Smyčky a match

Smyčky

- Používají se k opakování segmentů kódu
- Opakovat všechno ručně by trvalo moc dlouho a někdy programátor ani kolikrát se kód bude opakovat
- Hodí se například když programátor prochází (iteruje) nějakým seznamem, opakuje něco pořád dokola nebo komunikuje po síti
- Jedno ‘proběhnutí’ smyčky se nazývá iterace
- Rust má tři typy smyček - **loop**, **while** a **for**

Smyčka loop

- Nekonečná smyčka
- Za klíčovým slovek **loop** následuje blok kódu
- Když chcete smyčku ukončit, použije se klíčové slovo **break**

Příklad

```
fn main() {
    let mut i = 0;

    loop {
        println!("{}", i);
        i += 1;
        if i > 10 {
            break;
        }
    }
}
```

Smyčka while

- Opakuje kód dokud platí podmínka
- Podobně jako u if-u se píše za klíčové slovo **while**, před otevírací závorku bloku kódu a kulaté závorky kolem podmínky jsou nepovinné
- Zpřehledňuje kód, jehož opakování závisí na kontrole proměnné, výsledku funkce atd.

Příklad

```
fn main() {
    let mut i = 0;

    while i <= 10 {
        println!(“{}”, i);
        i += 1;
    }
}
```

Smyčka for

- Slouží ke zpracování všech prvků dané sbírky (např. posloupnosti čísel, seznamu jmen, slovníkových hesel..)
- Funkcí je identická k for-loopu v Pythonu
- Klíčové slovo **for** je následováno libovolným názvem, který je poté použit pro prvek, se kterým pracujeme v dané iteraci smyčky
- Libovolný název je následován klíčovým slovek **in**. To je následováno sbírkou se kterou pracujeme (buď v proměnné, nebo nově vytvořenou) a blokem kódu

For-loop 1

```
fn main() {
    // všechna celá čísla od 0 do 11, mimo 11
    for i in 0..11 {
        println!(“{}”, i);
    }
}
```

For-loop 2

```
fn main() {
    let seznam = vec![0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

    // všechna celá čísla z pole
    for i in seznam {
        println!("{}", i);
    }
}
```

Úkol

Napište program, který vypíše všechna čísla od 0 do 99, která

a) jsou dělitelná třemi
b) nejsou dělitelná čtyřmi
c) při dělení sedmi mají zbytek větší než dva

Kalkulačka v7.0

```
#[macro_use] extern crate text_io;

fn main() {
    loop {
        let a: f64 = read!();
        let b: f64 = read!();
        let op: String = read!();
        if op == “+” { … }
        // atd.
        else if op == “exit” { break; }
    }
}
```

match

- Když programátor potřebuje zkontrolovat určitou hodnotu a podle ní vyvodit několik závěrů, je zdlouhavé psát spoustu else-ifů
- Podobné jako **switch** v C

Ukázka

```
fn main() {
    let cislo = 5;
    match cislo {
        1 => println!(“tohle se nikdy nestane”),
        3 => println!(“taky ne”),
        5 => println!(“tohle ano”),
        _ => println!(“jiné hodnoty nám jsou jedno”),
    }
}
```

Kalkulačka v8.0

```
#[macro_use] extern crate text_io;
fn main() {
    loop {
        let a: f64 = read!();
        let b: f64 = read!();
        let op: String = read!();
        match op {
            “+” => println!(“{}”, a + b),
            “-” => println!(“{}”, a - b),
            // atd.
            “quit” => break,
        }
    }
}
```

DCV

- Program, který se zeptá uživatele na celá dvě čísla a vypíše všechna lichá čísla mezi nimy
- Předpokládejte, že uživatel není kretén a to, co zadá jsou vskutku čísla, a že první číslo zadá menší než druhé