Gmail **Lukáš Hozda <luk.hozda@gmail.com>**

Programování - Třináctá a čtrnáctá hodina

2 messages

Lukáš Hozda <luk.hozda@gmail.com> Sun, Feb 10, 2019 at 7:07 PM

Zdravím všechny,
posílám to něco málo, co jsem ukázal minulý týden a pak látku z poslední hodiny.

Chtěl bych vás poprosit, jestli máte nějaké kamarády, kteří by o seminář měli zájem, abyste je pozvali, buď na zítřejší hodinu, nebo na tu po prázdninách. Je totiž nové pololetí a zájemci mají znovu příležitost se přidat na předmět a samozřejmě, čím víc lidí, tím víc srandy! :^)

Řešička kvadratických rovnic

V podstatě jediné, co jsem tak nějak ukázal předminulou hodinu byla implementace řešičky kvadratických rovnic. Tady je finální verze tohoto malého programu:

```
extern crate rstd;

use rstd::prelude::*;

fn main() {
    // předpis: ax^2 + bx + c = 0
    // d == diskriminant
    let a: f32 = prompt("zadejte koeficient a");
    let b: f32 = prompt("zadejte koeficient b");
    let c: f32 = prompt("zadejte absolutni clen");
    let d: f32 = (b * b) - 4.0 * a * c;

    if a == 0.0 && b != 0.0 {
        println!("rovnice je lineární: x = {}", -c / b);
    } else if b == 0.0 {
        match c {
            c if c == 0.0 => println!("řešení lineární rovnice je R"),
            c => println!("lineární rovnice nemá řešení"),
        }
    } else {
        match d {
            d if d == 0.0 => println!("rovnice má jeden kořen: {}", -b / (2.0 * a)),
            d if d > 0.0 => println!(
                "x1: {} x2: {}",
                (-b + d.sq_rt()) / (2.0 * a), // kořen x1
                (-b - d.sq_rt()) / (2.0 * a)  // kořen x2
            ),
            _ => println!("D je menší než nula, rovnice nemá žádné reálné řešení"),
        }
    }
}
```

Všimněte si, že je potřeba ošetřit všechny možnosti, kde bychom teoreticky mohli dělit nulou. Stejně jako normálně v základní matematice platí, že dělit nulou je "velký špatný". V programování je obecně velmi důležité mít správnou nejen "šťastnou cestu", kde je všechen vstup v pořádku podle očekávání, ale také ošetřit a zpracovat chybný vstup, aby se nestal problém, nebo nešlo nějakým způsobem program přinutit dělat něco, co nemá, viz. explots a nedefinované chování (undefined behavior).

Zdroje

Dále jsem vám ukázal také nějaké zdroje, které můžete využít při práci, když si chcete něco najít nebo potřebujete pomoc.

Zdroje informací a knihoven

- **awesome_rust**
  - seznam všeho, co je v Rustu dobré (který já spravuji, mimo jiné). V současnosti má naše sbírka už okolo tisíce záznamů, jsou tam knihovny, odkazy na přednášky a učební materiály, programy napsané v Rustu a nástroje pro vývoj
  - odkaz: https://github.com/rust-unofficial/awesome-rust
- **crates.io**
  - Hlavní uložiště knihoven a programů pro balíčkovač Cargo.
  - Nalezne se zde i **rstd**, což je ta malá knihovnička, co jsem pro předmět napsal: https://crates.io/crates/rstd
  - odkaz: https://crates.io/
- **docs.rs**
  - stránka přidružená k **crates.io**. Pro každý balíček na docs.rs je automaticky vygenerována dokumentace s textem z dokumentačních komentářů a hostována zde
  - Například pro balíček **regex**, který zprostředkovává regulérní výrazy, o kterých se budeme bavit toto pololetí to vypadá takto
  - **crates.io**: https://crates.io/crates/regex
  - **docs.io**: https://docs.rs/regex/1.1.0/regex/
- **r/rust**
  - subreddit rustu, je zde možné získat pomoc, dozvědět se o novinkách, diskutovat a seznámit se s novými knihovnami
- **Dokumentace Rustu**
  - obsahuje online knihu **The Rust Programming Language**, dokumentaci standardní knihovny a pár dalších užitečných dokumentů
  - odkaz: https://doc.rust-lang.org/
- **Stack Overflow**
  - QA stránka, občas je zde možné najít užitečné informace, ale pozor, aby u odpovědi byly poslední úpravy po roce 2015. Ten rok totiž vyšel Rust 1.0 a verze před 1.0 nejsou zpětně kompatibilní, takže je možné najít starý chybný kód
  - odkaz: https://stackoverflow.com/questions/tagged/rust
- **Github**
  - Služba na hostování gitových repozitářů. Je zde možné najít užitečné věci a často zdrojový kód balíčků, které byly nahrány na crates.io
  - Můj profil: https://github.com/luciusmagn

Kde shánět pomoc

- **Discord**
  - Existují dva servery, já jsem primárně na **Rust Programming Language Community Server** (https://areweweweyet.com/)
  - Problémy můžete psát do #beginners nebo do jiných patřičných kanálů
- **Fóra Rustu**
  - https://users.rust-lang.org
- **Mozila IRC**
  - **irc.mozilla.org**
  - Kanály: #rust, #rust-beginners
- **V neposlední řadě také můžete napsat mě :^)**

Editory

- **Pro programování se hodí mít dobrý editor, který zvýrazní kód a pomůže s editací**
- **https://areweideyet.com/**
- **VSCode** - asi nejjednoduší možnost
- **Sublime Text 3** - lehčí a rychlejší, ale může chtít trochu nastavování než bude plně vyhovovat
- **Atom** - další varianta
- **Vim** - pro hardcore borce

Nástroje

- Tyto se dají nainstalovat pomocí příkazu **cargo install nazev**
- **loc** - počítá řádky, dělá statistiku
- **rg (ripgrep)** - vyhledává v souborech
- **clippy** - linter, hledá mouchy na kódu a doporučuje, jak by se některé věci daly udělat lépe
- **racer** - našeptávač, zpravidla ho využívají editory a ne uživatel sám
- **rustfmt** - automatický formátovač kódu, instalace pomocí **rustup component add rustfmt**
- **cargo-fix** - automaticky opraví jednoduché chyby/upozornění
- **bat** - vypisuje text do příkazové řádky, ale kód je obarvený a má očíslované řádky
- **cargo-outdated** - ukazuje, které balíčky, na kterých projekt závisí, jsou zastaralé

Hashovací funkce

Tyto funkce vytvářejí ze vstupu takzvaný Hash. Hash je číslo o určité délce (zpravidla 16, 32, 64, 128 nebo 256 bitů), které by mělo být přibližně unikátní a ze kterého by nemělo být lehké získat informace o původním vstupu. Je to tedy jednosměrná šifra.

Hashe se používají například k ověření validity dat (když si stáhnete soubor a jeho hash, tak pokud si vypočítáte hash staženého souboru a není stejný, tak víte, že je soubor porušený), ukládají se místo hesla a používají se pro srovnávání dat, protože je jednoduší porovnávat hashe než bajt po bajtu.

Pokud se stane, že dva vzorky dat mají stejný hash, tak se jedná o konflikt. Bezpečné hashovací algoritmy se snaží o to, aby bylo těžké vypočítat nějaký konflikt, a aby hash by byl odstatečně dlouhý a i malé změny se silně promítly v jeho hodnotě.

Příkladem bezpečnějších, poměrně často používaných hashovacích funkcí je například **MD5**, **SHA-1**, **SHA-256** a **SHA-512**.
Jednodušší hashovací funkce, které se hodí třeba na rychlé zjišťování, jestli jsou dva soubory identické jsou například **Adler32** a jemu podobné **Fletcher16-32, djb2, sdbm, loselose.**

Tady je pár příkladů náhodných SHA-1 hashů, jen pro představu:

```
99BCAEB35D66D89F2260D70DDCEF2ADD64C60586
FC58A2C7FD6A52BD284D7D7A43C951C248CFF227
5776AF144E80BED2858755FAA07A37C708A69A8E
7A35D062A51A1D479685F1CD7D7B9A89789DAB06
D1F34E21AC075D7EFB1DB2AEB3F0AC8400F10D8B
```

Všimněte si, že se tyto čísla zapisují v hexadecimální soustavě, zmiňuji to ještě dále v tomto emailu.

Adler32

Jako příklad si ukážeme tuto hashovací funkci.

Její algoritmus je poměrně jednoduchý:

1. Mějme proměnné **a = 1, b = 0**
2. Pro každý bajt **c** ze vstupu:
   a = (a + c) % 65521;
   b = (a + b) % 65521;
3. Výsledkem je tento výraz:
   (b << 16) | a

Implementace této funkce by vypadalo zhruba takto:

```
fn adler32(src: &[u8]) -> u32 {
        let mut a: u32 = 1;
        let mut b: u32 = 0;
```

```
        for c in src {
                a = (a + *c as u32) % 65521;
                b = (a + b) % 65521;
        }

        (b << 16) | a
} l
l
```

Jako typ vstupu je zde &[u8], abychom mohli využít užitečnou syntaxi **b"textový řetězec"**, která textový řetězec automaticky zkonvertuje na pole bajtů.

Tady je celý program:

```
fn main() {
    println!("{:08x}", adler32(b"brambora"));
}

fn adler32(src: &[u8]) -> u32 {
        let mut a: u32 = 1;
        let mut b: u32 = 0;

        for c in src {
                a = (a + *c as u32) % 65521;
                b = (a + b) % 65521;
        }

        (b << 16) | a
}
```

V **println!** je v závorkách použita speciální syntaxe pro zobrazování čísel. **Dvojtečka** tento mód zapíná, **nula** říká, že chceme zarovnat pomocí nul, **osmička** znamená zarovnat na osm číslic, a **x**, že se má číslo zobrazit v hexadecimálním zápisu, který se u hashů primárně používá. Důvodem proto je to, že číslice v šestnáctkové soustavě vždycky zamená **4 bity**. Takže 8 hexadecimálních číslic nám dá **4** bajty, resp. 32 bitů, což je přesně tolik, kolik potřebujeme na zobrazení hashe této funkce.

Jen pro informaci, je možné použít následující písmena pro zobrazování čísel pomocí **println!()**:

- **x** - hexadecimálně, číslice A-F budou malá písmena
- **X** - hexadecimálně, ale číslice A-F budou velká písmena
- **o** - oktálově, tj. číslice 0-7
- **b** - binárně, jen nuly a jedničky :)

Domácí úkol
Za domácí úkol, který je až na hodinu po prázdninách, si zkuste implementovat jeden z hashovacích algoritmů **djb2**, **sdbm** nebo **loselose (ten je ale možná až moc jednoduchý :^))**, viz stránka http://www.cse.yorku.ca/~oz/hash.html

Program by měl vypadat podobně, jako ten s **Adler32**, akorát změníte tělo funkce a přejmenujete jí.

Ve všech příkladech z odkazu na **yorku.ca** je použita tato syntaxe:

```
while (c = *str++)
```

Toto je taková "céčkovina", která je ekvivalentní procházení bajtů pomocí **for c in src**, jak je to v rustovém příkladu na Adler32.

Typy v céčkových příkladech z této stránky jsou identické těmto Rustovým:

- **unsigned long -> u32**
- **int -> i16**
- **unsigned char -> u8**
- **char* -> &[u8]**

V jazyce C se píšou typy vždy a navíc před proměnnou místo za dvojtečku po názvu proměnné

Všechny z těchto funkcí vrací 32-bitový hash, tj. typ **u32**.

Jestli něčemu nerozumíte, zasekli jste se nebo si nevíte rady, klidně mi napište :)

Zítra si řeknem něco o hardware a třeba von Neumannově a harvardské architektuře, podle toho, kolik toho stihneme.

Díky,
LH
