CALCULATOR PROJECT

cargo new calculator

use std::env::{args, Args};

fn main() {

    let mut args: Args = args();

    let first = args.nth(0);

    println!("{:?}", first);

}

Returns Some. Some is a veriant on the option enough

use std::env::{args, Args};

fn main() {

    let mut args: Args = args();

    let first = args.nth(1).unwrap();

    println!("{:?}", first);

}

Unwrap gets the variable asked, in this case first which is given on the cargo run – pprust returning this

PS C:\PP\RUST\calculator> cargo run -- pprust

Finished dev [unoptimized + debuginfo] target(s) in 0.01s

Running `target\debug\calculator.exe pprust`

"pprust"

Note: unwrapping on a non value cause the code to panic (need to give all values asked for in the code)

use std::env::{args, Args};

fn main() {

    let mut args: Args = args();

    let first = args.nth(1).unwrap();

    let operator = args.nth(2).unwrap();

    let second = arg.nth(3).unwrap();

    println!("{:?} {} {}", first, operator, second);

}

If we run cargo run – 1 + 2 something is missing and it will panic

PS C:\PP\RUST\calculator>

rustc ./src/main.rs

//HOW nth works.

fn nth(&mut self, n:usize) -> Option(String) {

    //assume n = 0

    //inner ["1","2"]

    self.inner.next() //returns "1"

    //calling next again result in second element:

    self.inner.next() // will return "2"

}

Nth moves to the next and set the 0 on the next one on the list, that’s why to get the stuff ordered and avoid panic instead of returning 0 1 2 as it was a static list, the way to obtain consecutive numbers of the array is the following code:

fn main() {

    let mut args: Args = args();

    let first = args.nth(1).unwrap();

    let operator = args.nth(0).unwrap();

    let second = args.nth(0).unwrap();

    println!("{:?} {} {}", first, operator, second);

}

And the launch it as:

cargo run -- 1 + 1 3 4 5 7 8

obtaining

Running `target\debug\calculator.exe 1 + 1 3 4 5 7 8`

"1" + 1

On the first one if you put 0 instead of 1 you get the name of the file calculator.exe

use std::env::{args, Args};

fn main() {

    let mut args: Args = args();

    let first = args.nth(1).unwrap();

    let operator = args.nth(0).unwrap().chars().next().unwrap();

    let second = args.nth(0).unwrap();

    let first\_number: f32 = first.parse().unwrap();

    let second\_number = second.parse::<f32>().unwrap();

    let result = operate(operator,first\_number,second\_number);

    println!("{:?}", result);

}

fn operate(operator: char, first\_number: f32, second\_number: f32) -> f32 {

    if operator == '+' {

        return first\_number + second\_number

    }

    else if operator == '-' {

        return first\_number - second\_number

    }

    else if operator == '/' {

        return first\_number / second\_number

    }

    else if operator == '\*' {

         first\_number \* second\_number

    }

    else {

        0.0

    } //add .0 \sking for f32, thre is implicit retunrs in rust, and no need for smicolon

}

    let operator = args.nth(0).unwrap()

operators have type string that generates a warning . because the function as it setups expect a value that is type char (string) we need to convert the operator into a character type. That can be done using the chars method on the string (unwrapped), the chars method returns an iterator over the characters of a string slice, the next method on top of chars, it’s a method that allows to access the first element of that slice, after that the unwrap method returns and option that allows to unwrap the value if it exists

    let operator = args.nth(0).unwrap().chars().next().unwrap();

fn output(first\_number: f32, operator: char, second\_number: f32, result: f32) -> String {

    format!("{} {} {} = {}", first\_number, operator, second\_number, result)

}

This function formats the result, format! Is a way to print that allows to customize the result

And its used here:

    println!("{:?}", output(first\_number, operator, second\_number, result);

modifying the previous one.

The next step is to use rust pattern matching expression instead of elseif statements.

So the operate function that was:..  
fn operate(operator: char, first\_number: f32, second\_number: f32) -> f32 {

    if operator == '+' {

        return first\_number + second\_number

    }

    else if operator == '-' {

        return first\_number - second\_number

    }

    else if operator == '/' {

        return first\_number / second\_number

    }

    else if operator == '\*' {

         first\_number \* second\_number

    }

    else {

        0.0

    } //add .0 \sking for f32, thre is implicit retunrs in rust, and no need for smicolon

Will change into

    match operator {

        '+' => first\_number + second\_number,

        '-' => first\_number - second\_number,

        '/' => first\_number / second\_number,

        '\*' => first\_number \* second\_number,

        \_ => 0

    }

Here the underscore part \_ => 0 is the else statement when none of the other conditions work

But returning a zero does not make sense, so its better to use the panic! Macro there

        \_ => panic!("The operator is not recognized")

Extend one option in the match to have multiple options, for example:

'\*' | 'x' | 'X'  => first\_number \* second\_number,

That allows that all x,X,\* are recognized as multiplicator operators.

Cargo build –release

This compiles the binary, the release flag, this provides some optimizations for the compiler and removes all debugging information running this while its compile the code

The code is found in the target directory, in the relasease directory as calculator.exe

The it can be launch as

.\target/release/calculator 3 X 10