
99     pub fn print_page(&self, doc: &mut PdfDocumentReference) {
100         let font_size = 11;
101         let spacing = font_size as f64 / 2.1;
102
103         let (page, mut layer) = add_new_page(doc, &self.name);
104         doc.add_bookmark(self.name.clone(), page.page);
105
106         let mut i = 0;
107         let mut line_num_ctr = 0;
108
109         for line in self.text.lines() {
110             if i >= MAX_HEIGHT_TEXT {
111                 layer = add_new_page(doc, &self.name).1;
112                 i = 0;
113             }
114             let mut b = true;
115             for bruh in textwrap::wrap(line, 85).iter() {
116                 i += 1;
117                 let mut _line: String;
118                 if b {
119                     _line = line_num_ctr.to_string();
120                     _line.push(' ');
121                     b = false;
122                 } else {
123                     _line = String::new();
124                 }
125                 _line.push_str(bruh);
126                 layer.use_text(
127                     _line,
128                     font_size as f64,
129                     Mm(2.0),
130                     Mm(264.0 - spacing * i as f64 - spacing),
131                     &self.font,
132                 );
133             }
134             line_num_ctr += 1;
135         }
136     }
137     fn from_file(fname: &str, font: IndirectFontRef) -> Result<Self, Error> {

```

```
138         let text = std::fs::read_to_string(fname)?;
139         Ok(Self {
140             text,
141             name: fname.to_string(),
142             font,
143         })
144     }
145 }
```



```
0 [package]
1 name = "pdfcr"
2 version = "1.0.0"
3 authors = ["g-w1 <jacoblevgw@gmail.com>"]
4 license-file = "LICENSE"
5 homepage = "https://github.com/g-w1/pdfcr"
6 repository = "https://github.com/g-w1/pdfcr"
7 description = "A tool to render a codebase to a pdf."
8 readme = "README.md"
9 edition = "2018"
10
11 # See more keys and their definitions at https://doc.rust-
12 lang.org/cargo/reference/manifest.html
13 [dependencies]
14 printpdf = "0.3.4"
15 textwrap = "0.13.0"
16 walkdir = "2"
```

0 /target  
1 code.pdf

0 MIT License  
1  
2 Copyright (c) 2020 g-w1  
3  
4 Permission is hereby granted, free of charge, to any person obtaining a copy  
5 of this software and associated documentation files (the "Software"), to deal  
6 in the Software without restriction, including without limitation the rights  
7 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell  
8 copies of the Software, and to permit persons to whom the Software is  
9 furnished to do so, subject to the following conditions:  
10  
11 The above copyright notice and this permission notice shall be included in all  
12 copies or substantial portions of the Software.  
13  
14 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR  
15 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,  
16 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE  
17 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER  
18 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,  
19 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE  
20 SOFTWARE.

```
0 # pdfcr
1
2 # PDF CODE RENDERER:
3
4 This is a utility to take a files and turn it into a pdf optimised for reading on a
kindle (or anywhere) with bookmarks.
5
6 ```
7 pdfcr version 1.0
8 usage:
9 pdfcr [files]... [directories]... [--stop-on-bad-file | -s] [--title | -t TITLE] -o
output-file.pdf
10
11 file: an optional list of files to render
12 directories: an optional list of directories to render
13 NOTE: at least one file or directory must be provided
14
15 --stop-on-bad-file | -s: if pdfcr finds a file such as a binary file, it will not
skip it (default), but stop and not render an output file
16
17 --title | -t: specify the title of the document, default is TITLE
18
19 -o: the output pdf file to render to, required
20
21 examples:
22
23 pdfcr src -o code.pdf # classic example
24 pdfcr src Cargo.toml -o code.pdf -t "is this a quine?" # this renders the src
directory and a Cargo.toml file to code.pdf, with a title of "is this a quine?"
25 pdfcr cmd -o test.pdf --stop-on-bad-file # renders every file in cmd to test.pdf,
if it encounters binary files, it aborts the rendering
26 ```
27
28 An example rendered file is [example_out.pdf](./example_out.pdf) from this codebase
29
30 This has much higher speeds, and a lower memory footprint than the main competitor
[render50](https://github.com/cs50/render50). The reason that I made this was that I
wanted to view a very large codebase on a kindle, and render50 used over 4gb of ram
to render it, which was unacceptable.
31
32 ```bash
33 time render50 src -o out.pdf
34 Rendered src/main.rs.
35 Rendered src/util.rs.
36 Rendered out.pdf.
37
38 real0m27.082s
39 user0m8.432s
```

```
40 sys0m0.743s
41 time pdfcr src -o out.1.pdf
42 Rendered: src/util.rs
43 Rendered: src/main.rs
44 saving document...
45 Saved into: out.1.pdf
46
47 real0m0.125s
48 user0m0.113s
49 sys0m0.012s
50 ````
```