Rewriting TEX Today

Tyge Tiessen

TUG 2024

Overview

TLDR[1]:

I wrote an implementation of TEX82 in Rust.

- Why I started it
- How I proceeded
- Difficulties I encountered
- Current state and outlook

[1] Too long; didn't read

About me

- Work as cryptographer in academia
- Not an expert in TEX
- Not an expert in Rust

Why I started this project

- Was looking for a medium-sized programming side-project
- Interest in gaining familiarity with Rust
- Considered writing a parser of sorts

Why I started this project

- Recently back to academia, working again with LaTeX
- Decided to write a parser for LaTeX
- Stumbled upon `tex.pdf`[2]
- Decided to attempt a rewrite of TeX in Rust

[2] Run `texdoc tex`

Motivating questions

- How well does the original code map to a modern programming language?
- Can it be rewritten as idiomatic code?
- Can the code become easier to understand?
- Can the performance be improved?

About TeX82

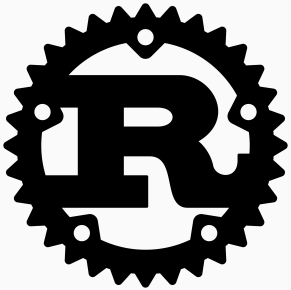
- Base of today's TeX engines
- Written in WEB which compiles to Pascal
- Great care to use minimal memory
- Ensures portability:
  - No external dependencies
  - Minimal assumptions on operating system
  - Uses only common subset of Pascal-dialects

About TEX82

- All memory is statically allocated
- Lots of global variables
- Constant strings are handled via a pool file
- Dynamic strings via a single buffer
- Lots of goto statements

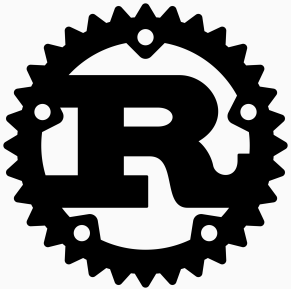
About Rust

- Fast
- Type-safe
- Memory-safe



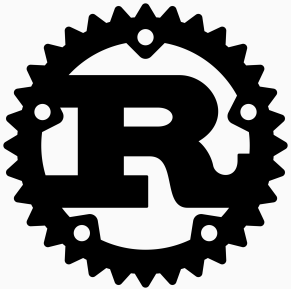
Translating WEB to Rust

- ✔ Imperative
- ✔ Has macros
- ✔ Good control over memory usage/layout



Translating WEB to Rust

- ✘ No mutable global state
- ✘ No goto statement
- ✘ No transmuting of memory
- ✔ escape hatch of **`unsafe`**



“Ship of Theseus” approach

- Write initial code as literal translation of WEB code
- Ensure correctness of code using TRIP test and others
- Make the code more idiomatic piece by piece

Changing implementation details

- Use dynamic memory
- Use built-in data structures such as hash tables
- Use richer types
- Use static strings (no string pool)

Challenges

- Manual translation inevitably created bugs
- TRIP test is *not* easy to follow
- Lots of interdependencies
- At times difficult to understand all nuances of the code

Example—Pascal (TEX: The Program, section 123)

```
procedure flush_list(p: pointer);
  var q, r: pointer;
  begin if p <> null then
    begin r := p;
    repeat q := r; r := link(r);
      stat decr(dyn_used); tats
    until r = null;
    link(q) := avail; avail := p;
    end;
  end;
```

Example—Rust (TEX: The Program, section 123)

```
unsafe fn flush_list(p: usize) {
    let mut r, q;
    if p != NULL {
        r = p;
        loop {
            q = r; r = link!(r) as usize;
            if cfg!(feature = "stats") { DYN_USED -= 1; }
            if r == NULL { break; }
        }
        link!(q) = AVAIL as Halfword;
        AVAIL = p;
    }
}
```

Interdependencies in TEX—An example

Reading an input token

- Requires knowing current category codes
- Might cause an error, depending on where the token is read.
- Might print to log file and/or standard out
- Might read an internal variable
- Might change alignment state (e.g. when constructing a table)

Logging an error

- Needs to know current definitions of control sequences (to print context)
- Might change current input stack through insertions and deletions

Three large entities

Logger

Responsible for interaction with the user and printing to log file and standard out.

Eqtb (“Table of equivalents”)

Stores current (and shadowed) values of current variables.

Scanner

Responsible for getting the next token from the input stack.

A typical function signature

```
/// See 1120.
fn ensure_list_is_empty_in_math_mode(
    cur_list: &mut Vec<Node>,
    scanner: &mut Scanner,
    eqtb: &mut Eqtb,
    logger: &mut Logger,
) {
```

Current state

- Implementation passes TRIP test[3]
- Produces identical output[4]
- No global variables, no **`unsafe`** code
- Most built-in limits removed, e.g.[5]
  - Number of control sequences
  - Number of registers
  - Number of strings
- No external dependencies

[3] Apart from one line being printed a little earlier then in TEX82
[4] When adapting time stamp and version strings
[5] Some are left in to catch programming errors though, such as the number of semantic levels

But…

- Slower by a factor of about 2.
- No integration with Kpathsea, only hard-coded directories.
- No working time stamps.
- Only implemented for Linux.
- No PDF output, no ε-TEX extentions.
- Little documentation, mostly references to TEX: The Program.

You can check it out here:

`https://github.com/tyti/rtex`

Thanks