pttypter

Programátorská příručka

verze `1.0.0`

Marek Smolík (62880168)

2. prosince 2022

1 Přehled codebase

Celá kódová báze programu je v jazyku Rust, konkrétně je využit program Cargo, který slouží ke správě Rustového projektu. Rust je vysoce výkonný nízkoúrovňový jazyk pro systémové programování, čiže je schopen zvládnout i rozsáhlé vstupy.

2 Rozvržení souborů

Rozvržení souborové struktury je striktně podřízeno řežii Cargo. Kořenový adresář obsahuje

- složku `src` obsahující samotný zdrojový rustový kód (jehož vstupním bodem je `src/main.rs`),
- soubor `Cargo.toml` obsahující metadata o projektu a případné dependencies,
- soubor `Cargo.lock` obsahující podrobné informace o specifických verzích dependencies (analog souboru `package.lock`, pro JavaScriptové developery).

3 Externí knihovny

Rustový ekosystém je založen na používání komunitních balíčku (tzv. *crates*). Z nich jsou `pttypter`em jsou využívány

- `pico-args` verze `0.5.0` pro parsování vstupních argumentů z příkazového řádku,
- `ansi_term` pro jednodušší použití ANSI kódů v terminálu.

4 Průběh exekuce

Jak již bylo řečeno, vstupním bodem programu je `src/main.rs`. Ten má dále speciální funkci `main`, která se automaticky spustí po spuštění programu. Hned na začátku proběhne nejprve kontrola vstupních argumentů ze kterých je získán barevný režim a výstupní zařízení (respektive jsou načtena výchozí nastavení). Pokud však argumenty neobsahují vstupní kód, je zahájen proces načítání vstupu přímo ze standartního vstupu (`STDIN`). To umožňuje v unixových terminálech využít rour k posílání vstupního kódu `pttypteru` přímo. Pokud však není možné získat vstup, je proces ukončen (s návratovým kódem `1`).

Nuže pokud je vstup získán, je následovně na základě volby způsobu výstupu zvolena příslušná „tiskárna“. Pro HTML+CSS výstup se nachází v `src/printer/html.rs` a pro terminál v `src/printer/term.rs`. Vybrané tiskárně je následně předán zvolený barevný režim a vstupní JSON. Ten však nejprve projde lexerem.

Lexer je umísten v `src/lex.rs`. Slouží k převádění kódu na sekvenci tokenů, o nichž lze jednoznačně říci, jak je reprezentovat. Jeho jádrem je enumerace těchto tokenů

```
1  pub enum LexItem {
2      LBrace,
3      RBrace,
4      LBracket,
5      RBracket,
6      Quote,
7      Colon,
8      BraceComma,
9      BracketComma,
10     Str(String),
11     Num(String),
12     Null,
13     True,
14     False,
15 }
```

Zde už se nachází první z konsekvencí faktu, že `pttypter` **neprovádí žádnou validitu kódu**. Pouze se snaží co nejlépe kód ztokenizovat. Zde konkrétně je to vidno na 11. řádku, kde je JSONové číslo reprezentované pouhým řetězcem znaků, nikoliv „skutečným“ číslem.

Samotné lexování je prováděno tak, že je iterováno postupně přes každý znak vstupu. Přitom je zachována informace o současném „prostředí“. Tím je reprezentováno, kde se momentálně exekuce nachází vzhledem k syntaktické struktuře JSONu a dle toho je s nově načtenými znaky různě zacházeno. Například se znakem `:` bude jinak zacházeno

- je-li momentálně zadáván název pole, nebo
- bylo-li právě zakončeno zadávání názvu pole uvozovkou a je nyní `:` očekávána pro přechod ku hodnotě pole.

Když byl celý vstup zpracován, lexer vrátí vektor (list) těchto položek `LexItem`. Následně přebírá otěže příslušná tiskárna. Každá z nich funguje samozřejmě trochu jinak, avšak obě si načtou barvy příslušného motivu pevně zakódované v souboru `src/printer/mod.rs`. Samotné barvy jsou reprezentované formátem RGB, tedy uspořádanou trojicí 8-bitových čísel reprezentující červenou, zelenou a modrou složku barvy. V této situaci je tento formát výhodný, jelikož

1. knihovna `ansi_term` je schopná RGB barvy převádět do ANSI kódu,
2. CSS podporuje RGB barvy.

Tedy není nutné tyto hodnoty nějak transformovat. Oba printery na konci procedury vrátí řetězec znaků, který je následovně vypsán na standartní výstup (`STDOUT`) a programem je vrácen výstupový kód `0`. Snad už jen dodat, že výstup je indentován 2 mezerami a jednotlivé položky listů neodřádkovávají za jejich mezerou. Ostatní možné volby pro formátovní jsou ty používané u souborů JSON zcela běžně.

5 Kompilace a cross-platform

Žádné z vývojových prostředí nebyly až na použití ANSI kódů pro terminál platformově specifické. ANSI kódy však může zkrátka zaručit využití „dobrého“ terminálu na kterékoli z běžných desktopových platforem. To příhodně nahrává schopnosti Rust cross-compilovat. Jen stačí stáhnout „build toolkit“ pro příslušnou platformu, například pomocí

```
$ rustup target add x86_64-pc-windows-msvc
```

Poté lze zkompilovat pro tuto platformu pomocí

```
cargo run --target x86_64-pc-windows-msvc
```

V nově vytvořené složce `target`