cargo new rust\_argument

cargo run

Parameter arguments.

//iterator, call next to get value until no value

let mut args= **std::env::arg()**.skip(1);

let key = **args.next()**.unwrap();

//unwrap() is a method returns a value if exists otherwise crash the program

* cargo r – foo

**Write**

Save key value in files.

**fs** module

let mut args = std::env::arg().skip(1)

let key = args.next().unwrap();

let value= args.next().unwrap();

let content = format!(“{}\t{}\n”, key, value)

(“Key.db”, content);

//Love to experiment by using key value direct

Rust has no classes, it has structure

let mut args = std::env::arg().skip(1)

let key = args.next().unwrap();

let value= args.next().unwrap();

let db = Database::new();

let content = format!(“{}\t{}\n”, key, value)

std::fs::write(“Key.db”, content);

**struct** Database {

inner: **std::collections::HashMap**<String, String>

}

//Added more functions tied to structure

**impl** Database{

**fn** new()-> Database{

//return hash map

Database {

Inner: std::collections::HashMap::new()

}

}

}

Now we want to read db file , parse file and store into hashmap

**impl** Database{

**fn** new()-> Result<Database, **std::io::Error**>{

//return hash map

//we are mutating hash map, mutating means add, deleting values from it

let **mut** hash\_map = **std::collections::HashMap**::**new**()

let contents = match **std::fs::read\_to\_string**(“kv.db”).unwrap(){

OK(c) => c,

Err(error) => return Err(error),

};

//Above alternative

**//let contents = std::fs::read\_to\_string(“kv.db”) ? ;**

//Now parse the content

//contents.lines(); gives iterator

for line in contents.**lines()** {

dbg!(line);

//let chunks = line.split(‘\t’); //gives iterator

let chunks: Vec<&str> = line.split(‘\t’).collect();

let key = chunks[0];

let value = chunks[1];

//This will break because key and value variables are part of content, not the owner

//hash\_map.insert(key.to\_owned(), value.to\_owned());

}

Ok(Database {

Inner: hash\_map

})

}

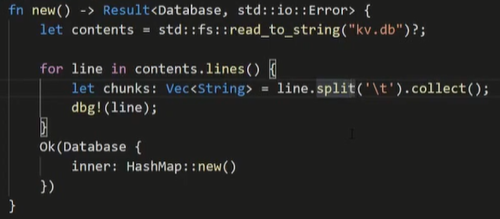
}

// As we know unwrap completely breaks the program, there is no way to handle the crash

//if we want to handle the crash, we should return the error type as Result<Database, Error>

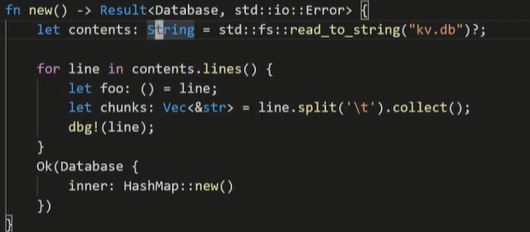
String > own string : heap allocated, freed after out of scope

&str > won’t get destroyed if out of scope. Borrowed value



Above will not work because contents variable is owner and line cannot be owner so it is just a reference.

Why do we this ?



* contents variable is an owner of all String.
* line variable is a simply pointer to part of contents variable.