Is this quine?

```
use crate::{HEIGHT, HELP_TXT, WIDTH};
use printpdf::*;

pub fn add_new_page(
    doc: &mut PdfDocumentReference,
    fname: &str,
) -> (PdfPageReference, PdfLayerReference) {
    let (page, layer) = doc.add_page(Mm(WIDTH), Mm(HEIGHT), fname);
    (doc.get_page(page), doc.get_page(page).get_layer(layer))
}

pub fn init_doc(title: &str, layer_name: &str) -> (PdfDocumentReference,
IndirectFontRef) {
    let (mut doc, title_page, title_layer) =
        PdfDocument::new(title, Mm(WIDTH), Mm(HEIGHT), layer_name);
    doc = doc.with_conformance(PdfConformance::Custom(CustomPdfConformance {
        requires_icc_profile: false,
        requires_xmp_metadata: false,
        ..Default::default()
    }));
    let title_layer = doc.get_page(title_page).get_layer(title_layer);

    let mut font_reader =
        std::io::Cursor::new(include_bytes!("../assets/JetBrainsMono-
Regular.ttf").as_ref());
    let font = doc.add_external_font(&mut font_reader).unwrap();

    title_layer.use_text(title, 50.0, Mm(0.0), Mm(HEIGHT / 2.0), &font);
    doc.add_bookmark("title page", title_page);
    (doc, font)
}
pub fn exit() -> ! {
    std::process::exit(1)
}

pub fn parse_cli() -> CliOpts {
    use std::env::args;
    let mut cmd_args_tmp = args().collect::<Vec<String>>();
    cmd_args_tmp.remove(0);
    let mut inputs = Vec::new();
    let mut output_file = None;
    let mut abort_on_binary = false;
    let mut it = cmd_args_tmp.iter();
    let mut opt_t: Option<String> = None;
    while let Some(i) = it.next() {
        match i.as_str() {
            "-o" => {
                output_file = match it.next() {
```

```rust
                    Some(x) => Some(x.clone()),
                    None => {
                        eprintln!("expected an output file after \"-o\"\n{} ", HELP_TXT);
                        exit();
                    }
                }
            }
            "--title" | "-t" => {
                opt_t = match it.next() {
                    Some(x) => Some(x.clone()),
                    None => {
                        eprintln!("expected an argument after \"--title\"\n{} ", HELP_TXT);
                        exit();
                    }
                }
            }
            "--stop-on-bad-file" | "-s" => abort_on_binary = true,
            n => {
                if n.starts_with("-") {
                    eprintln!("unexpected option: {}\n{}", n, HELP_TXT);
                    exit();
                }
                inputs.push(n.to_string());
            }
        }
    }
    if inputs.len() < 1 {
        eprintln!("");
        eprintln!("{}", HELP_TXT);
        exit();
    }
    if output_file.is_none() {
        eprintln!("printpdf needs one output file");
        eprintln!("");
        eprintln!("{}", HELP_TXT);
        exit();
    }

    let title = match opt_t {
        Some(t) => t,
        None => "TITLE".to_string()
    };
    CliOpts {
        inputs,
        output_file: output_file.unwrap(),
        title,
        abort_on_binary,
    }
}

pub struct CliOpts {
    pub inputs: Vec<String>,
    pub output_file: String,
    pub title: String,
    pub abort_on_binary: bool,
}
```

```rust
use printpdf::*;
use std::fs::File;
use std::io::BufWriter;
use util::*;
use walkdir::WalkDir;
mod util;

const WIDTH: f64 = 200.0;
const HEIGHT: f64 = 264.0;
const MAX_HEIGHT_TEXT: usize = 48;

const HELP_TXT: &'static str = "pdfcr version 1.0
usage:
pdfcr [files]... [directories]... [--stop-on-bad-file | -s] [--title | -t TITLE] -
output-file.pdf

file: an optional list of files to render
directories: an optional list of directories to render
NOTE: at least one file or directory must be provided

--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

--title | -t: specify the title of the document, default is TITLE

-o: the output pdf file to render to, required

examples:

pdfcr src -o code.pdf # classic example
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?\"
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
";

fn main() {
    let opts = parse_cli();
    let (mut doc, font) = init_doc(opts.title.as_str(), opts.title.as_str());

    for input in opts.inputs {
        for e in WalkDir::new(input) {
            match e {
                Ok(x) => {
                    if x.path().is_dir() {
                        continue;
                    }
                    let path = x.path().to_str().unwrap();
                    let c = match CodeFile::from_file(path, font.clone()) {
                        Ok(z) => z,
                        Err(e) => {
                            if !opts.abort_on_binary {
                                eprintln!(
                                    "Could not render file '{}' because {}, skippi
it",
                                    path, e
                                );
                                continue;
                            } else {
                                eprintln!(
                                    "Could not render file '{}' because {},
aborting",
                                    path, e
                                );
                                exit();
                            }
                        }
                    };
                    c.print_page(&mut doc);
                    println!("Rendered: {}", path);
                    drop(c);
                }
                Err(e) => {
                    eprintln!("Could not render: {}", e);
                    exit();
                }
            }
        }
    }

    println!("saving document...");
    match doc.save(&mut BufWriter::new(match File::create(&opts.output_file) {
        Ok(x) => x,
        Err(e) => {
            eprintln!("could not write file: {}", e);
            exit();
        }
    })) {
        Ok(_) => {
            println!("Saved into: {}", opts.output_file);
        }
        Err(e) => {
            eprintln!("could not save doc: {}", e);
            exit();
        }
    }
```

```rust
}

struct CodeFile {
    text: String,
    name: String,
    font: IndirectFontRef,
}

impl CodeFile {
    pub fn print_page(&self, doc: &mut PdfDocumentReference) {
         let font_size = 11;
         let spacing = font_size as f64 / 2.1;

         let (page, mut layer) = add_new_page(doc, &self.name);
         doc.add_bookmark(self.name.clone(), page.page);

         let mut i = 0;
         let mut line_num_ctr = 0;

         for line in self.text.lines() {
             if i >= MAX_HEIGHT_TEXT {
                 layer = add_new_page(doc, &self.name).1;
                 i = 0;
             }
             let mut b = true;
             for bruh in textwrap::wrap(line, 85).iter() {
                 i += 1;
                 let mut _line: String;
                 if b {
                     _line = line_num_ctr.to_string();
                     _line.push(' ');
                     b = false;
                 } else {
                     _line = String::new();
                 }
                 _line.push_str(bruh);
                 layer.use_text(
                     _line,
                     font_size as f64,
                     Mm(2.0),
                     Mm(264.0 - spacing * i as f64 - spacing),
                     &self.font,
                 );
             }
             line_num_ctr += 1;
         }
     }
     fn from_file(fname: &str, font: IndirectFontRef) -> Result<Self, Error> {
        let text = std::fs::read_to_string(fname)?;
        Ok(Self {
            text,
            name: fname.to_string(),
            font,
        })
    }
}
```

```toml
[package]
name = "pdfcr"
version = "1.0.0"
authors = ["g-w1 <jacoblevgw@gmail.com>"]
license-file = "LICENSE"
homepage = "https://github.com/g-w1/pdfcr"
repository = "https://github.com/g-w1/pdfcr"
description = "A tool to render a codebase to a pdf."
readme = "README.md"
edition = "2018"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
printpdf = "0.3.4"
textwrap = "0.13.0"
walkdir = "2"
```

```
0 /target
1 code.pdf
```

~~~markdown
# pdfcr

# PDF CODE RENDERER:

This is a utility to take a files and turn it into a pdf optimised for reading on a kindle (or anywhere) with bookmarks.

```
pdfcr version 1.0
usage:
pdfcr [files]... [directories]... [--stop-on-bad-file | -s] [--title | -t TITLE] -o output-file.pdf

file: an optional list of files to render
directories: an optional list of directories to render
NOTE: at least one file or directory must be provided

--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

--title | -t: specify the title of the document, default is TITLE

-o: the output pdf file to render to, required

examples:

pdfcr src -o code.pdf # classic example
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?\"
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
```

An example rendered file is [example_out.pdf](./example_out.pdf) from this codebas

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.

```bash
 time render50 src -o out.pdf
Rendered src/main.rs.
Rendered src/util.rs.
Rendered out.pdf.

real0m27.082s
user0m8.432s
~~~

~~~
sys0m0.743s
 time pdfcr src -o out.1.pdf
Rendered: src/util.rs
Rendered: src/main.rs
saving document...
Saved into: out.1.pdf

real0m0.125s
user0m0.113s
sys0m0.012s
```
~~~