Warning

This cheat sheet is in a reasonably useful state for basic things, but it does contain many errors and typos and pedagological mistakes of omission and ordering & etc.

Also note that Rust is still changing quite a bit in 2019-2022 so some of the below may be outdated/deprecated.

Rust in a Nutshell

- Syntax tokens somewhat similar to C / C++ / Go
- Ownership of memory enforced at build time
- Statically linked
- Not so Object-Orientish, tends to be Functional-ish
- Control flow using pattern matching, Option+Result enums
- Packages: 'cargo add' command, https://crates.io
- Testing: 'cargo test' command, #[test] unit tests, integration tests
- Concurrency: ownership, mutability, channels, mutex, crossbeam + Rayon packages
- Auto formatter: 'rustfmt filename.rs' (see rust-lang.org for installation)
- compiler engine: LLVM, no non-LLVM compilers yet
- To use raw pointers, low level, call C/C++: unsafe{} keyword + ffi package
- smart pointers and reference counted: Box, Rc, Arc
- online playgrounds: https://play.rust-lang.org, https://tio.run/
- A survivial horror game where griefing is ... oops wrong Rust

Hello World

See https://www.rust-lang.org for installation details.

```
fn main() {
    println!("Hello World");
}
```

```
$ rustc main.rs
$ ./main
Hello World
```

Hello Packages, Hello Tests, Hello Dependencies

```
$ rustup.sh         # install rust, see rust-lang.org for details
$ cargo new myproj  # start new executable project under myproj path
$ cd myproj         # cd into the new directory
$ ls -lR            # list our skeleton of files
src/main.rs         # main.rs, has main() entry point
Cargo.toml          # Cargo.toml defines packaging
$ $EDITOR Cargo.toml  # add dependencies and other details
```

```
[package]
name = "helloworld"
version = "0.1.0"
authors = ["ada astra <ada@astra.moon>"]

[dependencies]
serde = "1.0.80"
# edit main.rs to say "extern crate serde;" and "use serde::*;"
$ cargo add chrono # auto add dependency w/o wediting Cargo.toml
# edit main.rs to say "extern crate chrono;" and "use chrono::*;"
$ cargo build      # downloads dependencies + builds main.rs
$ cargo run        # runs program created from main.rs
$ cargo test       # runs tests (in parallel by default)
$ cargo test -- --test-threads=1  # run tests one at a time
$ cargo test -- --nocapture       # run tests, show output
$ cargo run --example fundemo -- --arg # run example with arg (./examples subdir)
$ cargo build --feature blargh     # build, enable the blargh feature of the crate
```

Mutability basics

```
let x = false;           // all variable bindings are immutable by default
x = true; // err         // compile error. can't change a variable which has an immutable binding
let mut p = false;       // "mut" designates a binding as mutable
p = true;                // ok, a variable with mutable binding can change;
```

types, variables, declarations, initialization

```
let x: bool = false;    // let keyword
let k = false;          // rustc can determine some types automatically
let y: char = '上';     // all chars are 4 bytes
let 上 = 5; //err       // error. identifiers must be ASCII characters
let a: i8 = -2;         // 8 bit signed integers, also i16, i32, i64
let b: u8 = 200;        // 8 bit unsigned integers, also u16, u32, u64
let n: f32 = 0.45;      // 32 bit float (automatcally converted+rounded from base-10 decimal to binary)
let n2 = 42.01f64;      // 64 bit float literal of the number 42.01 (approximately)
let r: [u8;3] = [3,4,5];          // array of 3 int, immutable, cannot grow or change values
let mut rm: [u8;3] = [3,4,5];     // same as r but mutable. cannot grow, but values can change.
let mut s1 = [0;500];             // array of 500 integers, each initialized to 0.
s1[0..200].fill(7);               // set the first 200 integers to the value 7
s1[400..].fill(5);                // set the last 100 integers to the value 5
let s2 = &r[0..2];                // slice of array, s==&[3,4]
let s3 = &r[0..2][0];             // index into slice, s==3
let s4 = &r[1..];                 // slice from index 1 to end
let s5 = &r[..2];                 // slice from beginning to index 2
let mut u:Vec<u8> = Vec::new();   // create empty vector of unsigned 8 bit int, can grow
let mut v = vec![3,4,5];          // initialize mutable vector using vec! macro
let w = vec![1,12,13];            // vectors can be immutable too
let z = (0..999).collect::<Vec<u32>>(); // init Vector from Range (0..999)
u.push( 2 );                      // append item to vector
u.pop();                     // vectors can pop, return+remove last input (like a stack)
v.contains(&3);              // true if vector contains value
v.remove(1);                 // remove the nth item from a vector...
v.append(u);                 // append v with u (u becomes empty ([]), both mutable)
v.extend(w);                 // extend v with w (v owns w, w can be immutable)
v.resize(200,0);             // make vector have 200 elements, set them to 0
v[0..100].fill(7);           // set the first 100 elements in v to the value 7
```

```
v[150..].fill(9);             // set the last 50 elements in v to the value 7
v.fill(9);                    // set all elements of v to have the value 9
let x = &w[1..];              // get a slice of a vector (a view into it's elements)
print("{:?}",x);              // [12,13];
let vs = v.len();             // length of vector
let (p,d,q) = (4,5,6);        // tuple() can assign multiple variables at once
print("{}",p);                // you can use them alone after tuple assignment
let m = (4,5,"a");            // tuples can have multiple different types as elements
let (a,b) = m.1, m.3;         // tuple dereference with .1, .2, .3
let (c,d) = m.p, m.q; //err   // error, cannot index into a tuple using a variable

let s = String::from("上善若水 "); // String is a heap variable. Strings are UTF8 encoded.
let s2 = "水善利萬物而不爭";       // "" literals are type &str, different from String
let s3 = &s;                  // & when prefixed to String gives &str
let s4 = s2.to_string();      // create String from &str
let s5 = format!("{}{}",s2,s3); // concatenate &str to &str
let s6 = s + s2;              // concatenate String to &str
for i in "말 한마디에 천냥 빚을 갚는다".split(" ") {print!("{i}");} // split &str
s.chars().nth(4);             // get nth char.
s.get(2..).unwrap(); // ERROR  // get substring failed because 上 was 3 bytes
s.get(3..).unwrap();        // 善若水
s.trim();                     // 上善若水, no trailing space
s.starts_with("上善");        // true
s.ends_with("水");            // true

let i4 = s.find('水').unwrap_or(-1); // index of character (not always a byte offset, b/c utf8)
let hellomsg = r###"                 // Multi-line &str with embedded quotes
 "Hello" in Chinese is 你好 ('Ni Hao')
 "Hello" in Hindi is नमस्ते ('Namaste')
"###;

let x = vec![5,12,13];          // indices into slices/arrays/vectors must be of type usize
let y = 2 as u8;                // using u8, u16, etc will not work
print!("{}",x[y]);              // error[E0277]: the type `[{integer}]` cannot be indexed by `u8`
let z = 2 as usize;             // usize is the pointer size. used in loops, vector length, etc
print!("{}",x[z]);              // ok.   there is also "isize" if usize needs to be signed

const BILBOG: i32 = 10;         // constant
static ORGOG: &str = "zormpf";  // static, global-ish variable
static FOOBY: i32 = 5;          // statics are only mutable inside unsafe{} blocks.
static Z_ERRMSG : [&str;2] = ["need input","need more input"]; // static strings

type Valid = bool;              // typedef ( make your own type names )

let mut v = vec![1u8,2u8,3u8];  // determine the type of expression expr by looking at rustc error
println!("{}",v.iter_mut());    // for example, if we want to know the type of v, build an error
println!("{}",v.iter_mut());    // type of v.iter_mut() is std::slice::IterMut<'_, u8>`
```

Operators

```
1 + 2 - 3 * 4 / 5  // arithmetic add, minus, multiply, divide
7 % 5              // modulo (remainder)
& | ^              // bitwise and, or, xor
<< >>              // leftshift, rightshift, will crash on overflow
// note that in C, overflowing << is actually undefined.
// Rust has multiple versions, each defined.
let a:u8 = 0b10110011; // print!("{:08b}",a); // 0b10110011 padded binary output
a.rotate_left(1)    // 01100111 circular bit rotation, out of left -> in at right
a.wrapping_shl(1)   // 01100110 this destroys the left-most bits that would cause overflow
```

```
a.overflowing_shl(1)// 01100110,false returns tuple (value,did it overflow the number type)
a.rotate_right(4)   // 11011001 circular bit rotation to the right
!a                  // 01001100 bitwise not
a == b != c < d <= e > f >= g  // logical comparison
a && b || c ! d    // logical boolean, and, or, not

let a = 5;         // pointer + dereference example
let b = &a;        // &a is 'address of a' in computers memory
let c = *b;        // *b is contents of memory at address in b (dereference)
print("{c}");     // 5

overloading: see struct
```

Run time errors, Crashing, panic, except, unwrap, Option, Result

```
panic!("oops");             // panic!() instantly crashes program
let v = vec![3,4,5];
let a = v[0];               // ok, normal lookup, a is 3
let b = v[12];              // will call panic! at runtime, v[12] doesn't exist


$ export RUST_BACKTRACE=1
$ cargo run    # will tell you exact line where panic occured, with call stack trace
```

Option - a type for functions that may return Some thing, or None thing

```
let v = vec![3,4,5];        // create Vector with three elements. then get() the element at index 12.
let c = v.get(12);          // there is no element at index 12, but this will not crash, get returns an Option
print!("{:?}",v.get(12));   // prints the word "None", since there is no 12th element in v, its only 3 long
print!("{:?}",v.get(0));    // prints the word "Some(3)", because there is an element at index 0
                            // Options have a value of either None, or Some(item), i.e. they are "Enums"
let e = v.get(0).unwrap();  // ok, 'unwrap' the Option returned by get(0), e is now 3
let d = v.get(12).unwrap(); // this crashes. 'unwrap' of a None Option will call panic!
let f = v.get(5).unwrap_or(&0); // unwrap_or gives a value if get() is None. f = 0
```

Option and Match - a control flow similar to **if else** but with more error checking at compile time

```
let x = v.get(12);
match x { Some(x)=>println!("OK! {x}"),  // print OK if v has 13th item
      None=>println!("sad face"), }   // otherwise print sad face
// you will get a compile-time error if you forget to handle both Some and None
```

Result - a type like Option but with Ok() and Err() instead of Some() and None.

```
match std::env::var("SHLVL") {  // env::var() returns a std::result::Result(<T,E>) enum type.
    Ok(v)=>println!("SHLVL = {v:?}"), // if OK, std::env returns value of Environment variable
    Err(e)=>println!("error message: {:?}",e.to_string()) }; // if not, error message

note there is also std::io::Result used a lot in file operations.
```

If Let - A control statemnt like match but with a no-op for None or Err()

```
if let Some(x) = v.get(12) { println("OK! {x}"); }
// we don't have to type out a handler for None. it does nothing.

if let Ok(x) = std::env::var("SHLVL") { println!("SHLVL = {x:?}"); }
// if the Result is Err, then nothing is done.
```

Option in particular can prevent the use of null pointers, preventing crashes one might see in C/C++.

```
struct Owlgr {
    name: String,
    fiznozz: Option<String>     // in C++ this might be a *char which could init as NULL
}

let owls = [Owlgr{name:"Harry".to_string(),fiznozz:None},
            Owlgr{name:"Tom".to_string(),fiznozz:Some("Zoozle".to_string())}];

for owl in &owls {
    match &owl.fiznozz {
        None=>println!("Owlgr named {} has no fiznozz!",owl.name),
        Some(x)=>println!("Owlgr named {} has fiznozz of {}",owl.name,x),
    }
    }
}

// note that we did not have to check for null pointers, nor derefernece any
// pointer. if we forgot to check for None, the compiler would give an error.
```

Note that there are no Exceptions. panic/Option/Result/multi-value-return are used instead.

Printing

```
println!("Hello, 你好, नमस्ते, Привет, ಹಬಿ");    // unicode text is OK
print!("Hi, is {}x{}={} ?",6,9,42);           // curly braces {} get replaced by arguments
Hi, is 7x9=42 ?
let (meepo,beepo) = (5,6);                    // print variables without using arguments..
print!("{meepo}:{beepo}");                    // .. by putting the names inside {}

let v = vec![1,2,3];
println!( "v[0] from {:?} = {}", v, v[0] )    // {:?} can print lots of special types
println!("{:02x?}",v);                        // {:02x?} can print 2 digit hex of vector
let s = format!( "x coord={}", p.X )          // print to string
s2 := fmt.Sprintf( "{e}", 17.0 )              // another way to print to string
use std::fmt::Write;                          // yet another way - like C++ stringstream
let mut s3 = String::new();                   // String implements fmt::Write so we can
match writeln!(s3,"Hello There") {            // write to a String like a file.
   Ok(_)=>(), Err(e)=>println!("error writing to string {} {:?}",s3,e),}
writeln!(s3,"Hello Too").unwrap_or(());       // the concise version w/o any error msg

// C / printf style formatted output:
println!("hex:{:x} bin:{:b} sci:{:e}",17,17,17.0); // hexadecimal, binary, etc.
//   "hex:11 bin:10001 sci:1.7e1"
// Pad with zeros:
println!("dec:{:#04} hex:{:#06x} bin:{:08b} sci:{:09e}",17,17,17,17.0);
```

```
//   "dec:0017 hex:0x0011 bin:00010001 sci:00001.7e1"
println!(" {:.40} ", 1.0f32/3.0f32 );  // print 40 digits of precision for floating point
//   "0.3333333432674407958984375000000000000000"
println!(" {:>4} {:>4} ", 232, 8 );    // pad as columns, width 4 spaces, align right
//   " 232    8"
let mut s=String::new();               // build string, concatenate over lines
s.push_str(&format!("{} {} ",1,2));
s.push_str(&format!("{} {}\n",3,4));    // "1 2 3 4\n"
let mut s2=String::new();              // alternate version, same goal
write!(s2,"{} {}",1,2).unwrap_or(());
writeln!(s2,"{} {}",3,4).unwrap_or(()); // "1 2 3 4\n"

println!("\u{2766}");                   // ❦  unicode floral heart, character hex 2766

// derive Debug can make your own structs, enums, and unions printable by {:?}
#[derive(Debug)]
struct Wheel{radius:i8}
println!("{:?}",vec![Wheel{radius:4},Wheel{radius:3},Wheel{radius:2}]);
// [Wheel { radius: 4 }, Wheel { radius: 3 }, Wheel { radius: 2 }]

// If you want to customize your debug output, you can implement Debug yourself
use std::fmt;
struct Wheel{radius:i8}
impl std::fmt::Debug for Wheel {
  fn fmt(&self, f: &mut fmt::Formatter)->fmt::Result{
    write!(f, "輪:徑[{}]", self.radius)
}}
println!("{:?}",vec![Wheel{radius:4},Wheel{radius:3},Wheel{radius:2}]);
// [輪:徑[4], 輪:徑[3], 輪:徑[2]]

// fmt::Display makes your own structs and enums printable with ordinary {} symbol
impl std::fmt::Display for Wheel{
  fn fmt(&self, f: &mut fmt::Formatter)->fmt::Result{
    write!(f, "W[{}]", self.radius)
   }
}

// Display for enums
pub enum Apple{PinkLady,HoneyCrisp}
impl std::fmt::Display for Apple {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        match self { Apple::PinkLady=>write!(f, "Ap:PLad"),
                     Apple::HoneyCrisp=>write!(f, "Ap:HonCr"),
             }
    }
}
```

loop, for, while

```
for i in 0..10 {print!("{},",x)};          // 0,1,2,3,4,5,6,7,8,9
for i in 0..10.rev() {print!("{},",x)};    // 9,8,7,6,5,4,3,2,1,0
for i in (0..10).step_by(2) {print!("{},",x)};              // 0 2 4 6 8
for i in (0..10).skip(1).step_by(2) {print!("{},",x)};      // 1 3 5 7 9
for i in (0..10).rev().step_by(2){print!("{},",x)};         // 9 7 5 3 1
for i in (0..=10).rev().step_by(2){print!("{},",x)};        // 10 8 6 4 2 0
for i in (0..=10).step_by(2){print!("{},",x)}     ;         // 0 2 4 6 8 10
for i in (0..9).rev().step_by(2){print!("{},",x)} ;         // 8 6 4 2 0
for i in (0..9).step_by(2){print!("{},",x)}       ;         // 0 2 4 6 8
for i in (0..10).cycle().skip(5).take(10){print!("{},",x)}  // 5 6 7 8 9 0 1 2 3 4
```

```
let v = vec![3,5,7];
for n in v { println!("{}",n) }             // for loop over vector

for (i, n) in v.iter().enumerate() {        // iterate with (index, item) tuple
    println!("{},{} ", i, n);}          // 0,3 \n 1,5 \n 2,7

let mut i = 0;                  // while loop
while i < 10 {print!("{}",i); i += 2;}      // 0 2 4 6 8
let mut j:u8 = 9;
while j > 0 {print!("{}",j); j -= 2;}       // 9 7 5 3 1 Panic! j is unsigned but goes below 0

let mut i = 0;                              // loop
loop { i=i+1; if i<10 { break; } };     // plain loop, exit with break;
let x = loop { i=i+1; if i>=10 {break i;} } // loop that returns value, x = 10
```

Iterators as an alternative to loops

```
let v = vec![3,5,7,11];                     // vector to iterate
v.iter().for_each(|x| print!("{} ",x));    // for_each over iterator, 3 5 7 11
print!("{}",v.iter().fold(0,|a,i| a+i));   // adds 0+1+2+3, prints 6.
// for more info , see iterators/functional section below


// While Let, can be used in situations where we expect Option::Some() for several iterations,
// but we expect a Option::None() to be at the end of the iterations. For example:
let mut x = (0..12).filter(|x| x%2==0);          // x is an iterator
while let Some(i) = x.next() {print!("{}:",i);}  // While loop over the iterator
// 0:2:4:6:8:10:   // prints the even numbers between 0 and 12
```

Parallel processing

```
extern crate rayon;
use rayon::prelude::*;

fn main() {
    let mut v = Vec::new();  // create a vector of floats, to multiply each by 0.9
    for i in 0..1024*1280 { v.push(i as f32); }
    v.iter_mut().for_each(     |x| *x = *x * 0.9 ); // single thread version
    v.par_iter_mut().for_each( |x| *x = *x * 0.9 ); // multiple threads version

    let v = (0..999).collect::<Vec<u32>>(); // par_iter is slightly different
    let tot = v.par_iter().map(|i|i*i).reduce(||0,|a,x|a+x); // parallel sum of squares
    // see https://docs.rs/rayon/latest/rayon/iter/trait.ParallelIterator.html#method.fold
    // for info on fold, reduce, map, etc, in Rayon

    very_slow_function1(); // two single threaded functions that take a long time
    very_slow_function2();

    rayon::join( || very_slow_function1()    // run them in parallel if appropriate
             || very_slow_function2() );

    s = "VeryLargeString ...pretend goes on for 10Mb "; // imagine 10Mb string
    s.chars().par_iter().do_stuff() // error // par_iter() cant work on string because
                                // it doesnt know where to cut the byte boundaries
    let v = s.iter().collect::<Vec<char>>9); // so convert to vector of char each 4 bytes
    v.par_iter().map(|ch| ch.to_ascii_lowercase()).do_stuff() // now you can
```

```
}
```

```
$ cargo add rayon  # add rayon dependency
$ cargo run        # installs rayon, runs program
```

```
channels: todo
mutex: todo
concurrency: todo
```

Smart pointers

Box is a smart pointer. It is not possible to become null or to point to invalid memory.

As an example, consider a tree data structure. In C you might have nodes that have pointers to child nodes, and then at the leaf nodes , the child node pointers are NULL.

In Rust, the nodes have Boxed pointers to child nodes. The Boxes are all wrapped with an Option. So if there is no child, the Option is None. If there is a child, it is Some(Box(childnode))

```
struct Node { data: char, leftchild: Option<Box<Node>>, rightchild: Option<Box<Node>> }
let mut x = Node{data:'x',leftchild:None,rightchild:None};
let y = Node{data:'y',leftchild:None,rightchild:None};
let z = Node{data:'z',leftchild:None,rightchild:None};
(x.leftchild,x.rightchild) = (Some(Box::new(y)),Some(Box::new(z)));
// Node { data: 'x',
//        leftchild: Some(Node { data: 'y', leftchild: None, rightchild: None }),
//        rightchild: Some(Node { data: 'z', leftchild: None, rightchild: None }) }
```

Another way is to use Enum variants for the different node types, then you dont even need to use Option at all. Each node is actually a different Variant so the leaf nodes do not even have children they only have data, while the inner branch nodes have children but do not have data.

See below section on "Enums - not like C" for more on how Enums work in Rust.

```
enum Node { Branch(Box<Node>,Box<Node>), Leaf(char) }
let y = Node::Leaf('y');
let z = Node::Leaf('z');
let x = Node::Branch(Box::new(y),Box::new(z));
// print!("{:?}",x); // Branch(Leaf('y'), Leaf('z'))
```

https://doc.rust-lang.org/std/boxed/index.html

https://rosettacode.org/wiki/Huffman_coding#Rust

Arc, RC - reference counted pointers. Todo

Functions and closures

```
fn add( a:i8, b:i8 ) -> i32 { b + a }     // 'return' keyword optional
fn getcodes(a:i8)->(i8,i32){ (9,a*99) }   // multi return via tuples
```

```
let (x, s) = getcodes( 3, 56 );           // assign multi-return w tuples
fn multy(a:i8,b:i8=5)->i16{a*b}   //error // Rust has no default parameters.
fn multy(a:i8,b:Option<i8>)->i16 {        // but you can fake it with Option unwrap_or
  a * b.unwrap_or(5) }                    // unwrap_or(x) means x is the default value
fn main(){ print!("{}",multy(3,None));}   // pass None to the func, result here=15
fm main(){ print!("{}",multy(3,Some(4));} // awkward, tho, as normal calls require Some()
fn f(t:i8) {                              // nesting functions is OK
  fn g(u:i8) { u*5 };                     // g nested inside f
  let a = t + g(2);  }                    // g can be called from inside f
fn f2(t:i8) {             // however, nested functs cannot access outside variables
  let mut m = 2;          // m is declared outside the scope of g2 block
  fn g2(u:i8){u*5 + m};}  // error[E0434]: can't capture dynamic environment in a fn item

// function pointers
fn addtwo(t:i8)->i8{t+2}; // simple function, adds 2 to argument.
println!("{}",addtwo(5)); // prints 7
let fp = addtwo;          // fp = function pointer to addtwo function
println!("{}",fp(5));     // now we can call fp() just like we called addtwo
fn f<F>(fp: F) where F: Fn(i8)->i8 { println!("{}",fp(1)) }
// 'where F:Fn' lets us build a function that can accept another function as an argument
f(fp);  // call function f, passing a pointer to the 'addtwo' function. result=3

type ZFillCallback = fn(bottom:u32,top:u32)->u32;  // typedef of a function

fn maximum(t:i8,...) {} // error, can't have variable number of arguments.
                        // only macros! can be Variadic in Rust (see below)

// closures
let c = |x| x + 2;    // define a closure, which is kinda like a lambda function
let a = c(5);         // closures can be called, like functions. result = 7
let value = 5;        // a closure can also read values outside its scope
let d = |x| value + x;// d(n) will now add 'value' to any input n. (in this case,5)
fn f<F>(fp: F) where F: Fn(i8)->i8 { println!("{}",fp(1)) }  // f takes a function as an argument
f(c);                 // a closure can be passed, like a function pointer, result = 3
f(|x| x * value);     // and a closure can be anonymous, without a name. result = 5

for i in 0..4.filter(|x| x>1) // anonymous closures are used often with iterators (see below)
print!("{} ",i)               // 2 3  (0 1 2 3 filtered to only values greater than 1)
```

Unit tests, integration tests

Unit tests, placed in the same file as the code being tested

```
./src/lib.rs:

pub fn process(v:&mut Vec<u8>)->&Vec<u8>{ v.update(|x| f(x)) } // main function called by users
fn f(x:u8)->u8 { x*x }   // small piece of our code, to test in unit testing

#[cfg(test)]        // cfg -> section will only compiled during 'cargo test'
mod tests {         // namespace helper
    use super::*;   // bring in our functions above
    #[test]         // next function will be a single test
    fn test_f() { assert!(f(4)==16); } // test f() by itself (unit)
}
```

Integration tests, for overall crate, lives under ./tests/*.rs

```
./tests/file.rs:         // will only be built dring 'cargo test'
extern crate mypackage;  // include package we are testing
#test                    // treat next function as a test
fn bigtest() {           // not a unit test. instead, test overall code
    let mut d = vec![1,2,3];              // set up some typical data users would have
    let expected_results = vec![1,4,9];    // some results we expect
    assert!(process(d)==expected_results); // test what a user would typically call, process()
}
```

```
$ cargo test           # test build, will include cfg(test) sections
-> test_f passed       # cargo reports on passed tests
-> test_bigtest failed # cargo reports on failed tests
```

Documentation

rust-doc and cargo doc allow automatic building of html documentation for code. precede documentation of your code with three slashmarks instead of the normal two slashmarks, like so:

```
/// blorg() returns the blorgification of input x
/// # Details
/// this code implements the krishnamurthi procedure
/// for blimfication of zorgonautic primes
/// # Arguments
/// * `x` - typically a square number
/// # Safety
/// Cannot panic unless x overflows u64
/// # Example
///     let n = blorg(36);
fn blorg(x:u64)->u64 {
   x+x*x
}
```

Then run rust-doc or cargo doc and view the result.

```
$ cargo doc
$ firefox target/doc/cratename/index.html
```

Good examples of the results are on https://crates.io

If, conditionals, patterns, match, control flow

```
fn zorgnaught_level() -> i32 { 3 }

let (zoogle, noogle, poogle) = (3,4,5);

let x = zorgnaught_level();

if x == zoogle {                  // normal if else, like C, Pascal, etc
    print!("zoogle")
} else if x == noogle {
    print!("noogle")
```

```
} else if x == poogle {
    print!("poogle")
} else {
    print!("unknown zorgnaught level");
}

match x {           // match... mostly works on literals not variables
    3 => print!("zoogle"),    // "=>" signifies a branch or leg of the match
    4 => print!("noogle"),    // have as many=> legs as you want
    5 => print!("poogle"),    // end each leg with a comma ,
    _ => print!("unknown zorgnaught level"), // underscore _ will match anything not previously matched
}

 // match... can work on enums too
pub enum Zorglvl{Zoogle,Noogle,Poogle}

fn zorgnaught_level2() -> Zorglvl { Zorglvl::Zoogle }

let x = zorgnaught_level2();

match x {
    Zorglvl::Zoogle => print!("zoogle"),    // "=>" signifies a branch or leg of the match
    Zorglvl::Noogle => print!("noogle"),    // have as many=> legs as you want
    Zorglvl::Poogle => print!("poogle"),    // end each leg with a comma ,
        // don't need underscore, enum checks all, covers all
}

let x = [1i8,2i8];                  // match patterns can be structs, enums, arrays, etc
let y = match x {                   // match can 'return a result' to y
        [1,0] => "1,0",             // match a specific array
        [2,0]|[4,0] => "2,0 or 4,0",  // |, binary or, can be used in patterns
        [_,2] => "ends with 2",       // [_,2] will match [0,2] [1,2] [2,2], [3,2], etc
        _ => "other"
};
println!("{}",y);                   // "ends with 2"


let m = 3;              // match patterns can only be constant, but you can
let n = 4;                          // do similar things with "match guard", an if statement
let y = match (n,m) {               // which is inside of a => arm. First, we match tuple (n,m)
        (0,0) => "ok",              //  this leg matches any tuple with first element 0, return ok
        (3,_) if m%5==0 => "ok",    //  this leg matches when first element=3, and second divisible by 5
    (_,_) if m%3==0 => "ok",    //  this leg matches any tuple where the second element is divisble by 3
        _ => "stop",                //  this leg matches anything else.
};

let hour = get_24hr_time();         // patterns can be integer ranges (x..=y)
ampm = match hour {                 // however it has to be inclusive
    0..=11 => "am"              // 1..11 is not ok, 1..=11 is ok.
    12..=23 => "pm"
    _=> "unknown"
};

let x = 5i32;                         // we can use the match value in match arms,
let y = match x-1 {                   // this is also called "binding with @", like so:
    0..=9 => 7,                   // since x is 5, x-1 is 4 and 4 is in 0..=9, so y=7
    m @ 10..=19 => m*2,           // if x-1 was 14, m becomes 14, so y would be 28
    _=> -1,                       // if x-1 was >=20, y would become -1
};

let mut v = vec![0;4096];               // match also works with Result<>
match File::open("/dev/random") {
```

```
    Ok(f)=>f.read(&v),
    Err(why)=>println!("file open failed, {}",why),
}

let v = vec![1,2,3];            // match works with Option too
let n = 1;
match v.get(n) {
    Some(x) => println!("nth item of v is {}",x),
    None => println!("v has no nth item for n={}",n),
};
```

See also: While let, if let.

Ownership, Borrowing, References, Lifetimes

Resources have exactly one owner. They can be 'moved' from one owner to another.

```
// stack memory, no moves, only copies
let a = 5;
let b = a;  // ok
let c = a;  // ok

// heap memory
let a = String::new();
let b = a;  // 'move' of ownership from a to b
let c = a;  // error. cannot "move" a again, b already owns it

// heap memory + function call
let a = String::new();
let b = a;  // 'move' of ownership from a to b
fn f(t:String) { } // function takes ownership of the variable passed as t
f(a); // error. f(a) would move a into f(), but a was already moved into b
```

Borrowing is an alternative to moving. It is done with References & (memory addresses)

```
// heap memory, using borrows and references instead of moves
let a = String::from("☪☯ॐ✡γ$†");
let b = &a;  // this is borrowing, not moving, a to b
let c = &a;  // it is OK to have more than one borrower of non-mutable variables
println!("{}",a);    // ☪☯ॐ✡γ$†
println!("{:p}",&a); // 0x7ffcffb6b278
println!("{:p}",b);  // 0x7ffcffb6b278  // b and c hold the address of a
println!("{:p}",c);  // 0x7ffcffb6b278
println!("{:p}",&b); // 0x7ffcffb6b290  // b and c are distinct variables
println!("{:p}",&c); // 0x7ffcffb6b298  // with their own addresses
```

However borrowing has special rules regarding mutability.

In a block, only one of these statements can be true for a given resource R

- The program has one or more immutable references to R
- The program has exactly one mutable reference to R

```
// example error[E0502]: cannot borrow `a` as mutable because `a` is also borrowed as immutable
```

```
let mut a = 5;
let b = &a;
let c = &mut a;

// example error[E0499]: cannot borrow `a` as mutable more than once at a time
let mut a = 5;
let b = &mut a;
let c = &mut a;
```

Another way to think about it is Shared vs Exclusive

- A plain old & borrow is a Shared Reference, since multiple people can share at once
- A &mut reference is an Exclusive reference, only one can have it.

Lifetime in Rust: Resources are destroyed, (their heap memory is freed), at the end of a 'scope'. Their owners are also destroyed. That is the point of ownership - so that resources won't be accessed after they are destroyed, which is the source of a huge number of errors in C programs.

Borrowed resources are not destroyed when the borrowed reference itself goes out of scope. However the borrow cannot "outlive" the destruction of the original resource nor it's owner.

Take, for example, the case where we borrow a variable via `&`. The borrow has a lifetime that is determined by where it is declared. As a result, the borrow is valid as long as it ends before the lender is destroyed. However, the scope of the borrow is determined by where the reference is used.

```
fn main() { // main block starts
    let i = 3; // Lifetime for `i` starts. ────────────────┐
    //                                                     │
    { // new block starts                                  │
        let borrow1 = &i; // `borrow1` lifetime starts. ──┐│
        //                                                ││
        println!("borrow1: {}", borrow1); //              ││
    } // block for `borrow1 ends. ─────────-──────────────┘│
    //                                                     │
    //                                                     │
    { // new block starts                                  │
        let borrow2 = &i; // `borrow2` lifetime starts. ──┐│
        //                                                ││
        println!("borrow2: {}", borrow2); //              ││
    } // block for `borrow2` ends. ──────────────────────┘│
    //                                                     │
}   // main block ends, so does Lifetime ends. ──────────┘
```

Structs

```
struct Wheel{ r:i8, s:i8};  // basic struct, like C, pascal, etc. r = radius, s = number of spokes
struct badWheel{ mut r: i8, mut h: i8, }; // error, mut keyword doesnt work inside struct
let w = Wheel{r:5,s:7};   // new wheel, radius 5, spokes 7, immutable binding
w.r = 6; // error, cannot mutably borrow field of immutable binding
let mut mw = Wheel{r:5,s:7}; //  new mutable wheel. fields inherit mutability of struct;
mw.r = 6;  // ok

impl Wheel {              // impl -> implement methods for struct, kinda like a C++ class
        fn new(r: isize) -> Wheel { Wheel { r:r, s:4 } }  // our own default
```

```
        fn dump(    &self) { println!("{} {}",self.r,self.s); }  // immutable self
    fn badgrow(    &self) { self.s += 4; } // error, cannot mutably borrow field of immutable binding
        fn  okgrow(&mut self) { self.s += 4; } // ok, mutable self
};
w.dump();    // ok , w is immutable, self inside dump() is immutable
w.okgrow();  // error, w is immutable, self inside okgrow() is mutable
             //  cannot borrow immutable local variable `w` as mutable
mw.dump();   // ok, mw is mutable, self inside dump is immutable.
mw.okgrow(); // ok, mw is mutable, self inside grow() is mutable.

#[derive(Default,Copy,Clone,Debug,PartialEq)] // automatic implementations
struct Moo {x:u8,y:u8,z:u8,}
let m1:Moo=Default::default();            // Default, x=0 y=0 z=0
let m2:Moo=Moo{x:3,..Default::default()}; // Default, x=3 y=0 z=0
let (mut n,mut k)=(M{x:-1,y:-1,z:-1},Default::default());
n=k;                          // Copy
vec![M{x:0,y:1,z:2};42];      // Clone
println!("{:?}",n);           // Debug, formatter
if n==k {println!("hi");};    // PartialEq, operator overloading

// customized operator overloading
impl PartialEq for Wheel{ fn eq(&self,o:&Wheel)->bool {self.r==o.r&&self.s==o.s} }
if mw == w { print!("equal wheels"); }

// default values for members of struct
#[derive(Debug,Default)]
struct Wheel1{ r:i8, s:i8};
println!("{:?}",Wheel1{..Default::default()});        // Wheel1 { r: 0, s: 0 }

// custom default values for members of struct
#[derive(Debug)]
struct Wheel2{ r:i8, s:i8};
impl Default for Wheel2 { fn default() -> Wheel2{Wheel2 { r: 8, s: 8 }}  }
println!("{:?}",Wheel2{..Default::default()});        // Wheel2 { r: 8, s: 8 }
println!("{:?}",Wheel2{r:10,..Default::default()});   // Wheel2 { r: 10, s: 8 }

#[derive(Debug)]                      // Initialize one struct from another
struct Apple {color:(u8,u8,u8),price:f32};
let a = Apple{color:(100,0,0),price:0.2};
let b = Apple{color:(9,12,38),..a };      // this is called "struct update"
```

Structs with function pointers as members

```
struct ThingDoer {
    name: String,
    do_thing: fn(x:&Vec<u8>) -> u8,
}
fn baseball(v:&Vec<u8>)->u8{ 42 }
fn main (){
    let adder = ThingDoer{name:"addthing".to_string(), do_thing:|n| n.iter().fold(0,|a,b|a+b)};
    let multer = ThingDoer{name:"multhing".to_string(), do_thing:|n| n.iter().fold(1,|a,b|a*b)};
    let nother = ThingDoer{name:"nothing".to_string(), do_thing:|n|baseball(&n)};
    let (f1,f2,f3) = (adder.do_thing,multer.do_thing,nother.do_thing);
    let v = vec![1,2,3,4];
    println!("{} {} {}",f1(&v),f2(&v),f3(&v));
}
```

Enums - not like C

Enums in Rust do more than Enums in C/C++. They are actually more like Tagged Unions or ADT / Algebraic Data Types from Functional programming languages. Other places call them Sum Types

```
enum Fruit { Apple, Banana, Pear }
let x = call_some_function( Fruit::Apple );
enum Errs { ErrOK = 3, ErrFile = 5, ErrFire = 12 } // custom Discriminats (integers)
enum Planet { Earth = 3, Mars, Jupiter } // mars will be 4, Jupiter 5.

enum Blorg {     // enums can have different types as members
 Flarg(u8),
 Blarg(u32),
 Norg(String),
 Florg(bool)
}

let x = Blorg::Flarg(1); // enums can be detected with a match
match x {
    Blorg::Flarg(1) => println!("x is a Flarg with value of 1!"),
    Blorg::Flarg(_) => println!("x is a Flarg with non-1 value"),
    Blorg::Norg(_) => println!("x is a Norg"),
    _ => println!("neither Flarg nor Norg"),
}

// Enums can also derive traits
#[derive(Clone,Debug,PartialEq)]  // cannot derive Default on enum
enum ColorMapData {
    OneByteColors(Vec<u8>),
    FourByteColors(Vec<u32>),
}

// Enums can also have methods / functions, using impl
// the key is that inside the function, pattern matching is used
// to determine which variant of the Enum the function receives
impl ColorMapData {
  fn description(&self)->String {
    let (numcolors,numbytes) = match self {
      ColorMapData::OneByteColors(x) => (x.len(),"1"),
      ColorMapData::FourByteColors(x) => (x.len(),"4"),
    };
    format!("ColorMap with {} colors, {} bytes per color",numcolors,numbytes)
}}

// functions can take Enum as arguments
fn examinecm( c:ColorMapData ) { println!("{}",c.description()); }
let ca = ColorMapData::FourByteColors(vec![0xFFAA32FFu32,0x00AA0011,0x0000AA00]);
let cb = ColorMapData::OneByteColors(vec![0,1,3,9,16]);
examinecm(ca);
// ColorMap with 3 colors, 4 bytes per color
examinecm(cb);
// ColorMap with 5 colors, 1 bytes per color
```

Using strum, a reflection package, you can iterate an enum's values

```
$ cargo add strum strum_macros
use strum::IntoEnumIterator;
#[derive(Debug, strum_macros::EnumIter)]
```

```
enum Tofu { Smooth, Firm, ExtraFirm }
Tofu::iter().for_each(|d|print!("{d:?},")); // Smooth,Firm,ExtraFirm
```

Collections, Key-value pairs, Sets

HashMap, aka associative array / key-value store / map

```
use std::collections::{HashMap};
let mut m = HashMap<char,i32>::new();
m.insert('a', 1);                   // key is 'a', value is 1
let b = m[&'a'];                    // [] lookup, this crashes at runtime if 'a' is not in map
let c = m.get(&'a').unwrap_or(&-1); // .get(), c == -1 if a is not in map. no crash.
match m.get(&'a') {                 // deal with map get() lookup using Match + Option
    Some(x)=>println!("a found in map, value is {}",x),
    None=>println!("a not found in map"),
}
if let Some(x) = m.get(&'a') {     // deal with map get() lookup using if let + Option
    println!("a found in map, value is {}",x);
}  // if 'a' is not in map, do nothing

*m.get_mut(&'a').unwrap() += 2;  // change a value inside a map
```

Concise initialization with from and from_iter using tuples of (key,value)

```
let mut phonecodes = HashMap::from(  // from uses a fixed size array
  [("Afghanistan",93),("American Samoa",1),("Ukraine",380)]   );
let mut squares:HashMap<u32,u32> = HashMap::from_iter(
  (0..24).map(|i| (i, i*i))   );     // from_iter uses an iterator
println!("{:?}",phonecodes); //{"Afghanistan: 93, Ukraine: 380...
println!("{:?}",squares); //{10: 100, 3: 9, 1: 1, 13: 169, ...
```

HashSet

```
use std::collections::HashSet;
let mut squares = HashSet::new();
squares.insert(0);
squares.insert(4);
let b = squares.contains(&4);   // b==true
```

Macros

Does not act like a preprocessor. It replaces items in the abstract syntax tree

```
macro_rules! hello {
    ($a:ident) => ($a = 5)
}
let mut a = 0;
println!("{}",a); // 0
hello!(a);
println!("{}",a); // 5

macro_rules! bellana {
```

```
    ($a:expr,$b:expr) => ($a + $b)
}
println!("{:?}",bellana!(5,(9*2))); // 23

macro_rules! maximum {
    ($x:expr) => ($x);
    ($x:expr, $($y:expr),+) => ( std::cmp::max($x, maximum!($($y),+)) )
}
maximum!(1,2,3,4);   // 4

macro_rules! dlog {
    ($loglevel:expr, $($s:expr),*) => (
        if DLOG_LEVEL>=$loglevel { println!($($s),+); }
    )
}
let DLOG_LEVEL=5;
dlog!(4,"program is running, dlog:{}",DLOG_LEVEL);  // "program is running, dlog:5"

/*
designators:
block   // rust block, like {}.        expr    // expressions
ident   // variable/function names.    item    //
pat     // pattern.                    path    // rust path
stmt    // statement.                  tt      // token tree
ty      // type.                       vis     // visibility qualifier
*/
```

Arrays, Slices, Ranges

Arrays

```
let arr: [u8; 4] = [1, 2, 3, 4]; // immutable array. cant change size.
let mut arrm: [u8; 4] = [1,2,3,4]; // mutable array. cant change size.
let n = arr.len();     // length of array = 4 items
let s1 = &arr[0..2];   // slice of underlying array
let n2 = s1.len();     // length of slice = 2 items
println!("{:?}",s1);   // 1 2, contents of slice
let s2 = &arr[1..];    // slice until end
println!("{:?}",s2);   // 2 3 4 contents of slice until end
let sm = &mut arrm[0..2];  // mutable slice
sm[0] = 11;                // change element of mutable slice,
println!("{:?}",sm);       // 11 2 3 4
                           // underlying array was changed
println!("{:?}",arrm);     // 11 2 3 4

let z = [1,6,1,8,0,0,3];
println!("{:?}",z[0..4]);   // error - not a slice
^^^^^^^^^^^^^^^^^^^^^^^^^ doesn't have a size known at compile-time
println!("{:?}",&z[0..4]);  // OK to take a slice
// 1,6,1,8

// pass array slice to function
fn dostuff(x:&mut [u8]) {
    x[0] = 5;
    println!("{:?}  {}",x,x.len()); // 5 2 3 4   4
}

fn main() {
    let mut arr: [u8; 4] = [1, 2, 3, 4];
```

```
    dostuff( &mut arr );
}
```

Get(), a byte index into a UTF8 String

```
let sa = String::from("a"); // get() - byte index slice from a String
let sga = sa.get(0..1); // Some("a"). get returns Option, but not a Option<char>
let s = String::from("上善若水"); // get() with a UTF8 String where char.len()!=1
let sg0 = s.get(0..0);  // Some("") , the empty string.
let sg1 = s.get(0..1);  // None, because the first byte of 上 is not a utf8 char
let sg2 = s.get(0..2);  // None, same reason
let sg3 = s.get(0..3);  // Some("上"), this works because 上 in utf8 is 3 bytes
let sg4 = s.get(0..4);  // None again, because we now have 上 + 1 other byte
// see 上 at http://www.unicode.org/cgi-bin/GetUnihanData.pl?codepoint=4E0A
```

Split_At_Mut - Mutability and References into a Vector

For the special case for getting references to items within a vector:

Imagine we have three Wheels (struct W) with radius (W.r) and they are inside a vector v. We start with wheels of radius 3,4,5 and want to change it to be radiuses of 2, 4, 8.

```
#[derive(Debug)]
struct W{ r:u32 }       // wheel struct
let mut v = vec![W{r:3},W{r:4},W{r:5}];   // vector of structs
let wheela = &mut v[0];      // mutable reference to Wheel with radius 3
let wheelb = &mut v[2];      // compile error! two mutables for one variable!
//  error[E0499]: cannot borrow `v` as mutable more than once at a time
wheela.r = 2;  // nope
wheelb.r = 8;  // nope
```

The borrow checker treats the entire vector as one gigantic variable, so you can't edit part of it with a mutable reference and then edit another part, because the parts aren't considered parts. They are considered as if you are editing the variable v.

But there is a workaround built into the language. It is called..

split_at_mut

It can create mutable slices, which allow mutable access to the vector.

```
 #[derive(Debug)]
struct W{ r:u32 }       // wheel struct
let mut v = vec![W{r:3},W{r:4},W{r:5}];   // vector of structs
let (l, r) = v.split_at_mut(2);  // split after the wheel with r4
let wheela = &mut l[0];  // first item of left part of split
let wheelb = &mut r[0];  // first item of right part of split
wheela.r = 2;       // no problem
wheelb.r = 8;       // no problem
println!("{:?} {:?}",l,r); // [W { r: 2 }, W { r: 4 }] [W { r: 8 }]
println!("{:?}",v); // [W { r: 2 }, W { r: 4 }, W { r: 8 }]
```

Object-Oriented alternatives

Inheritance and Polymorphism - Rust doesn't have the object-oriented style of these things. Alternatives:

- "composition" (struct within struct),
- Traits (sort of like Interfaces),
- Enums (which can do much more than C enums, they are more like Tagged Unions aka Algebraic Data Types aka Sum Types)
- "NewType" pattern, aka Tuple Structs where for example you say struct MyWrapper(u32) to wrap u32 and then impl your own methods on MyWrapper to mimic u32, along with derive_more and implement deref (u32 could be any other struct or type from another external crate).
- todo - describe Dynamic Trait Objects
- todo - describe Generics

Files

```
use std::fs::File;         // File handling module
use std::io::{Write,Read}  // read and write capabilities for File
use std::fs::OpenOptions;  // specialized version of File

// read file, non-crashing example
let filename = "test.txt";
match File::open(filename) {
    Err(why) => println!("failed to open file '{}': {}", filename, why),
        Ok(mut f) => {
            println!("ok, opened {}",filename);
                let mut data = String::new();
                match f.read_to_string( &mut data ) {
                    Err(why) => println!("failed to read {}, {}",filename,why),
                        Ok(n) => println!("read {} bytes",n),
                };
    },
}

// crash-on-error examples ( unwrap() calls panic! on error, crashes the program )
let mut s = String::new();
File::open("test.txt").unwrap().read_to_string(&mut s).unwrap();    // read into s
File::create("output.txt").unwrap().write(s.to_bytes()).unwrap();   // write string
f = File::create("output.txt").unwrap();                            // create if doesnt exist
f = OpenOptions::new().write(true).truncate(true).create(true).open("out.txt").unwrap();       // create if doesnt exist
f = OpenOptions::new().append(true).open("out.txt").unwrap();       // append
f = OpenOptions::new().append(true).create(true).open("out.txt").unwrap(); // append + create
write!(f,"{}",s).unwrap(); // write formatted string, given file object f

let mut v = vec![];                                                 // read binary data, vec of u8
f = File::open("test.bin").unwrap().read_to_end(&mut v);            // without using buffering

use std::io::{self,BufRead};  // read from stdin aka standard input
let line = io::stdin().lock().lines().next().unwrap().unwrap();

let x = include!("datafile.txt"); // include external data file, example: vec![0,0,0];

// handle errors inside a io::Result-returning function with the question mark ?
fn readfunc()  -> std::io::Result<()> {
  let mut tmpbuf = vec![0u8;4096];
  let mut f = File::open("somefile.bin")?;   // instead of unwrap, just return error
  let count = f.read( &mut tmpbuf )?;        // instead of match, just return error
  Ok(())                                     // we got this far, so there's no error
}
```

```
// read binary data in a loop to a buffer
fn readfunc2()  -> std::io::Result<()> {
  let mut tmpbuf = vec![0u8;4096];
  let mut f = File::open("somefile.bin")?;
  loop {
    let numbytes_read = f.read( &mut tmpbuf )?;
    println!("read data {}",numbytes_read);
    if numbytes_read==0 { break Ok(()); }
  }
}

// same loop as above, but with While Let
fn readfunc3()  -> std::io::Result<()> {
  let mut tmpbuf = vec![0u8;4096];
  let mut f = File::open("somefile.bin")?;
  while let numbytes_read = f.read( &mut tmpbuf )? {
    println!("read # bytes: {}",numbytes_read);
    if numbytes_read==0 { break; }
  };
  Ok(())
}


// Passing File to function...
pub fn zipread( mut rd:&File )  { let x = rd.read(&[buf])?; println!("{}",x); }
// .. or... pass any struct with read trait
pub fn zipread( mut rd:impl Read )  { let x = rd.read(&[buf])?; println!("{}",x); }

// File position
// note this here is a u64 so y'all with a exabyte of data gone hafta finda workaroun'
// also for some reason f has to be mutable if you do this.
use std::io::Seek;
println!("{:?}",f.stream_position());

// there is no 'close', files close automatically at the end of scope ( "}" )
```

Filesystem

```
use std::fs;
match std::fs::copy( "file.txt", "newfile.txt") {
    Ok => println!("copy successfull"),
    Err(why) => println!("copy failed"),
}
std::fs::remove_file("newfile.txt").unwrap(); // panic if failure
```

System, arguments to main(), environment variables

```
std::env::args().for_each(|x| print!("{} ",x)); // main arguments as iterator, print each one
for arg in std::env::args().collect::<Vec<String>>() { print!("{} ",arg); }; // same, as vector
if std::env::args().any(|x| x=="--help") {help()};            // if called with --help, run help()
let progname = std::env::args().nth(0)  // first argument. but this wont compile! its an Option()
let progname = std::env::args().nth(0).unwrap_or("yr system is very broken".to_string());
// on most systems, first argument = program name. but args is not guaranteed to exist, its an iterator
// that could be empty. so we can use unwrap_or() to deal with the Option if no arguments are there
```

clap - command line argument parser

```
# if you want your program 'mycompiler' to have args like this:
mycompiler input.txt -o binary.exe -optimize -I./include -I/usr/include
# install clap crate:
cargo add clap --features derive   # bash command to add clap dependency



mycompiler.rs:
extern crate clap;
// clap will parse the struct and automatically handle errors,
// detect the argument type based on the variable (Vec,Option,bool)
// with Option for optional args, no Option for Requred args, Vec for
// multiple args, bool for flag style args, etc. clap will also
// automatically provide --verison and --help using /// comments
#[derive(clap::Parser, Debug)]
pub struct Args {
    /// inputfile - filename to compile
    inputfile: String,

    /// outfile - filename for binary output
    #[arg(short = 'o', long)]
    outfile: Option(String),

    /// paths to search for header files
    #[arg(short = 'I', long)]
    include: Vec<String>,

    /// use the optimizer during compilation
    #[arg(short='O', long)]
    optimize: bool
}

fn main() {
    let clapargs = Args::parse_from(std::env::args.clone());
    println!("{:?}",clapargs);
    let mut output = "a.out";
    if let Some(outname) = clapargs.outfile.as_deref() { output = outname};
    if let Some(input) = clapargs.inputfile.as_deref() {
    compile(input,output,clapargs.optimize);
    }
}
```

Environment variables

```
std::env::var("IGNORE_CASE")  // environment variable IGNORE_CASE, returns Result or Err
if let Ok(v) = std::env::var("SHLVL") {println!("{:?}",v);} // print SHLVL if it's there, otherwise ignore
let sl = match std::env::var("SHLVL") { Ok(v)=>v,Err(e)=>"not set".to_string() }; // set s1 to SHLVL or "not set" if unset
println!("{:?}",std::env::vars().collect::<Vec<(String,String)>>()); // print all environment variables
for v in std::env::vars() {println!("{:?}",v);}  // print env vars line by line
```

Reflection

For enums:

```
use strum::IntoEnumIterator;
#[derive(strum_macros::EnumIter)]
enum Attachment { Avoidant, Anxious, Secure };
Attachment::iter().for_each(|d|print!("{d:?},")); // Avoidant, Anxious, Secure
```

For others: todo

Traits

todo. Traits are like 'interfaces' in other languages. You can see examples above where a Struct will Derive the Debug trait so that gives it certain abilities. Traits are used extensively when dealing with iterators, see below.

Iterators, functional style programming

```
let v = vec![3,4,5];
let it = v.iter(); // iterator as object
println!("{:?} {:?} {:?}",it.next(),it.next(),it.next()); // consumed
println!("{:?}",it.next()); // None

let v = vec![3];
let mut i = v.iter().peekable();
println!("{:?}",i.peek());  // Some(3)
println!("{:?}",i.peek());  // Some(3)
println!("{:?}",i.next());  // 3
println!("{:?}",i.next());  // None

let mut i = v.iter();
print!("{:?}",i.collect::<Vec<_>>(); // iterator back to vector

// basic iteration and sorting
for i in v.iter() {print!("{i} ");} // 3 4 5
vec![3,4,5].iter().for_each(|x| print!("{x} ")); // 3 4 5
let biggest = v.iter().max();        // 5
let hasle2 = v.iter().any(|x| x<=4); // true if any element is less than or equal to 2
let biggest = v[0..1].iter().max();  // will return 4, not 5, b/c we took a slice of the vector
for i in vec![3,4,5].iter().take(2) {print("{i}");} // 3, 4
for (i,n) in vec![3,4,5].iter().enumerate() {print!("{i}:{n} ");} // 0:3 1:4 2:5
vec![3,4,5,3].iter().find(|&&x| x == 3) // Some(&3), first three
vec![3,4,5].iter().position(|&&x| x == 5) // 2 // kind of like indexOf() in other langs

// combine multiple iterators
(3..=5).chain(9..=11).for_each(|i|print!("{i:?} ")); // 3 4 5 9 10 11
(3..=5).zip(9..=11).for_each(|i|print!("{i:?} ")); // (3, 9) (4, 10) (5, 11)

// skipping, removing or modifying items
for i in v.iter().step_by(2) {print!("{i} ");} // 3 5 vec![
for i in vec![3,4,5].iter().skip(1) {print("{i}");} // 4, 5
print!("{:?}",vec![3,4,5].into_iter().map(|x| 2 * x).collect::<Vec<u8>>()); // 6 8 10
for i in vec![3,4,5].iter().filter(|x| x<=4) {print!("{i}");} // 3 4
for i in vec![Some(3),None,Some(4)].iter().fuse() {print!("{i}");} // Some(3), None
for i in vec![4,5,3,3].iter().skip_while(|x| **x>=4) {print!("{i},");} // 3,3
for i in vec![3,4,5,3].iter().skip_while(|x| **x>=4) {print!("{i},");} // 3,4,5,3
for i in vec![4,5,3,3,9,6].iter().take_while(|x| **x>=4) {print!("{i},");} // 4,5
for i in vec![3,4,5,3].iter().take_while(|x| **x>=4) {print!("{i},");} //   (nothing)
let v =[3,4,5]; for i in v.iter().cycle().skip(2); // 5,3,4 - cycle allows wraparound

// mathematical operations
```

```
for i in vec![3,4,5].iter().scan(0,|a,&x| {*a=*a+x;Some(*a)}) {print!("{i}")} // 3,7,12
print!("{}",vec![3,4,5].iter().fold(0, |a, x| a + x)); // 12
vec![3,4,5].iter().sum() // 3+4+5, 12
vec![3,4,5].iter().product() // 12*5, 60
(1..4).reduce(|x,y| x*y).unwrap_or(0); // 6   // 6 is 1*2*3
println!("{}",vec![1.,2.,3.].iter().cloned().fold(std::f64::MIN, f64::max)); // max, 3
println!("{}",vec![1.,2.,3.].iter().cloned().fold(std::f64::MAX, f64::min)); // min, 1
print!("{:?}",vec![['a','b','c'], ['d','e','f']].iter().flatten().collect::<String>() ); // "abcdef"
print!("{:?}",vec![vec![3,4],vec![5,1]].iter().flatten().collect::<Vec<_>>()); // 3,4,5,1
let (a,b): (Vec<_>, Vec<_>) = vec![3,4,5].into_iter().partition(|x| x>&4);
println!("{:?} {:?}",a,b); // [5] [3,4]
vec![3,4,5].iter().all(|x| x>2) // true
vec![3,4,5].iter().all(|x| x>4) // false
vec![3,4,5].iter().any(|x| x<4) // true
vec![3,4,5].iter().any(|x| x<2) // false
// comparisons: cmp, le, eq, ne, lt, etc
print!("{}",vec![3,4,5].iter().eq(vec![1,3,4,5].iter().skip(1))); // true

// misc
cloned() // clones each element
unzip() // backwards of zip()
rev() // reverse iterator
rposition() // combine reverse and position
max_by_key() // max using single func on each item
max_by()    // max using compare closure |x,y| f(x,y)
v.min_by(|a,b| a.x.partial_cmp(&b.x).unwrap_or(std::cmp::Ordering::Equal)); //min val,float
find_map() // combine find and map
flat_map() // combine flatten and map
filter_map() // combine filter and map

into_iter() // can help you collect
let (v,u)=(vec![1,2,3],vec![4,5,6]);
print!("{:?}", v.into_iter().zip(u.into_iter()).collect<Vec<(i8,i8)>>()); // E0277
// value of type `Vec<(i8, i8)>` cannot be built from `std::iter::Iterator<Item=(&{integer}...
print!("{:?}", v.into_iter().zip(u.into_iter()).collect<Vec<(i8,i8)>>()); // [(1, 4), (2, 5), (3, 6)]
```

Itertools library: more adapters

```
use itertools::Itertools;
// these are periodically integrated into Rust core, and/or may change

// sort related
vec![13,1,12].into_iter().sorted(); // [1,12,13]
vec![(3,13),(4,1),(5,12)].into_iter().sorted_by(|x,y| Ord::cmp(&x.1,&y.1)); // [(4,1),(5,12),(3,13)]
"Mississippi".chars().dedup().collect::<String>(); // Misisipi
"agcatcagcta".chars().unique().collect::<String>(); // agct

// mingling single items into others
"四四".chars().join("十")); // 四十四
(1..4).interleave(6..9).collect_vec(); // 1,6,2,7,3,8
" 十是十 ".chars().intersperse('四').collect::<String>(); // "四十四是四十四"
"愛".chars().pad_using(4,|_| '.').collect::<String>(); // "愛..."

// multiple iterators
"az".chars().merge("lm".chars()).collect::<String>(); // "almz"
"za".chars().merge_by("ml".chars(),|x,y| x>y ).collect::<String>(); // "zmla"
vec!["az".chars(),"lm".chars()].into_iter().kmerge().collect::<String>(); //"almz"
for c in &vec![1,2,3,4,5].into_iter().chunks(2) {  // chunk - split into new iterator groups of 2
  for x in c { print!("{:?}",x); } print!(","); } // each group is 2 items. output: 12,34,5,
```

```
    // mathematical operations
    "ab".chars().cartesian_product("cd".chars()).collect_vec() ; // [(a,b),(a,d),(b,c),(b,d)]
    (1..=3).combinations(2).collect_vec(); // [[1,2], [1,3], [2,3]]
    "Go raibh maith agat".chars().positions(|c| c=='a').collect_vec();// [4, 10, 15, 17]
    ("四十四是四十四".chars().all_equal(), "謝謝".chars().all_equal()); // (false, true)
    [1,5,10].iter().minmax().into_option().unwrap() // (1,10) // faster than min() then max()
    (0..3).zip_eq("abc".chars()).collect_vec(); // [(0,'a'), (1,'b'), (2,'c')]

    // modifying and removing items
    "bananas".chars().coalesce(|x,y| if x=='a'{Ok('z')}else{Err((x,y))}).collect::<String>();
    // coalesce will replace a pair of items with a new single item, if the item matches
    vec![1,2,3].into_iter().update(|n| *n *= *n).collect_vec(); // [1,4,9]
    let mut s=['.';3];s.iter_mut().set_from("abcdefgh".chars()); s.iter().collect::<String>(); // "abc"
```

Implementing your own iterator for your own struct

If you want to be able to use all the cool adapters like filter(), map(), fold(), etc, on your own cusom data structure you will need to implement the Iterator trait for your struct. Basically you need to create another struct called a "XIterator" that implements a 'next()' method. Then you create a method in your own struct called 'iter()' that returns a brand new XIterator struct.

Here is a super simple example, all it does is iterate over a data inside of custom data struct Mine. The data is stored in a vector so it's pretty straightforward to access it.

```
#[derive(Debug)]
struct Mine{
    v:Vec<u8>
}

impl Mine{
    fn iter(self:&Mine) -> MineIterator {
        MineIterator {
            m:&self, count:0,
        }
    }
}

struct MineIterator<'a> {
    m: &'a Mine,
    count: usize,
}

impl<'a> Iterator for MineIterator<'a> {
    type Item = &'a u8;
    fn next(&mut self) -> Option<Self::Item> {
        self.count+=1;
        if self.count<=self.m.v.len() {
        Some(&self.m.v[self.count-1])
    } else {
        None
    }
    }
}

fn main() {
    let m=Mine{v:vec![3,4,5]};
    m.iter().for_each(|i| print!("{:?} ",i));println!(""); // 3 4 5
    m.iter().filter(|x|x>3).fold(0,|a,x| a+x); // 9
```

```
}
```

If you want to create a Mutable Iterator, you will (as of writing, 2018) have to use an unsafe code with a raw pointer. This is alot like what the standard library does. If you are not extremely careful, your program may exhibit bizarre behavior and have security bugs, which defeats the purpose of using Rust in the first place.

- https://gitlab.com/Boiethios/blog/snippets/1742789
- https://doc.rust-lang.org/book/ch19-01-unsafe-rust.html#dereferencing-a-raw-pointer

```
impl Mine{
    fn iter_mut(self:&mut Mine) -> MineIterMut {
            MineIterMut {
            m:self, count:0,//_marker:std::marker::PhantomData,
        }
    }
}

pub struct MineIterMut<'a> {
    m: &'a mut Mine,
    count: usize,
}

impl<'a> Iterator for MineIterMut<'a> {
    type Item = &'a mut u8;
    fn next(&mut self) -> Option<&'a mut u8> {
            if self.count == self.m.v.len() {
                None
            } else {
                let ptr: *mut u8 = &mut self.m.v[self.count];
                self.count += 1;
        unsafe{ Some(&mut *ptr) }
            }
    }
}

fn main() {
    let mut m=Mine{v:vec![3,4,5]};
    m.iter().for_each(|i| print!("{:?} ",i));println!(""); // 3 4 5
    m.iter_mut().for_each(|i| *i += 1);
    m.iter().for_each(|i| print!("{:?} ",i));println!(""); // 4 5 6
}
```

Math

arithmetic

```
let x=250u8;           // addition can crash, max value of u8 is 255
println!("{}",x+10);   // crashes, attempt to add with overflow
println!("{}",x.wrapping_add(10));   // wraps around, result: 5
println!("{}",x.saturating_add(10)); // stays at max, result: 255
println!("{}",x.wrapping_sub(u8::MAX)); // wraparound subtraction
std::f32::MIN;         // minimum binary floating point 32 bit value
```

```
std::i32::MAX;         // maximum 32 bit integer value

let b = -5;
let c = b.abs();       // error, abs not defined on generic integer
let b = -5i32;
let c = b.abs();       // ok, abs is defined on i32, 32 bit integer
42.25f32.round();      // can also call functions on float literals
9.0f32.sqrt();         // square root approximation
9.sqrt()               // error, sqrt only for floats
let x = 3/4;           // 0
let y = 3.0/4;         // error, no implementation for `{float} / {integer}`
let z = 3.0/4.0;       // 0.75
```

Exponentials, exponentiation, power, raising to a power

```
let p1 = 2.pow(32) //err    // error[E0689]: can't call method `pow` on ambiguous numeric type `{integer}`
let p2 = 2_u8.pow(32) //err // thread 'main' panicked at 'attempt to multiply with overflow'
let p3 = 2_u8.checked_pow(32); // v becomes "None".
let p4 = match 2_u8.checked_pow(32) { Ok(n)=>n,None=>0 } // match a checked_pow b/c it returns Option
let p5 = 2u8.checked_pow(k).unwrap_or(0); // 0 if overflow, simpler than match
let p6 = 2_u64.pow(32)                   // ok
```

data conversion

```
let x: u16 = 42;           // integer conversion. start with u16
let y: u8 = x as u8;       // cast to unsigned 8 bit integer
let z: i32 = x as i32;     // cast to signed 32 bit integer
let bez = z.to_le_bytes(); // get individual 8 bit bytes in the 32 bit int
let bbz = z.to_be_bytes(); // same, but in big endian order
println!("{:?},{:?},{:?},{:?}",bez[0],bez[1],bez[2],bez[3]); // 42,0,0,0
println!("{:?},{:?},{:?},{:?}",bbz[0],bbz[1],bbz[2],bbz[3]); // 0,0,0,42

let a = u32::from('A')             // get integer value of char. all chars are 4 bytes.
let a = char::from_u32(68)         // get char value of an integer (utf8/ascii)
let a = char::from_digit(4,10)     // get char '4' from integer 4, base 10

let s = [0,1,2,3];                 // u8 to u32, start with array of four u8
let ls = u32::from_le_bytes(s);    // convert to u32, little endian
let bs = u32::from_be_bytes(s);    // convert. to u32, big endian
println!("{:?} {:?}",ls,bs);       // 50462976 66051
let s2 = vec![0,0,0,1,2,3,4,5,6,7]; // vector of ten u8
let ars2 = [s2[2],s2[3],s2[4],s2[5]]; // array of four u8
let be = u32::from_be_bytes(ars2);    // create u32 from 3rd-6th bytes of ars2
println!("{:?}",be);               // 66051

// converting vectors of integers
let mut v1 = vec![0u8;2];               // 2 u8, each with '0'
let v2 = vec![0xffeeaa00u32,0xff0000dd]; // two u32 in a vector
v2.extend(v2);  // error the trait bound `Vec<u8>: Extend<u32>` is not satisfied
for d in v2 { v1.extend(d.to_le_bytes()) }; // convert each u32 by itself
println!("{:?}",v1) // [0, 0, 0, 170, 238, 255, 221, 0, 0, 255]

// integer conversion using byteorder crate
extern crate byteorder; // modify your Cargo.toml to add byteorder crate. then:
use byteorder::{BigEndian, ReadBytesExt, NativeEndian, ByteOrder, LittleEndian, WriteBytesExt};
let arr = [0,1,2,3];
let s = NativeEndian::read_u32(&arr[0..4]);   // array of four bytes into u32.
let mut v = vec![0u8];
```

```
s.write_u8( v );

let vx = vec![0u8,1,2,0xff,4,5,6,0xff];     // vector of u8
//let mut vy = vec![0u32];                  // vector of u32
//LittleEndian::read_u32_into(&vx,&mut vy); // panic, vy cant hold vx
let mut vy = vec![0u32;2];                  // vector of u32
LittleEndian::read_u32_into(&vx,&mut vy);   // convert u8s to u32s
println!("{:x?}",vy);                       // [ff020100, ff060504]
let mut vx2 = vec![0u8;8];                  // new vector of u8
LittleEndian::write_u32_into(&vy,&mut vx2); // convert back to u8s
println!("{:02x?}",vx);                     // [00, 01, 02, ff, 04, 05, 06, ff]

// converting vectors of integers using unsafe mem::transmute
let v = vec![0u8;80]; let i = 0;
let n:i32 = {unsafe { std::mem::transmute::<&[u8],&[i32]>(&v[i..i+4])}}[0];
let x:f32 = {unsafe { std::mem::transmute::<&[u8],&[f32]>(&v[i..i+4])}}[0];
let mut block = vec![0u8;64];  // convert entire block at once
let mut X = unsafe { mem::transmute::<&mut [u8], &mut [u32]>(&mut block) };
#[cfg(target_endian = "big")]   // deal with endian issues if needed
for j in 0..16 { X[j] = X[j].swap_bytes(); }




let s = format!("{:e}",0.0f32);    // convert float32 to base-10 decimal string (scientific format)
let n = s.parse::<f32>().unwrap(); // parse float from string, panic/crash if theres an error
let mut n = 0.0f32;                // parse float from string, without panic/crashing
match s.parse::<f32>() {
    Err(e)=>println!("bad parse of {}, because {}",s,e),
    Ok(x)=>n=x
}
let a = "537629.886026485"                       // base-10 decimal string to binary float point
print!("{:.50}",a.to_string().parse::<f32>().unwrap();// 537629.87500000000000000000000000000000
let b = 537629.886026485;    print!("{:.50}",b);      // 537629.88602648500818759202957153320312 5
let c = 537629.886026485f32; print!("{:.50}",c);      // 537629.87500000000000000000000000000000
let d = 537629.886026485f64; print!("{:.50}",d);      // 537629.88602648500818759202957153320312 5

let ch='e';
ch.is_digit(16) // true, 'e' is a numerical digit in base 16
ch.is_digit(10) // false, 'e' is not a numerical digit in base 10, only 0-9 are.
let y:u32 = ch.to_digit(16).unwrap_or(0);  // convert 'e' into 15, since base=16. if fail, make y 0
let n = 'x'.is_alphabetic(); // detect whether letter
println!("𝄞 is hex 0x{:06x}, or decimal {}",'𝄞' as u32,'𝄞' as u32); // unicode codepoint
println!("a is hex 0x{:06x}, or decimal {}",'a' as u32,'a' as u32); // for <128 this is the ascii code

let m = 0b0001; // binary literal
let m = 0o0007; // octal literal
let m = 0x000f; // hexadecimal literal
let (a,b,c) = (0b00_01,0o00_07,0x00_0f); // literals with underscores for ease of reading

println!("{:x}",0x12345678u32.swap_bytes());  // 0x78563412 32-bit byteswap
```

string conversion

```
use std::i64;
let z = i64::from_str_radix("0x1f".trim_start_matches("0x"), 16).unwrap(); // hex string to integer
```

```
let q = i64::from_str_radix("0b10001".trim_start_matches("0b"), 2).unwrap(); // binary string to integer

println!("{:?}","abc".as_bytes()); // &[u8] slice, [97, 98, 99] ( utf8 string into bytes )
println!("{:?}","abc".as_bytes().to_vec()); // Vec<u8> [97, 98, 99] string into slice into Vector of bytes
println!("{:?}","ᏣᎳᎩ".as_bytes()); // [225, 143, 163, 225, 142, 179, 225, 142, 169] ( Cherokee utf8 )
let s = b"\x61\x62\x63"; // b precedes sequence of bytes, hexadecimal, pseudo-string form
println!("{}",std::str::from_utf8( s ).unwrap()); // abc // decodes utf8 to string, crash if invalid utf8

println!("{:?}", "सूर्य नमस्कार्कार".as_bytes()); // [224, 164, 184,...]
let s = [224, 164, 184, 224, 165, 130, 224, 164, 176, 224, 165,
         141, 224, 164, 175, 32, 224, 164, 168, 224, 164, 174, 224,
     164, 184, 224, 165, 141, 224, 164, 149, 224, 164, 190, 224, 164, 176];
println!("{}",std::str::from_utf8( &s ).unwrap()); // सूर्य नमस्कार
println!("{:?}",std::str::from_utf8( &s ).unwrap()); // "स\u{942}र\u{94d}य नमस\u{94d}कार" different decoding



let s = "hello" ;                   // s = &str
let s = "hello".to_string();        // s = String
let m = s.replace("hello","new");   // m = "new"
let y = s.to_uppercase();           // y = "NEW"

// str.as_bytes() converts to slice, which implements Read
let s = "Reedeth Senek, and redeth eek Boece";
let s2= "Ther shul ye seen expres that it no drede is"
let buf = &mut vec![0u8; 64];
s.as_bytes().read(buf).unwrap();
s2.as_bytes().read(&buf[30]).unwrap();
```

Regular expressions typically use an external crate. Syntax for regex: https://docs.rs/regex/1.1.2/regex/index.html#syntax

In cargo.toml, [dependencies] regex = "1.1.0"

In main.rs:

```
use regex::Regex;
let text = r###"The country is named France, I think the biggest city is Paris;
the Boulangerie are beautiful on la rue du Cherche-Midi"###;             // multiline string

let re = Regex::new(r"country is named (?P<country>.*?),.*?biggest city is (?P<biggestcity>.*?);").unwrap();
let caps = re.captures(text).unwrap();
println!("{}, {}",&caps["biggestcity"],&caps["country"]);                 // Paris, France

let re = Regex::new(r"(?ms)city is (?P<citname>.*?).$.*?beautiful on (?P<streetname>.*?)$").unwrap();
let caps = re.captures(text).unwrap();
println!("{}, {}",&caps["streetname"],&caps["citname"]);              // la Rue de Cherche Midi, Paris
```

comparison and sorting

```
use std::cmp                    // max/min of values
let a = cmp::max(5,3);          // maximum value of 2 integers
let b = cmp::max(5.0,3.0);      // build error, floats cant compare b/c of NaN,Inf
let v = vec![1,2,3,4,5];        // list, to find max of
let m = v.iter.max().unwrap();      // max of numbers in a list, crash on error
match v.iter().max() {          // max of numbers in a list, don't crash
    Some(n)=>println!("max {}",n),  // do stuff with max value n
    None=>println!("vector was empty")
}
```

```
let m = v.iter.max().unwrap_or(0);      // max of numbers in a list, or 0 if list empty

v = vec![1,3,2];
v.sort();                                   // sort integers
v = vec![1.0,3.0,2.0];                      // sort floats
v.sort();                                   // error, float's NaN can't be compared
v.sort_by(|a, b| b.cmp(a));                 // sort integers using closure
v.sort_by(|a, b| a.partial_cmp(b).unwrap());   // sort floating point using closure
v.min_by(|a,b| a.x.partial_cmp(&b.x).unwrap_or(std::cmp::Ordering::Equal)); // minimum value
println!("{}",vec![1.,2.,3.].iter().cloned().fold(std::f64::MIN, f64::max)); // 3
println!("{}",vec![1.,2.,3.].iter().cloned().fold(std::f64::MAX, f64::min)); // 1

struct Wheel{ r:i8, s:i8};                  // sort a vector that holds a struct
let mut v = vec![Wheel{r:1,s:2},Wheel{r:3,s:2},Wheel{r:-2,s:2}];
v.sort_by(|a, b| a.r.cmp(&b.r));            // sort using closure, based on value of field 'r'
v.sort_by(|a, b| a.r.cmp(&(&b.r.abs())));   // sort using closure, based on abs value of r

fn compare_s( a:&Wheel, b:&Wheel ) -> std::cmp::Ordering {      // sort using a function
     a.s.partial_cmp(&b.s).unwrap_or(std::cmp::Ordering::Equal)
}
v.sort_by( compare_s );
```

Pseudo randoms

Pseudo Random number generators, aka PRNGs

```
extern crate rand;   /// add rand to dependencies in Cargo.toml
use rand::prelude::*;
let x = rand::random(); // boolean
let y = rand::random::<int>(); // integer
let x = rand::thread_rng().gen_range(0, 100); // between
let mut rng = rand::thread_rng();
let z: f64 = rng.gen(); // float 0...1
let mut nums: Vec<i32> = (1..100).collect();
nums.shuffle(&mut rng);



// alternative, without needing external crate, based on codeproject.com by Dr John D Cook
// https://www.codeproject.com/Articles/25172/Simple-Random-Number-Generation
// License: BSD, see https://opensource.org/licenses/bsd-license.php
use std::time::{SystemTime, UNIX_EPOCH};
let mut m_z = SystemTime::now().duration_since(UNIX_EPOCH).expect("Time reversed").as_millis();
let mut m_w = m_z.wrapping_add( 1 );
let mut prng = || {  m_z = 36969 * (m_z & 65535) + (m_z >> 16);
                     m_w = 18000 * (m_w & 65535) + (m_w >> 16);
                     m_z.rotate_left( 16 ) + m_w };
let buf = (0..1000).map(|_| prng() as u8).collect::<Vec<u8>>();
// creates 1000 pseudo random bytes in buf, using Closure named prng
```

Hashing

```
use std::collections::hash_map::DefaultHasher;
use std::hash::{Hash, Hasher};
let mut hasher = DefaultHasher::new(); // hashing, via a Hasher object, which holds state
let x = 1729u32;
let y = 137u32;
hasher.write_u32( x );                 // input x to hash func, store output in hasher
```

```
hasher.write_u32( y );                // input y, combine w existing state, store in hasher
println!("{:x}", hasher.finish());    // .finish() function Hasher gives current state
345.hash(&mut hasher);                // update hasher's state using '.hash()' trait
println!("{:x}", hasher.finish());    // .finish() does not 'reset' hasher objects
```

Date and Time

```
use std::time::{Instant};
let t = Instant::now();
// do something for 5.3 seonds
println!("{}",t.elapsed().as_secs());          // 5, seconds, rounded
println!("{}",t.elapsed().subsec_millis());    // 300. remainder in milliseconds

use chrono::prelude::*;
println!("{:?} ", Utc::now().to_rfc2822());
println!("{:?} ", Utc::now().format("%Y-%m-%d %H:%M:%S").to_string());
println!("{:?} ", Local::now().to_rfc2822());
println!("{:?} ",DateTime::parse_from_str("2014-11-28 21:00:09 +09:00", "%Y-%m-%d %H:%M:%S %z"));
println!("{:?} ",Utc.with_ymd_and_hms(2014, 11, 28, 12, 0, 9).unwrap());
println!("{:?} ",Utc.timestamp(1500000000, 0)); // Epoch time seconds, aka mtime on Unix files.
"Tue, 17 Jan 2023 03:12:07 +0000"
"2023-01-17 03:12:07"
"Mon, 16 Jan 2023 21:12:07 -0600"
Ok(2014-11-28T21:00:09+09:00)
2014-11-28T12:00:09Z
2017-07-14T02:40:00Z
```

Annotations

Aka hashtags aka warning thingies. These statements typically affect compilation and begin with a hashtag, and square brackets. They are sort of like a mix of #pragma and #ifdef from C.

```
#![allow(dead_code)]             // stop compiler from printing warnings on unused funcs/vars,
#![allow(unused_variables)]      // this is good when you are working on a library
//  = note: #[warn(unused_assignments)] on by default
#![allow(unused_assignments)]    // you can take most warnings, and disable by changing 'warn' to 'allow'

// by removing ! and placing on line before a fn, you can disbale warning only for the function
#[allow(unused_assignments)]
fn zoogle_poof( n:u8 )->u8 { let a = 0; 5+n };

let mut a = 0x00ffee22;          // modify numbers for big endian machines.
#[cfg(target_endian = "big")]    // target = machine the code will be run on
a = a.swap_bytes();              // the line (or block) immediately following the # gets affected.
```

Linked Lists

Textbook C/Java-style implementations of linked lists often involve ownership that is not allowed by the Rust borrow checker. However it can be accomplished. There is builting "LinkedList" type in std::collections, also some resources from A. Beinges :

https://cglab.ca/~abeinges/blah/too-many-lists/book/

You can also implement a linked list as a Vector of structs, using integer indexes into the vector instead of pointers.

FFI, Calling C functions, porting C code

foreign function interface, aka calling code from other languages.

layout

```
    src/lib.rs
    src/ccode.c
    build.rs
    Cargo.toml
```

Cargo.toml

```
[package]
name = "duh"
version = "0.1.0"
authors = ["akhmatova"]
build = "build.rs"
[build-dependencies]
cc = "1.0"
libc = "0.2.0"
```

build.rs

```
extern crate cc;

fn main() {
    cc::Build::new().file("src/ccode.c").compile("ccode");
}
```

src/ccode.c

```
#include <stdio.h>
int quadrance( int x1, int y1, int x2, int y2) {
    return (x1-x2)*(x1-x2)+(y1-y2)*(y1-y2);
}
char * blerg( *char txt ) {
    printf("This is C. Here is text from Rust: %s", txt);
    int fd = open("/tmp/blerg",0);
    return "blerg";
}
void printerrs() {
        int code = errno;
        printf("C: errno: %i strerror: %s\n",code,strerror(code));
}
```

src/main.rs

```
use std::os::raw::{c_int,c_char,c_void};
use std::ffi::{CString,CStr};
extern "C" {
    fn quadrance(x1: c_int, y1: c_int, x2: c_int, y2: c_int) -> c_int;
    fn blerg(v: *const char)->*const c_char;
    fn printerrs()->c_void;
```

```
}
fn main() {
    unsafe {
        println!("4^2+3^2={:?}", quadrance(0, 0, 4, 3));
    let tmp = CString::new("blarg").unwrap(); // CString must be assigned
    let msg = blerg(tmp.as_ptr()); // so as_ptr() will have something to point at
    println!("This is Rust. Here's text from C: {:?}",CStr::from_ptr(msg));
    printerrs(); // was there an error?
    }
}
```

Run:

```
don@oysters:~/duh$ cargo run
   Compiling duh v0.1.0 (/home/don/duh)
    Finished dev [unoptimized + debuginfo] target(s) in 1.95s
     Running `target/debug/duh`
4^2+3^2=25
This is C. Here is text from Rust: blarg
This is Rust. Here's text from C: blerg
```

See also: https://s3.amazonaws.com/temp.michaelfbryan.com/index.html

c++ - https://hsivonen.fi/modern-cpp-in-rust/

https://doc.rust-lang.org/nomicon/ffi.html

Michael Bryan's unofficial guide https://s3.amazonaws.com/temp.michaelfbryan.com/index.html

Using char** C functions: https://stackoverflow.com/questions/42717473/passing-vecstring-from-rust-to-char-in-c

Note that much C code does not initialize structs or memory, which may be forgotten when one is used to Rust http://www.ex-parrot.com/~chris/random/initialise.html

Structs and bindgen auto header conversion https://medium.com/dwelo-r-d/using-c-libraries-in-rust-13961948c72a

Setting CFLAGS examples:

modify build.rs:

```
extern crate cc;

fn main() {
    std::env::set_var("CFLAGS","-w");  // turn off all warnings
    std::env::set_var("CFLAGS","-v");  // run compiler in verbose mode
    std::env::set_var("CFLAGS","-v -O2");  // optimize level 2 + verbose mode
    cc::Build::new().file("src/ccode.c").compile("ccode");
}
```

Unions:

In Rust, Enums are typically preferred over union, as a traditional C-style union is only available by using unsafe{}. But unions can help talking to / porting C code:

```
#[repr(C)]
union Borgle { dorgle: u32, porgle: u8[4] }
```

```
let mut a=Borgle{dorgle:1234};
unsafe{ a.dorgle = 0xa0ca0c };
```

Goto vs labels, loops, and breaks:

Rust does not have Goto, however sometimes a similar result can be achieved using labelled loops and breaks.

```
    'label1: loop {
        if zimblatz == 5 {
        break; // breaks labe1 loop
    }
        'label2: for frobnoz in 0..4 {
            if blorg==KUMQUAT {break;} // breaks label2 only
        while j<5 {
            if snog==7 {
            // goto end of label1 loop immediately
            break 'label1;
        }
        j+=1;
        }
        }
    } // end of 'label1 loop
    do_something();
```

Pointer arithmetic

```
use std::raw::c_uchar;
unsafe{
    let mut x = some_c_func() as *const c_uchar;
    x = x.offset(1);  // instead of x++
}
```

source code layout and modules

Here is an example source code organization layout for a simple library, which has a handful of modules, some examples, integration tests, benchmarking, and two executable binaries.

```
$ ls ..
./mycrate                    # main crate folder
./mycrate/Cargo.toml         # cargo file, lists dependencies etc
./mycrate/src/lib.rs             # only one library is allowed per crate
./mycrate/src/mod1.rs        # module 1. a library can use multiple modules
./mycrate/src/mod2.rs            # module 2 , a second module
./mycrate/src/bin/program1.rs    # source code for an executable
./mycrate/src/bin/program2.rs    # source code for another executable
./mycrate/tests                  # integration tests (as opposed to unit tests at end of every .rs file)
./mycrate/tests/integration_test.rs  # big test that tests big part of library
./mycrate/tests/test_data_file       # integration tests often use test files
./mycrate/examples               # easy to follow examples
./mycrate/examples/helloworld.rs  # simple example, "use mycrate::*"
./mycrate/examples/zoogbnoz.rs    # more complicated example
./mycrate/benches                # benchmarking code and files
./mycrate/benches/benchtest.rs    # program to run benchmarking speed tests
./mycrate/build.rs               # optional program to run before build
./mycrate/target          # binaries are generated under target
./mycrate/target/debug/program1   # debug build executable
```

```
./mycrate/target/release/program1 # release build executable (optimized for speed)
$ cargo build                     # this builds all files in crate
```

src/lib.rs: (our main library file which the world will use)

```
mod mod1;      // we must add each modules in lib.rs with 'mod' before we can use them within each other
use mod1::*;   // import symbols from mod1.rs
pub mod mod2;  // this module is public, anyone using this crate can access its functions too
use mod2::*;   // import symbols from mod2.rs
pub fn mycrate_init() {
  let x = do_stuff1(9);  // call function from mod1.rs
  mycrate_monster(x);
}
pub fn mycrate_monster(y:u8) -> Monster { // Monster - type from mod1.rs
  let mut a = Monster{ true,y };
  do_stuff2( &mut a );   // call function from mod2.rs
  a
}
pub fn mycrate_calcteeth()-> { 23 }
```

src/mod1.rs: (our 1st module that does stuff)

```
pub fn do_stuff1(x:u8)->u8 { x+5 } // public function, available to other modules
pub struct Monster1 {  // public struct, available to other modules
    pub is_happy:bool, // public struct member, available to others
    pub num_teeth:i8,
}
```

src/mod2.rs: (

```
use super::*; // use functions from lib.rs
use mod1::*;  // use functions/types from mod1.rs, like Monster struct
              // note: this only works if "mod mod1" line is in lib.rs
          // otherwise you get E0432 unresolved import , maybe a missing crate
pub fn do_stuff2( a: &mut Monster ) {
    a.is_happy = calc_happy();           // call our own private function
    a.num_teeth = mycrate_calcteeth();   // call a function from lib.rs
}
fn calc_happy() -> bool { // private function only available within mod2.rs
  match today { Tuesday => true ,_=>false } }
```

See also https://doc.rust-lang.org/book/ch07-02-modules-and-use-to-control-scope-and-privacy.html

ANSI colors

In C, ansi colors for terminal text output are typically done with the escape code in octal format, such as printf("\033]0;31m test red"); In Rust, you can use the unicode escape sequence, instead of 033 octal, we can use unicode with 001b hexadecimal:

```
fn main() {
    println!("\u{001b}[0;31m{}", "test red");
    println!("\u{001b}[0;32m{}", "test green");
    println!("\u{001b}[0;33m{}", "test orange");
    println!("\u{001b}[0;34m{}", "test blue");
    println!("\u{001b}[0;35m{}", "test magenta");
```

```
    println!("\u{001b}[0;36m{}", "test cyan");
    println!("\u{001b}[0;37m{}", "test white");
    println!("\u{001b}[0;91m{}", "test bright red");
    println!("\u{001b}[0;92m{}", "test bright green");
    println!("\u{001b}[0;93m{}", "test yellow");
    println!("\u{001b}[0;94m{}", "test bright blue");
    println!("\u{001b}[0;95m{}", "test bright magenta");
    println!("\u{001b}[0;96m{}", "test bright cyan");
    println!("\u{001b}[0;97m{}", "test bright white");
    println!("\u{001b}[0m{}", "restored to default");
}
```

There are also several crates that do this. Search the web for additional ANSI color features.

Thanks

Based on a8m's go-lang-cheat-sheet, https://github.com/a8m/go-lang-cheat-sheet, and

- rust-lang.org, Rust book, https://doc.rust-lang.org/book/second-edition
- rust-lang.org, Rust reference, https://doc.rust-lang.org
- rust-lang.org, Rust by example, https://doc.rust-lang.org/rust-by-example/
- rust playground, from integer32, https://play.integer32.com/
- Phil Opp builds an OS in rust, https://os.phil-opp.com/unit-testing/
- Rust Cookbook, Language Nursery https://rust-lang-nursery.github.io/rust-cookbook/algorithms/sorting.html
- Itertools docs https://docs.rs/itertools/*/itertools/trait.Itertools.html for more details
- carols10cent's js-rust cheatsheet, https://gist.github.com/carols10cents/65f5744b9099eb1c3a6f
- c0g https://stackoverflow.com/questions/29483365/what-is-the-syntax-for-a-multiline-string-literal
- codngame https://www.codingame.com/playgrounds/365/getting-started-with-rust/primitive-data-types
- Adam Leventhal post here, http://dtrace.org/blogs/ahl/2015/06/22/first-rust-program-pain/
- Shepmaster, https://stackoverflow.com/questions/33133882/fileopen-panics-when-file-doesnt-exist
- again, https://stackoverflow.com/questions/32381414/converting-a-hexadecimal-string-to-a-decimal-integer
- user4815162342, https://stackoverflow.com/questions/26836488/how-to-sort-a-vector-in-rust
- mbrubek, https://www.reddit.com/r/rust/comments/3fg0xr/how_do_i_find_the_max_value_in_a_vecf64/
- https://stackoverflow.com/questions/19671845/how-can-i-generate-a-random-number-within-a-range-in-rust
- Amir Shrestha https://amirkoblog.wordpress.com/2018/07/05/calling-native-c-code-from-rust/
- Julia Evans https://jvns.ca/blog/2016/01/18/calling-c-from-rust/
- huon https://stackoverflow.com/questions/23850486/how-do-i-convert-a-string-into-a-vector-of-bytes-in-rust
- EvilTak https://stackoverflow.com/questions/43176841/how-to-access-the-element-at-variable-index-of-a-tuple
- https://www.90daykorean.com/korean-proverbs-sayings/
- oli_obk https://stackoverflow.com/questions/30186037/how-can-i-read-a-single-line-from-stdin
- ogeon https://users.rust-lang.org/t/how-to-get-a-substring-of-a-string/1351
- A.R https://stackoverflow.com/questions/54472982/converting-a-vector-of-integers-to-and-from-bytes-in-rust
- dten https://stackoverflow.com/questions/25060583/what-is-the-preferred-way-to-byte-swap-values-in-rust
- How To Rust-doc https://brson.github.io/2012/04/14/how-to-rustdoc
- Pavel Strakhov https://stackoverflow.com/questions/40030551/how-to-decode-and-encode-a-float-in-rust
- Zargony https://stackoverflow.com/questions/19650265/is-there-a-faster-shorter-way-to-initialize-variables-in-a-rust-struct/19653453#19653453
- Wesley Wiser https://stackoverflow.com/questions/41510424/most-idiomatic-way-to-create-a-default-struct

- u/excaliburhissheath and u/connorcpu https://www.reddit.com/r/rust/comments/30k4k4/is_it_possible_to_modify_wrapped
- Simson https://stackoverflow.com/questions/54472982/how-to-convert-vector-of-integers-to-and-from-bytes
- Raul Jordan https://rauljordan.com/rust-concepts-i-wish-i-learned-earlier/
- Will Crichton https://www.youtube.com/watch?v=bnnacleqg6k
- Jimmy Hartzell https://www.thecodedmessage.com/posts/oop-2-polymorphism/
- Peter Hall https://stackoverflow.com/questions/63437935/in-rust-how-do-i-create-a-mutable-iterator