Aritmetika + Proměnné část 2

A taky správa kódu

Výraz

- Cokoliv, jehož přímým výsledkem je hodnota
- V Rustu je mnoho věcí výraz funkce, bloky kódu, vzory, if, loop, match, aritmetické výrazy
- Aritmetické výrazy
 - Skládá se z operátorů a operandů
 - operand hodnota, která se účastní operace
 - operátor znak vyjadřující matematickou operaci
 - Příklad 1 + 2 * 6
 - 126 operandy
 - + * operátory
 - Výraz se skládá alespoň z jednoho operandu
 - Platí přednost násobení a dělení jako v matematice

Aritmetické operátory

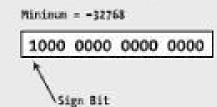
- + : sčítání 5 + 2 == 7
- : odečítání 5 2 == 3
- / : dělení 6 / 2 == 3
- * : násobení 5 * 2 == 10
- %: modulo/zbytek 10 % 3 == 1
- << : bitový posun vlevo 0011 << 1 == 0110</p>
- >> : bitový posun vpravo 0011 >> 1 == 0001
- | : bitové NEBO/OR 0011 | 1100 == 1111
- & : bitové A/AND 1101 | 1010 == 1000
- ^ : bitové exkluzivní nebo/XOR 1010 | 0110 == 1100

Čísla

- Všechno jsou bity
- Little-endianový systém 'nejmenší' bity na konci
 - U čísla 17 bude bit vyjadřující 1 na konci a bit vyjadřující 16 na začátku
- Dělení:
 - Celá čísla (integer)
 - N-tý bit z levé strany reprezentuje n-tou mocninu čísla 2
 - Bez znaménka všechny bity jsou čísla u8, u16, u32, u64
 - Se znaménkem jeden bit na znaménko i8, i16, u32, u64
 - Čísla s desetinou (plovoucí) čárkou
 - Složitější, mohou reprezentovat i nekonečna nebo NaN (Not a Number)
 - f32
 - f64

Pozn: Bitu nejvíce 'vlevo' se říká **nejvýznamnější bit**, bitu nejvíce vpravo se říká **nejméně významný bit**

16-Bit Signed Integers:





16-Bit Unsigned Integers:

Minimum = OU

0000 0000 0000 0000

Maximum = 65535U

1111 1111 1111 1111

All bits are data bits with unsigned types, so there is no sign bit.

Proměnná v Rustu

- Uchovává informaci hodnotu, která se nachází někde v paměti
- Staticky typované každá proměnná má jasně určený typ od spuštění do konce programu
 - proměnná co obsahuje textový řetězec se nemůže mávnutím kouzelného proutku změnit na proměnnou co obsahuje mandarinku
- Rustové proměnné mají životnost každá proměnná žije tak dlouho jak je dlouhý její blok kódu, nebo přesně, jak je potřeba
 - V C# nebo Pythonu žije proměnná tak dlouho, než jí smaže Garbage Collector velký popelář programování
- Ze základu neměnná mutabilita (proměnnost) se nastaví klíčovým slovem 'mut'
- Jedna proměnná může 'zastínit' druhou

- Typ lze výslovně specifikovat pomocí dvojtečky, ve většině případů ovšem Rust odvodí typ z kontextu
- = je operátor přiřazení hodnoty, jakmile dojde k přiřazení, program se dál o hodnoty ve výrazu nestará, pokud 'a = b * 2;', tak pozdější změna 'b' neovlivní 'a' a naopak

```
let a = 5 * 25; // neměnná proměnná :D -- 125
let mut b: i32 = a + 2; // proměnlivá proměnná -- 127
a = 10; // chyba
b = 42; // ok
let mut a: u8 = 10; //nové a skryje to staré, ': u8' přímo specifikuje typ
a = b * 2; // ok -- 84
b = a * 2; // ok -- 168
```

Kalkulačka v3.0

```
fn main() {
    let argumenty: Vec<_> = args().collect();
    let a: i32 = argumenty[1].parse().unwrap();
    let b: i32 = argumenty[2].parse().unwrap();
    println!("vysledek: {}", a + b);
}
```

- Spuštění: 'cargo run -- cislo1 cislo2'
- Když program nedostane oba parametry, zpanikaří (= vypíše chybu a zastaví vlákno to ukončí program)

Analýza

- → use std::env::args; import funkce ze standardní knihovny
 - funkce args mi dá iterátor iterátor == něco co umožňuje sekvenční procházení prvků jeden po jednom
- → fn main() { .. } funkce main(), poběží, když spustíme program
- → let argumenty: Vec<_> = args().collect(); vytvoří novou proměnnou, která obsahuje seznam argumentů příkazové řádky
 - ◆ args() mi dá iterátor, collect() ho 'posbírá' do seznamu
 - ◆ typ **Vec<>** znamená Vektor, prozatím vektor == seznam, _ nechá překladač domyslet typ
- → let a: i32 = args[1].parse().unwrap(); vezme hodnotu prvního argumentu (args[1]), pokusí se ho zkonvertovat na číslo (.parse()) a získá faktickou hodnotu výsledku (.unwrap())
- → println!("vysledek: {}", a + b); vypíše textový řetězec. {} se vymění za výsledek a+b

Úkol

- Změňte matematické operace
- Změňte počet čísel se kterými se pracuje
- Změňte číselné typy proměnných

Domácí úkol

Napište program, který si přečte tři čísla a vypočítá obvod kvádru o těchto stranách

Odevzdávání

- 1. Repozitář vytvoříte v naší organizace, gjkbot ho přesune
- Repozitář vytvořený Cargem přidáte jako zrcadlo pomocí 'git remote ..atd.'
 nebo klonujete prázdný repozitář a v něm vytvoříte nový projekt pomocí
 Cargo
- 3. Uděláte změny, revizi a pushnete
- 4. Když jste spokojení, přidáte k popisu repozitáře na Githubu tag 'r'