Introduction to Unicode & i18n in Rust

Behnam Esfahbod
Software Engineer

Quora

Abstract

42nd Internationalization & Unicode Conference

September 2018

Santa Clara, CA, USA

The Rust Programming Language has native support for Unicode Characters' Unicode Scalar Values, to be exact. The language provides fast and compact string type with low-level control over memory consumption, while providing a high-level API and enforcing memory and data safety at compile time. The Rust Standard Library covers the basic Unicode functionalities, and third-party libraries – called Crates – are responsible for the rest. UNIC's Unicode and Internationalization Crates for Rust is a project to develop a collection of crates for Unicode and internationalization data and algorithm, and tools to build them, designed to have reusable modules and easy-to-use and efficient API.

In this talk we will cover the basics of Rust's API for characters and strings, and look under the hood of how they are implemented in the compiler and the standard library. Afterwards, we look at UNIC's design model, how it implements various features, and lessons learned from building sharable organic micro components.

The talk is suitable for anyone new to Unicode, or Unicode experts who like to learn about how things are done in the Rust world.

Abstract

42nd Internationalization & Unicode Conference

September 2018

Santa Clara, CA, USA

The Rust Programming Language has nativ[redacted]upport for Unicode Characters' Unicode Scal[redacted]alues, to be exact. T[redacted]guage provides fast and compact string ty[redacted]low-level contr[redacted]ory consumption, while providing a high[redacted] and enforc[redacted] and data safety at compile time. The R[redacted]dard Li[redacted] the basic Unicode functionalities, and thir[redacted]ries[redacted]tes – are responsible for the rest. UNIC's Unicode [redacted]Crates for Rust is a project to develop a collection of[redacted]and internationalization data and algorithm, and tools to[redacted]ned to have reusable modules and easy-to-use and efficien[redacted]

In this talk we will cover t[redacted]API for characters and strings, and look under the hoo[redacted]mented in the compiler and the standard library.[redacted]C's design model, how it implements various[redacted] from building sharable organic micro co[redacted]

The talk is suitable[redacted]ne new to Un[redacted]de, or Unicode experts who like to learn about how th[redacted]are done in the Rust world.

Looking for L10n in Rust?

Happening NOW on Track 3!

Fluent 1.0 — Next Generation Localization System from Mozilla

by Zibi Braniecki

Localization systems have been largely stagnant over the last 20 years. The last major innovation - ICU MessageFormat - has been designed before Unicode 3.0, targeting C++ and Java environments. Several attempts have been made since then to fit the API into modern programming environments with mixed results.

Fluent is a modern localization system designed over last 7 years by Mozilla. It builds on top of MessageFormat, ICU and CLDR, bringing integration with modern ICU features, bidirectionality, user friendly file format and bindings into modern programming environments like JavaScript, DOM, React, Rust, Python and others. The system comes with a full localization workflow cycle, command line tools and a CAT tool.

With the release of 1.0 we are ready to offer the new system to the wider community and propose it for standardization.

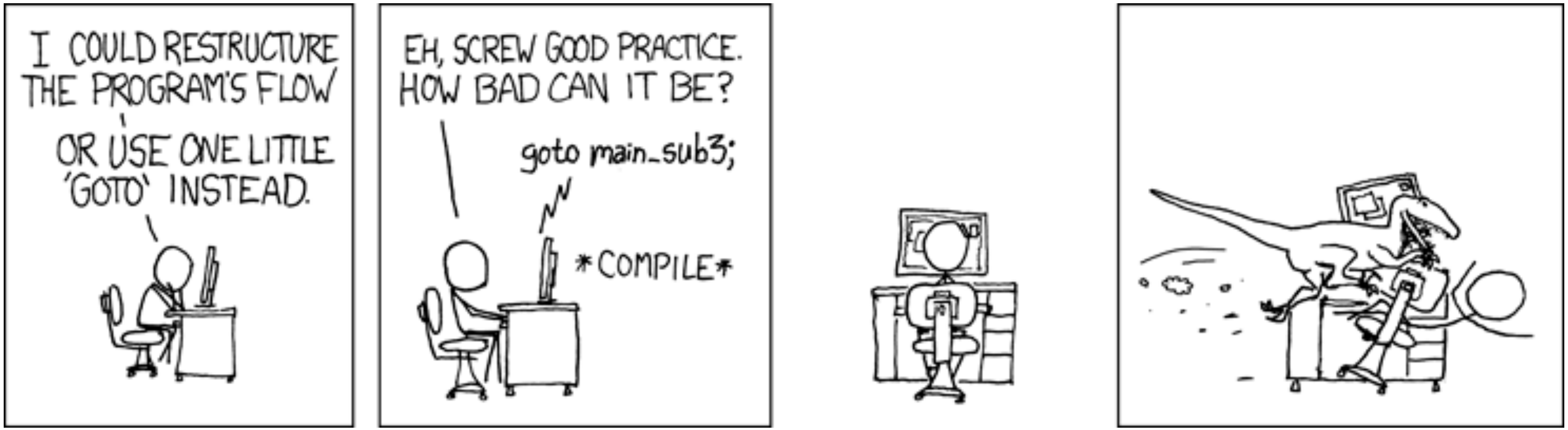
About me

- **Software Engineer @ Quora, Inc.**
- **Co-Chair of Arabic Layout Task Force @ W3C i18n Activity**
- **Virgule Typeworks**
- **Facebook, Inc.**
- **IRNIC Domain Registry**
- **Sharif FarsiWeb, Inc.**

This talk

- Quick Intro to Rust
- Characters & Strings
- It Gets Complicated!
- On Top of the Language

Quick Intro to Rust


I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE

XKCD | GOTO [CC BY-NC 2.5]

History

- **2006: The project started out of a personal project of Graydon Hoare**
  – OCaml compiler

- **2009: Mozilla began sponsoring**

- **2011: Self-hosting compiler, using LLVM as backend**

History

- **2006: The project started out of a personal project of Graydon Hoare**
  - OCaml compiler
- **2009: Mozilla began sponsoring**
- **2011: Self-hosting compiler, using LLVM as backend**
- **Pre-2015: Many design changes**
  - Drop garbage collection
  - Move memory allocation out of the compiler
- **2015: Rust 1.0, the first *stable* release**
- **2018: First major new edition, *Rust 2018***

Build System & Tooling

- **Cargo**
  - Package manager
  - Resolve dependencies
  - Compile
  - Build package and upload to crates.io

Build System & Tooling

- **Cargo**
  - Package manager
  - Resolve dependencies
  - Compile
  - Build package and upload to crates.io

- **Common tooling**
  - Rustup
  - Rustfmt
  - Clippy
  - Bindgen

Systems Language

- **Abstraction without overhead (ZCA)**
  - & without hidden costs

Systems Language

- **Abstraction without overhead (ZCA)**
  - & without hidden costs

- **Compile to machine code**
  - & runs on microprocessors (no OS/malloc)

Systems Language

- **Abstraction without overhead (ZCA)**
  - & without hidden costs
- **Compile to machine code**
  - & runs on microprocessors (no OS/malloc)
- **Full control of memory usage**
  - Even where there's no memory allocation

Systems Language

- **Abstraction without overhead (ZCA)**
  - & without hidden costs
- **Compile to machine code**
  - & runs on microprocessors (no OS/malloc)
- **Full control of memory usage**
  - Even where there’s no memory allocation
- **Compiles to Web Assembly**
  - & runs in your favorite browser

Typed Language

- **Statically typed**
  - All types are known at compile-time
  - Generics for data types and code blocks

Typed Language

- **Statically typed**

  – All types are known at compile-time

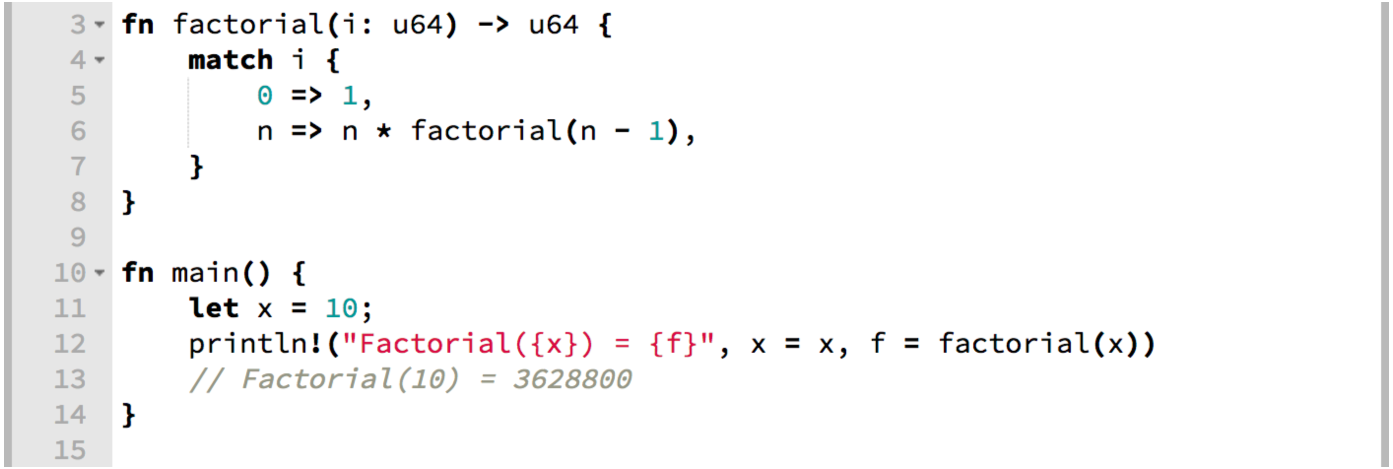
  – Generics for data types and code blocks

- **Strongly typed**

  – Harder to write incorrect programs

  – No runtime null-pointer failures

Syntax



```
fn factorial(i: u64) -> u64 {
    match i {
        0 => 1,
        n => n * factorial(n - 1),
    }
}

fn main() {
    let x = 10;
    println!("Factorial({x}) = {f}", x = x, f = factorial(x))
    // Factorial(10) = 3628800
}
```

Similar to C/C++ & Java

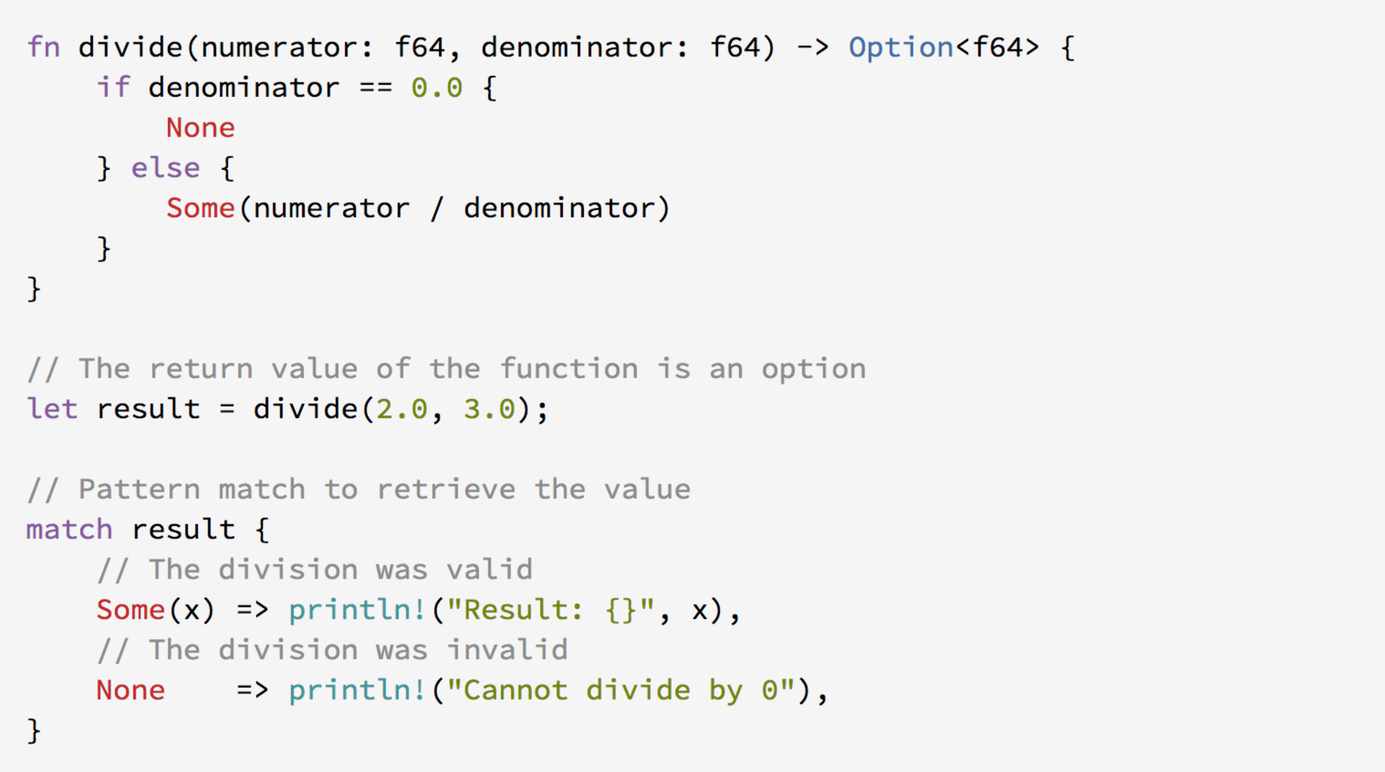
Type System

- **Algebraic types**
  - First Systems PL
  - Tuples, structs, enums, & unions
  - Pattern matching (`match`) for selection and destructure

- **Some basic types**
  - `Option` enum type: `Some` value, or `None`
  - `Result` enum type: `Ok` value, or `Err`

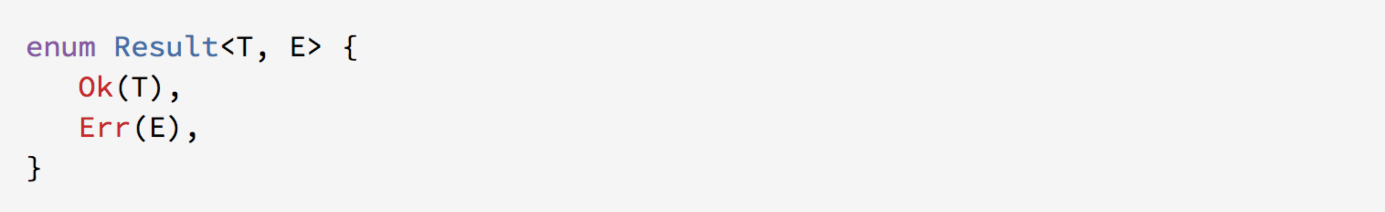
Type System

Option (example)



```
fn divide(numerator: f64, denominator: f64) -> Option<f64> {
    if denominator == 0.0 {
        None
    } else {
        Some(numerator / denominator)
    }
}

// The return value of the function is an option
let result = divide(2.0, 3.0);

// Pattern match to retrieve the value
match result {
    // The division was valid
    Some(x) => println!("Result: {}", x),
    // The division was invalid
    None    => println!("Cannot divide by 0"),
}
```

Type System

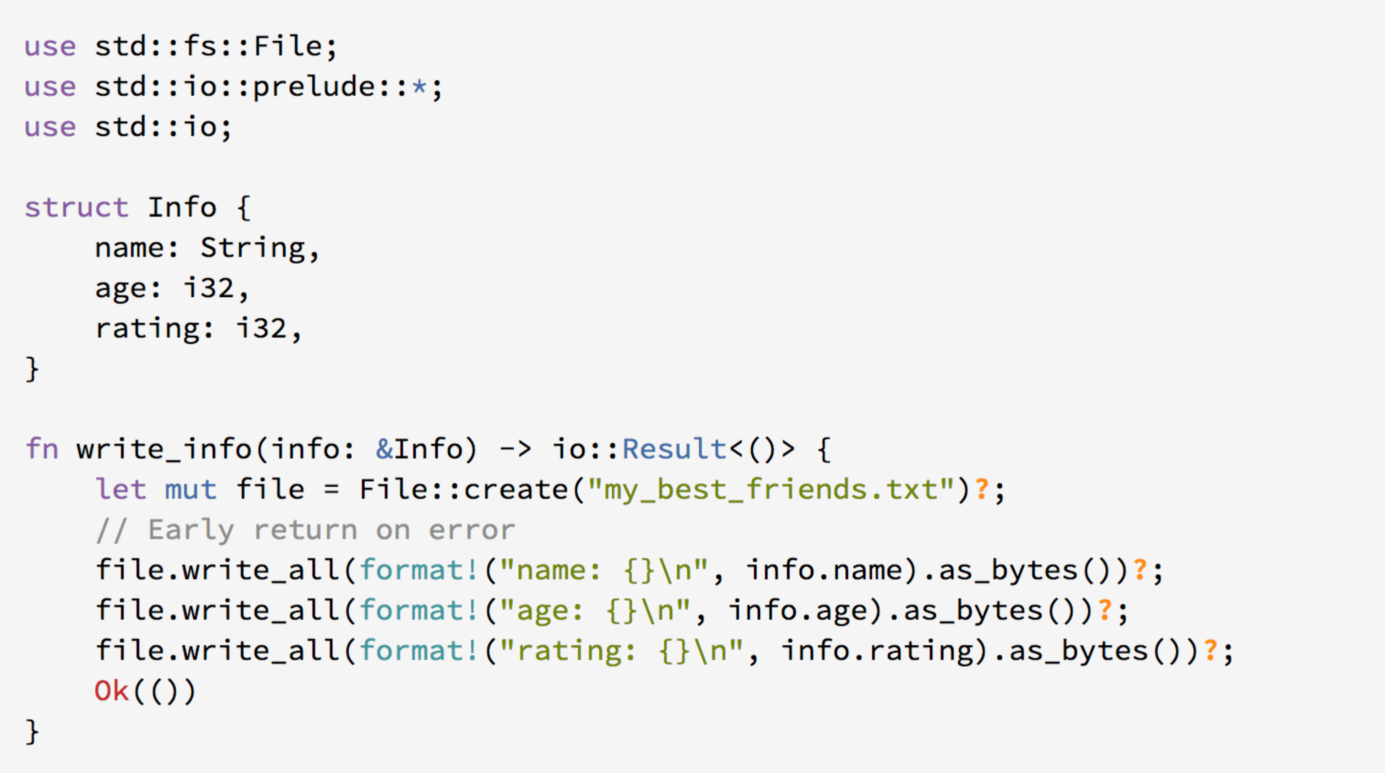


```
enum Result<T, E> {
   Ok(T),
   Err(E),
}
```

Result (definition)

Type System

Result (example)



```rust
use std::fs::File;
use std::io::prelude::*;
use std::io;

struct Info {
    name: String,
    age: i32,
    rating: i32,
}

fn write_info(info: &Info) -> io::Result<()> {
    let mut file = File::create("my_best_friends.txt")?;
    // Early return on error
    file.write_all(format!("name: {}\n", info.name).as_bytes())?;
    file.write_all(format!("age: {}\n", info.age).as_bytes())?;
    file.write_all(format!("rating: {}\n", info.rating).as_bytes())?;
    Ok(())
}
```

Memory Management & Safety

- **No garbage collection**
  – Strict memory management

Memory Management & Safety

- **No garbage collection**
  - Strict memory management

- **Ownership**
  - Memory parts are owned by exactly one variable
  - Destruct memory when variable goes out of scope

Memory Management & Safety

- **No garbage collection**
  – Strict memory management

- **Ownership**
  – Memory parts are owned by exactly one variable
  – Destruct memory when variable goes out of scope

- **Borrow checker**
  – Data-race free
  – Similar to type checker
  – Either read-only pointers or one read-write pointer

Memory Management & Safety

- **No garbage collection**
  – Strict memory management
- **Ownership**
  – Memory parts are owned by exactly one variable
  – Destruct memory when variable goes out of scope
- **Borrow checker**
  – Data-race free
  – Similar to type checker
  – Either read-only pointers or one read-write pointer
- **Lifetimes**
  – ≈ Position in the stack that owns the heap allocation

Interfaces & Impl.s

- **Traits**
  - Define behavior (can’t own data)
  - Inheritance
    - `Deref`

Interfaces & Impl.s

- **Traits**
  - Define behavior (can’t own data)
  - Inheritance
    - `Deref`

- **Impl blocks**
  - Implement types and traits (can’t own data)
  - Composition

Interfaces & Impl.s

- **Traits**
  - Define behavior (can’t own data)
  - Inheritance
    - `Deref`

- **Impl blocks**
  - Implement types and traits (can’t own data)
  - Composition

- **Code blocks**
  - Functions, methods and closures

Interfaces & Impl.s

- **Traits**
  - Define behavior (can’t own data)
  - Inheritance
    - `Deref`

- **Impl blocks**
  - Implement types and traits (can’t own data)
  - Composition

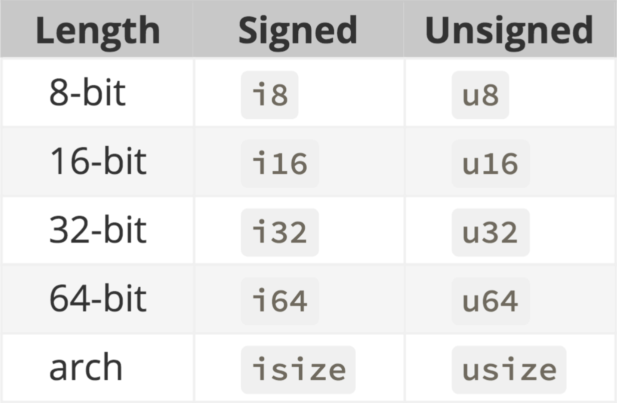
- **Code blocks**
  - Functions, methods and closures

- **Macros**
  - `assert!(), format!(), print!(), println!()`

Characters & Strings

Numeric Types

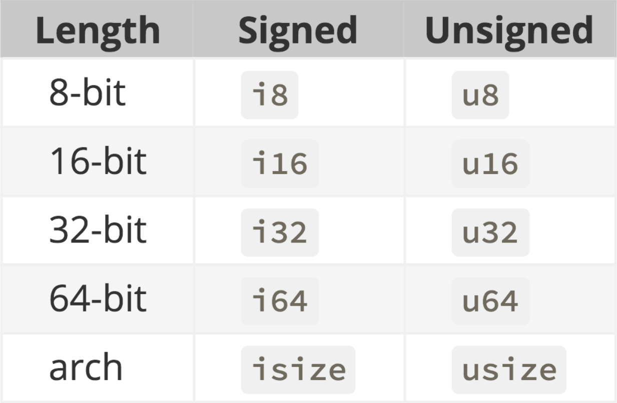
- **Signed & unsigned integer types**



Length	Signed	Unsigned
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
arch	`isize`	`usize`

Numeric Types

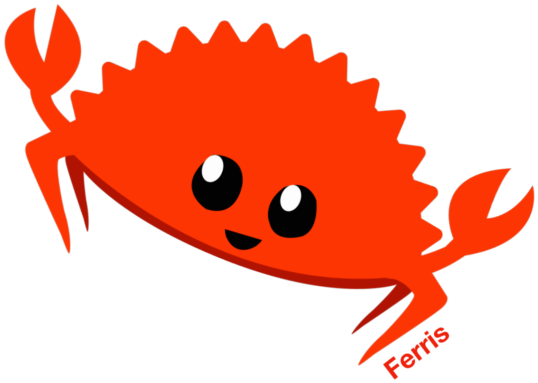
- **Signed & unsigned integer types**



Length	Signed	Unsigned
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
arch	`isize`	`usize`

- **Floating-point types**
  - `f32`, `f64`

```
let x = 1_112_064;
```



```
let x = 1_112_064;
```

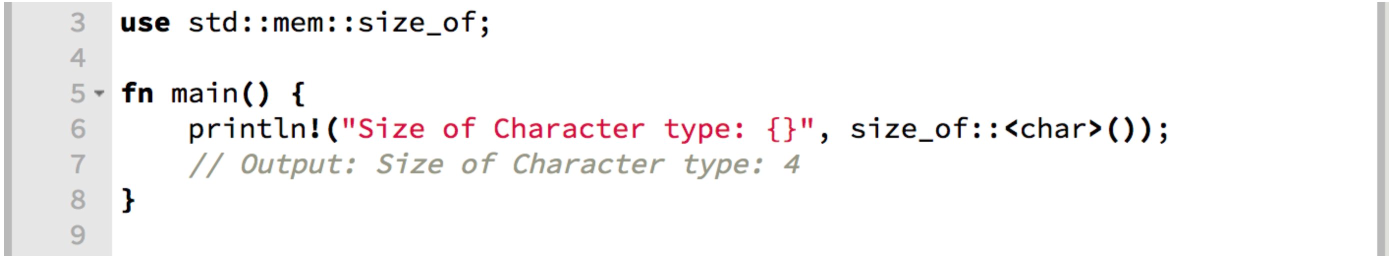
Character Type

Unicode scalar values

- **As defined by The Unicode Standard**

  – “Any Unicode code point, except high-surrogate and low-surrogate code points.”

  - U+0000 to U+D7FF (inclusive)
  - U+E000 to U+10FFFF (inclusive)
  - Total of 1,112,064 code points

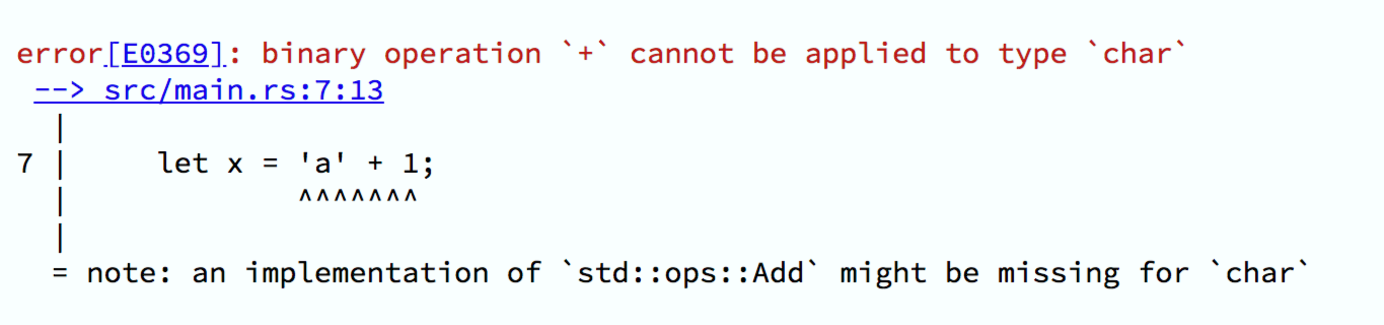


```rust
use std::mem::size_of;

fn main() {
    println!("Size of Character type: {}", size_of::<char>());
    // Output: Size of Character type: 4
}

```

Character Type

Limited integer type

- **No numerical operations on the `char` type**
  - What would the result of `` `U+D7FF + 1` ``?

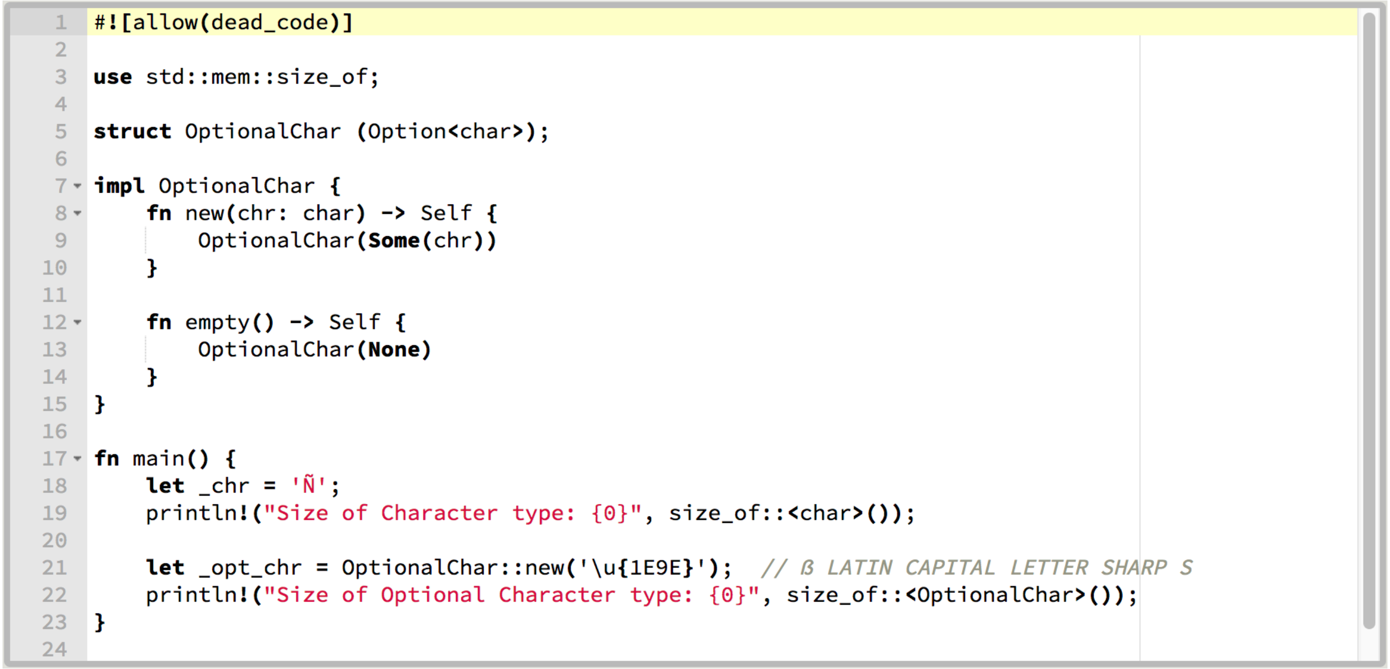


```
error[E0369]: binary operation `+` cannot be applied to type `char`
 --> src/main.rs:7:13
  |
7 |     let x = 'a' + 1;
  |             ^^^^^^^
  |
  = note: an implementation of `std::ops::Add` might be missing for `char`
```

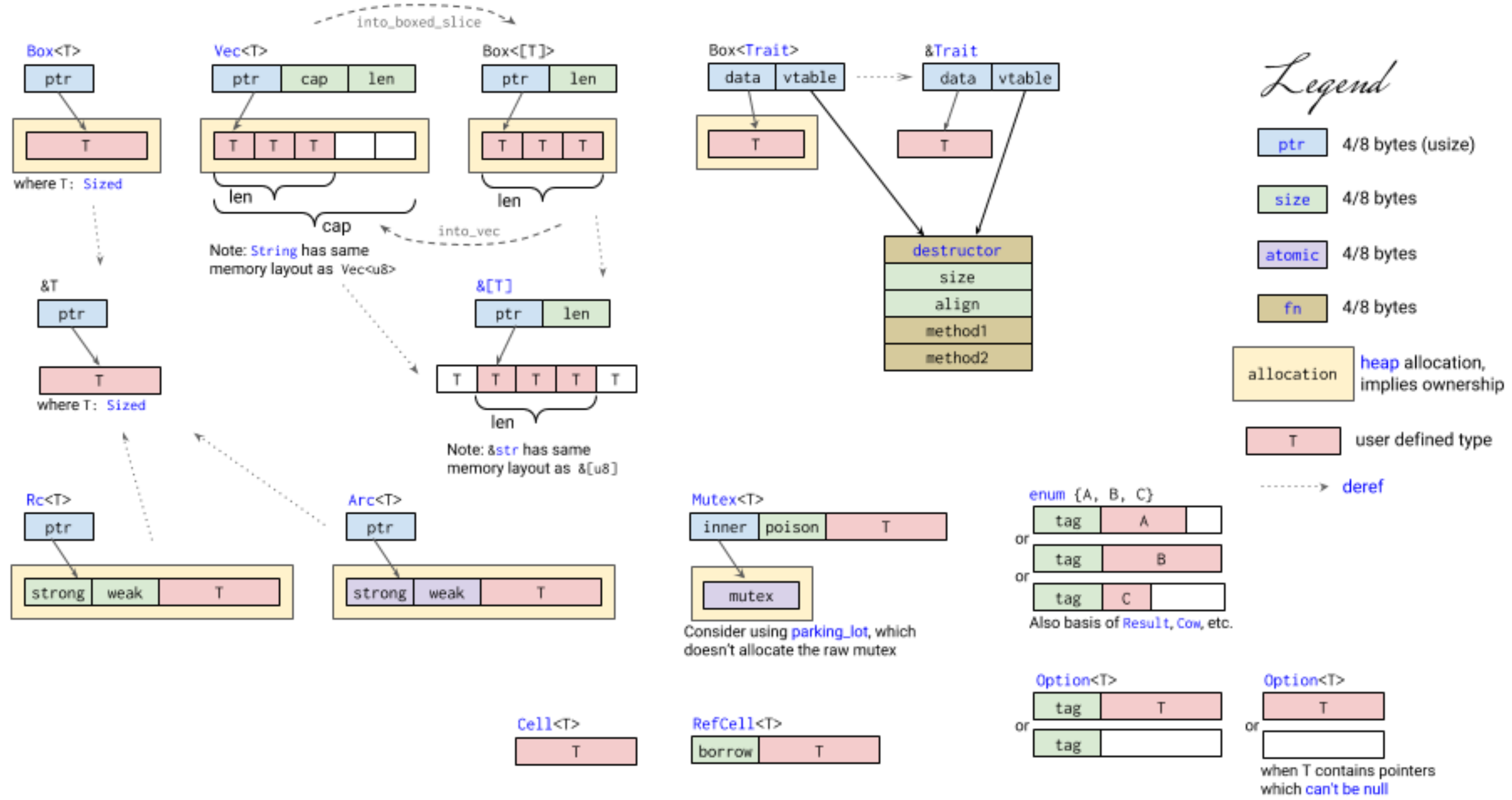
Character Type

Algebraic types in action

- **Compiler knows that all values of the 4 bytes are not used!**



```rust
#![allow(dead_code)]

use std::mem::size_of;

struct OptionalChar (Option<char>);

impl OptionalChar {
    fn new(chr: char) -> Self {
        OptionalChar(Some(chr))
    }

    fn empty() -> Self {
        OptionalChar(None)
    }
}

fn main() {
    let _chr = 'Ñ';
    println!("Size of Character type: {0}", size_of::<char>());

    let _opt_chr = OptionalChar::new('\u{1E9E}');  // ẞ LATIN CAPITAL LETTER SHARP S
    println!("Size of Optional Character type: {0}", size_of::<OptionalChar>());
}

```



Note: String has same memory layout as Vec<u8>

Note: &str has same memory layout as &[u8]

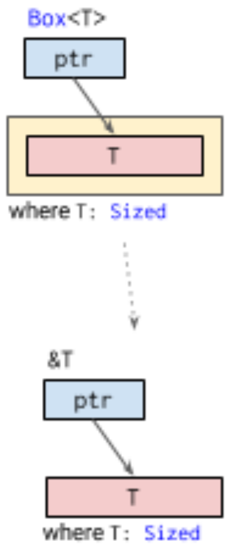
Consider using parking_lot, which doesn't allocate the raw mutex

Also basis of Result, Cow, etc.

when T contains pointers which can't be null

Pointer Types

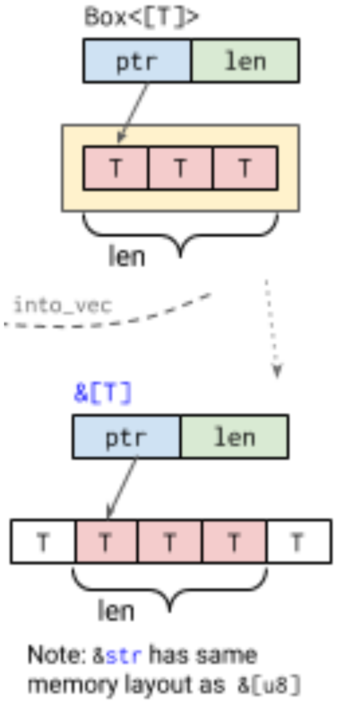
- **Narrow pointers**
  - Point to `Sized` types (size is known at compile-time)
  - Single `usize` value



Pointer Types

- **Narrow pointers**
  - Point to `Sized` types (size is known at compile-time)
  - Single `usize` value

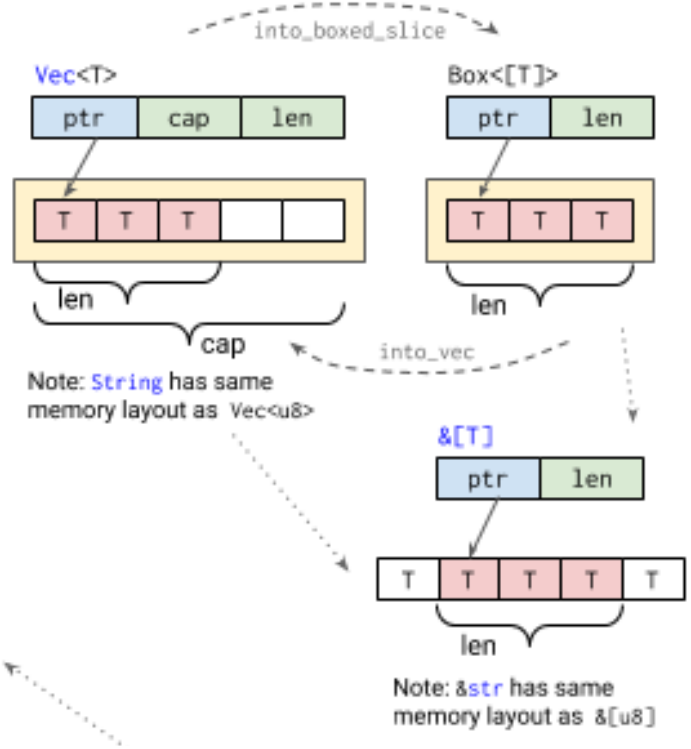
- **Fat Pointers**
  - Point to something with unknown size (at compile-time)
  - Single `usize` value, plus more data



Note: &str has same memory layout as &[u8]

Arrays, Slices & Vectors

- **Arrays**
  - Sized sequence of elements
    - `[T; size]`
  - Unsized sequence of elements
    - `[T]`



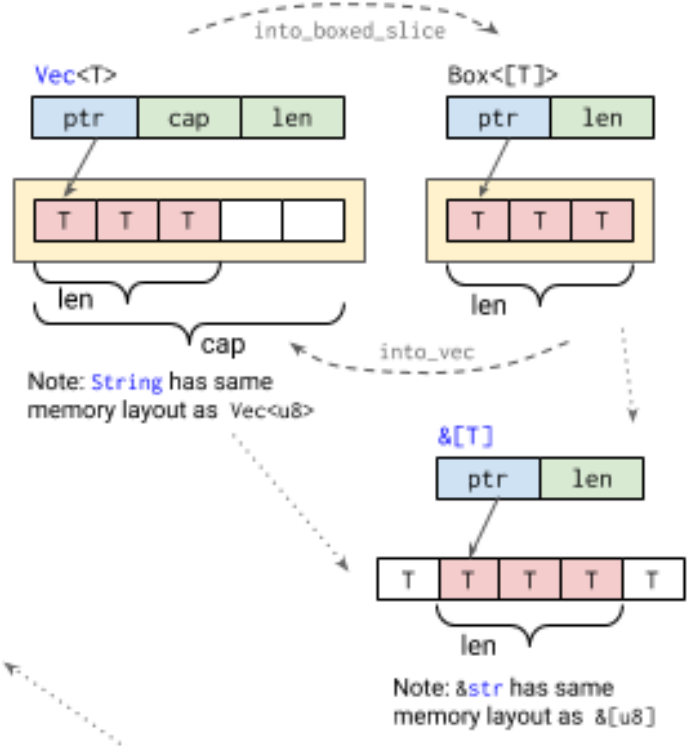
Note: String has same memory layout as Vec<u8>

Note: &str has same memory layout as &[u8]

Arrays, Slices & Vectors

- **Arrays**
  - Sized sequence of elements
    - `[T; size]`
  - Unsized sequence of elements
    - `[T]`

- **Slice**
  - A view into a sequence of elements
    - `&[T]`
  - On arrays, vectors, …



Note: String has same memory layout as Vec<u8>

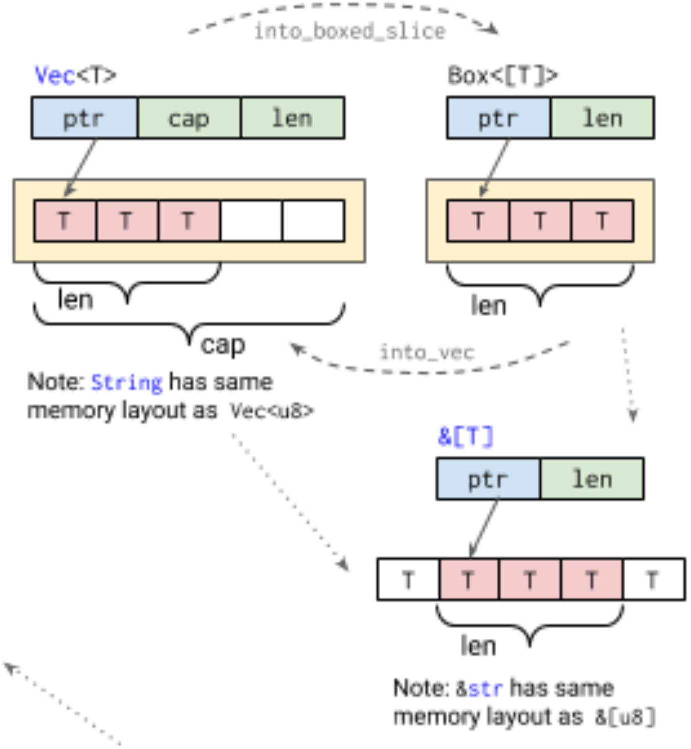
Note: &str has same memory layout as &[u8]

Arrays, Slices & Vectors

- **Arrays**
  - Sized sequence of elements
    - `[T; size]`
  - Unsized sequence of elements
    - `[T]`

- **Slice**
  - A view into a sequence of elements
    - `&[T]`
  - On arrays, vectors, …

- **Vector**
  - A dynamic-length sequence of elements
  - Sit in the heap

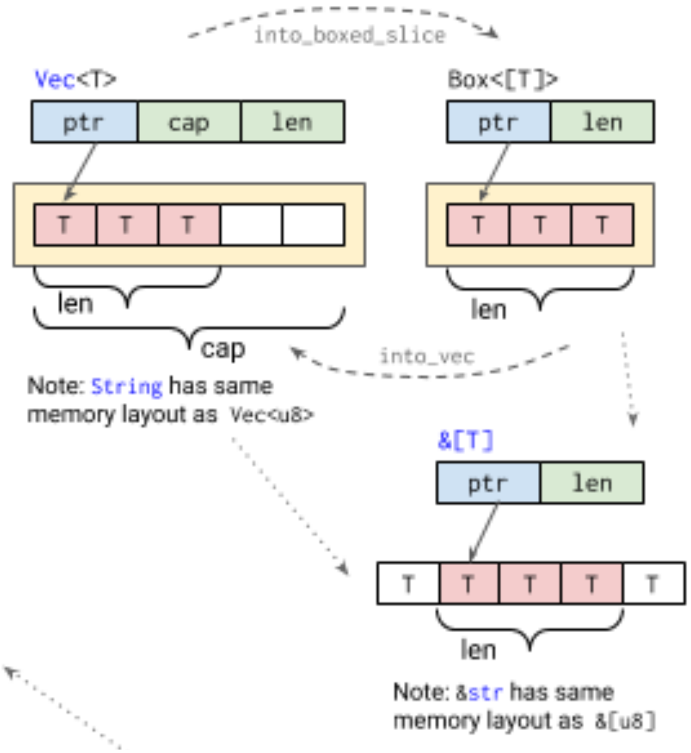


Note: String has same memory layout as Vec<u8>

Note: &str has same memory layout as &[u8]

String Types

- `str`
  - A special `[u8]`
  - Always a valid UTF-8 sequence

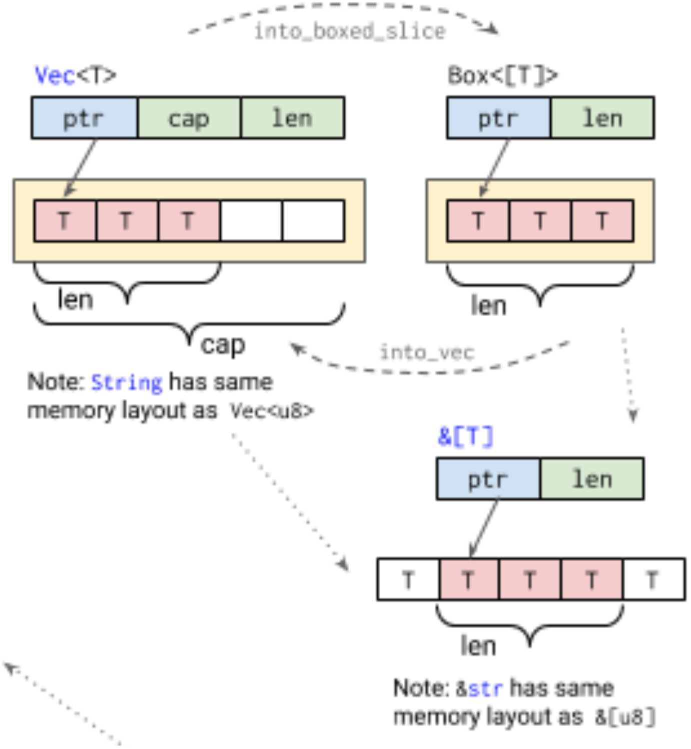


into_boxed_slice
Vec<T>
ptr
cap
len
T
T
T
len
cap
Box<[T]>
ptr
len
T
T
T
len
into_vec
Note: String has same memory layout as Vec<u8>
&[T]
ptr
len
T
T
T
T
T
len
Note: &str has same memory layout as &[u8]

String Types

- `str`
  - A special `[u8]`
  - Always a valid UTF-8 sequence

- `&str`
  - A special `&[u8]`

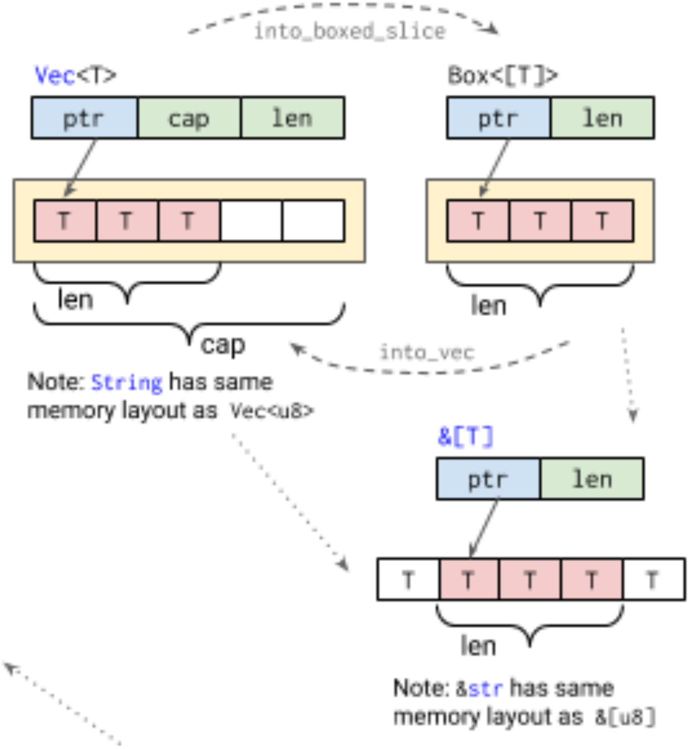


Note: String has same memory layout as Vec<u8>

Note: &str has same memory layout as &[u8]

String Types

- `str`
  - A special `[u8]`
  - Always a valid UTF-8 sequence
- `&str`
  - A special `&[u8]`
- `String`
  - A dynamic UTF-8 sequence
  - Return type of `str` functions that cannot guarantee preserving bytes length



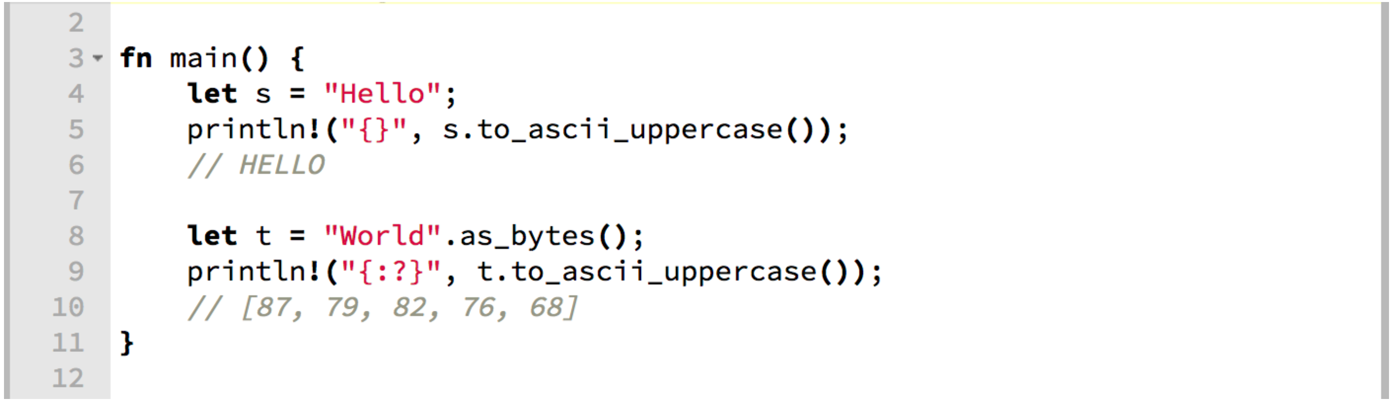
Note: String has same memory layout as Vec<u8>

Note: &str has same memory layout as &[u8]

String Operations

ASCII-only

- **Apply to both** `&[u8]` **and** `&str`

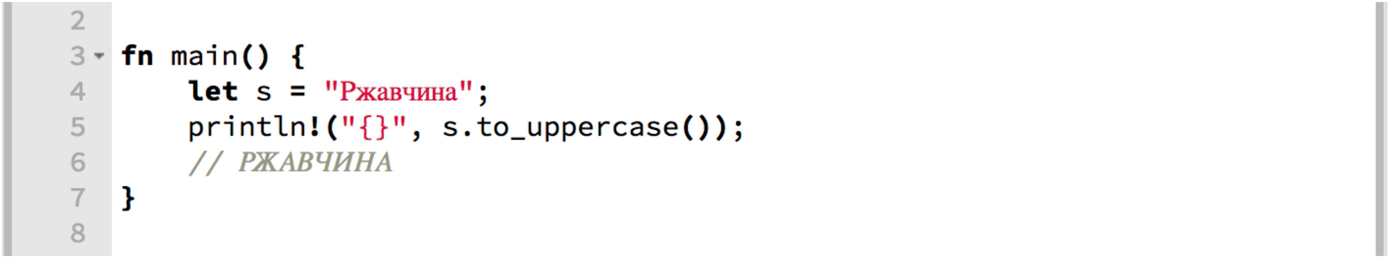


```
fn main() {
    let s = "Hello";
    println!("{}", s.to_ascii_uppercase());
    // HELLO

    let t = "World".as_bytes();
    println!("{:?}", t.to_ascii_uppercase());
    // [87, 79, 82, 76, 68]
}

```

String Operations

Non-ASCII Unicode

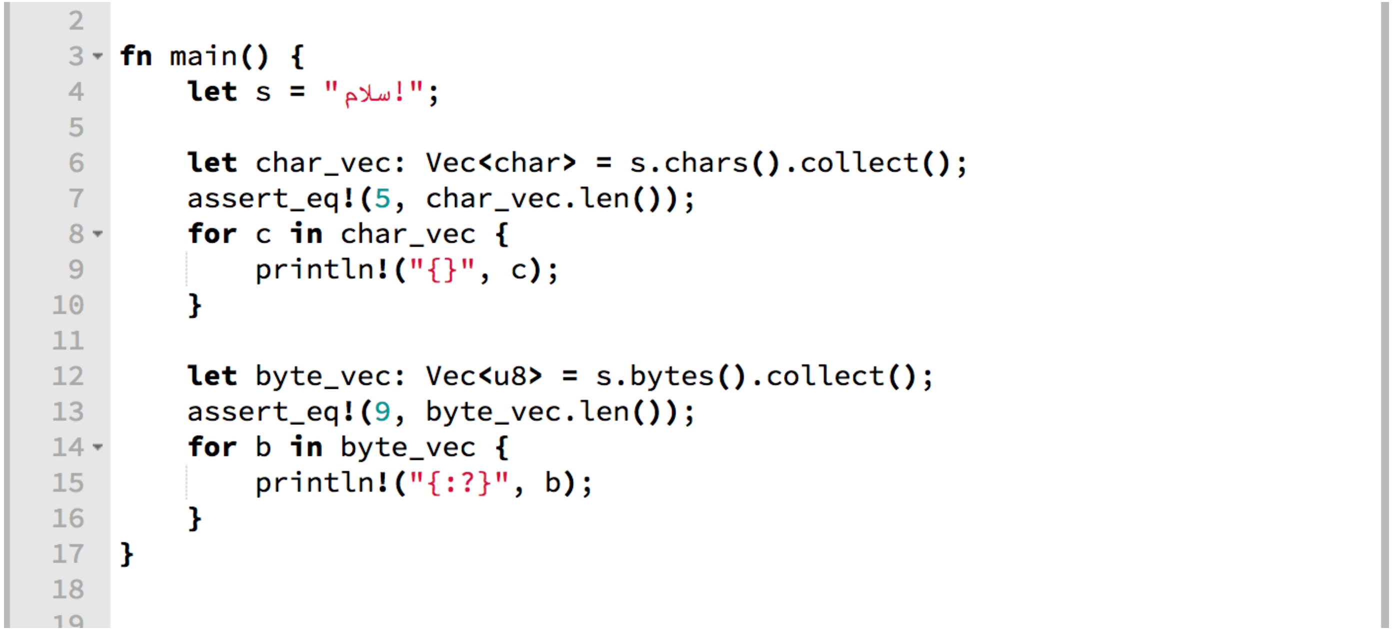
- **Apply only to** `&str`



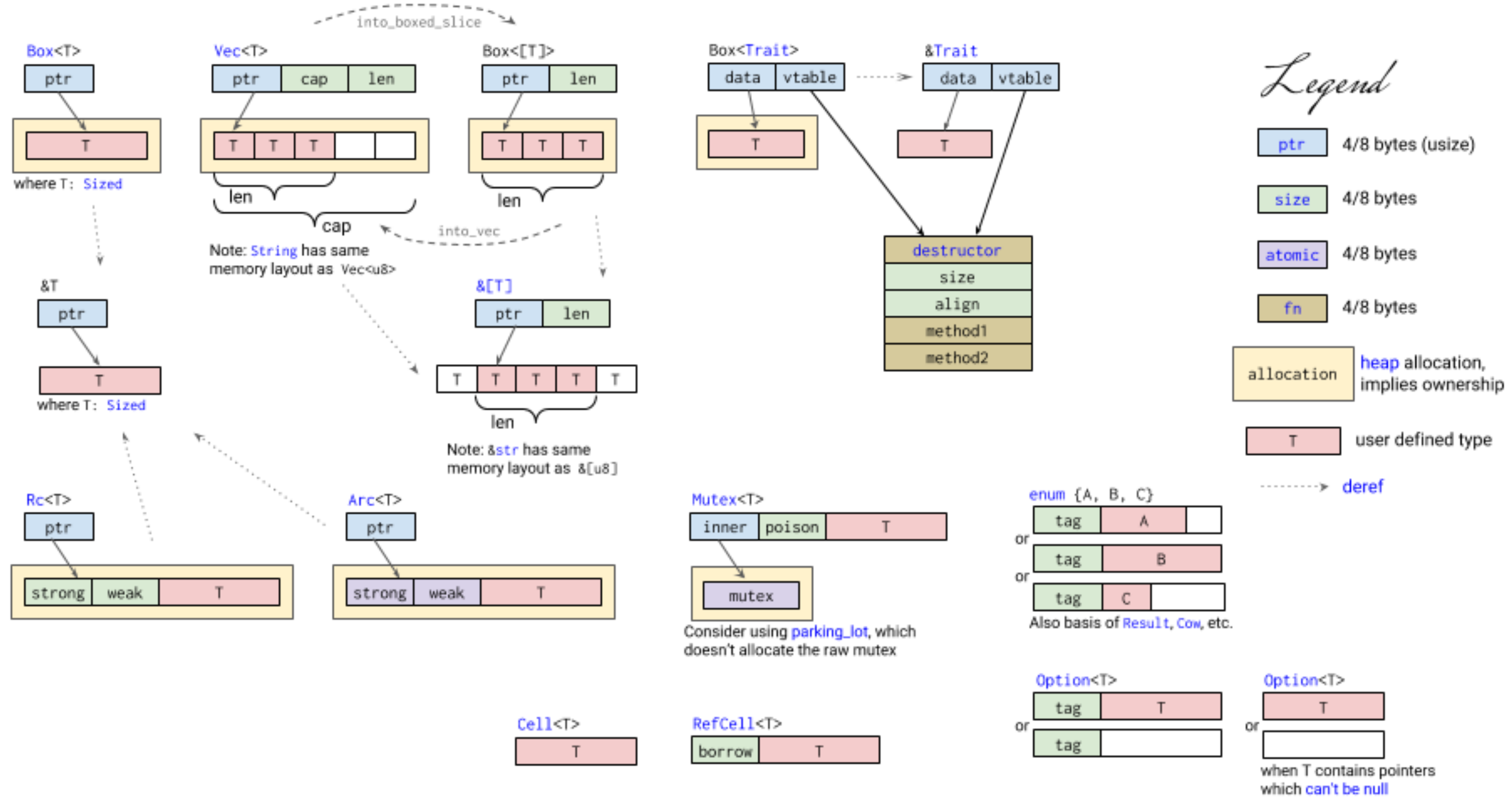
```
fn main() {
    let s = "Ржавчина";
    println!("{}", s.to_uppercase());
    // РЖАВЧИНА
}
```

Iterating Strings

- **Iterating over characters of a string**



```
fn main() {
    let s = "سلام!";

    let char_vec: Vec<char> = s.chars().collect();
    assert_eq!(5, char_vec.len());
    for c in char_vec {
        println!("{}", c);
    }

    let byte_vec: Vec<u8> = s.bytes().collect();
    assert_eq!(9, byte_vec.len());
    for b in byte_vec {
        println!("{:?}", b);
    }
}
```



Note: String has same memory layout as Vec<u8>

Note: &str has same memory layout as &[u8]

Consider using parking_lot, which doesn't allocate the raw mutex

Also basis of Result, Cow, etc.

when T contains pointers which can't be null

Rust container cheat sheet, by Raph Levien, Copyright 2017 Google Inc., released under Creative Commons BY, 2017-04-21, version 0.0.3

It Gets Complicated!

Cross-Platform Encoding Challenges

- **OS & environment variables**
  - File Names
  - Environment variables
  - Command-line parameters

- **Different per system**
  - Unix: bytes; commonly UTF-8 these days
  - Windows: UTF-16, but not always well-formed

Cross-Platform Encoding Challenges

- **OS & environment variables**
  - File Names
  - Environment variables
  - Command-line parameters

- **Different per system**
  - Unix: bytes; commonly UTF-8 these days
  - Windows: UTF-16, but not always well-formed

- `OsStr` **and** `OsString`
  - Internal data depends on OS
  - `&OsStr` is to `OsString` as `&str` is to `String`

Cross-Platform Data Types

- `OsStr` **and** `OsString`
  - Internal data depends on OS
  - `&OsStr` is to `OsString` as `&str` is to `String`

Cross-Platform Data Types

- `OsStr` **and** `OsString`
  - Internal data depends on OS
  - `&OsStr` is to `OsString` as `&str` is to `String`

- **Trait** `std::ffi::OsStr`
  - `pub fn to_str(&self) -> Option<&str>`

Cross-Platform Data Types

- `OsStr` **and** `OsString`
  - Internal data depends on OS
  - `&OsStr` is to `OsString` as `&str` is to `String`

- **Trait** `std::ffi::OsStr`
  - `pub fn to_str(&self) -> Option<&str>`

- **Trait** `std::os::unix::ffi::OsStrExt`
  - `fn from_bytes(slice: &[u8]) -> &Self`
  - `fn as_bytes(&self) -> &[u8]`

Cross-Platform Data Types

- `OsStr` **and** `OsString`
  - Internal data depends on OS
  - `&OsStr` is to `OsString` as `&str` is to `String`

- **Trait** `std::ffi::OsStr`
  - `pub fn to_str(&self) -> Option<&str>`

- **Trait** `std::os::unix::ffi::OsStrExt`
  - `fn from_bytes(slice: &[u8]) -> &Self`
  - `fn as_bytes(&self) -> &[u8]`

- **Trait** `std::os::windows::ffi::OsStrExt`
  - `fn encode_wide(&self) -> EncodeWide`

Working with C APIs

- `CStr` **and** `CString`
  - A borrowed reference to a nul-terminated array of bytes
  - `CStr` is to `CString` as `&str` is to `String`

Working with C APIs

- `CStr` **and** `CString`
  - A borrowed reference to a nul-terminated array of bytes
  - `CStr` is to `CString` as `&str` is to `String`

- **Trait** `std::ffi::CStr`
  - `pub unsafe fn from_ptr<'a>(ptr: *const c_char) -> &'a CStr`
  - `pub fn to_str(&self) -> Result<&str, Utf8Error>`

On Top of the Language

Unicode & i18n Crates

- **Encoding/Charsets**
  – Firefox is already using a Rust component for that!

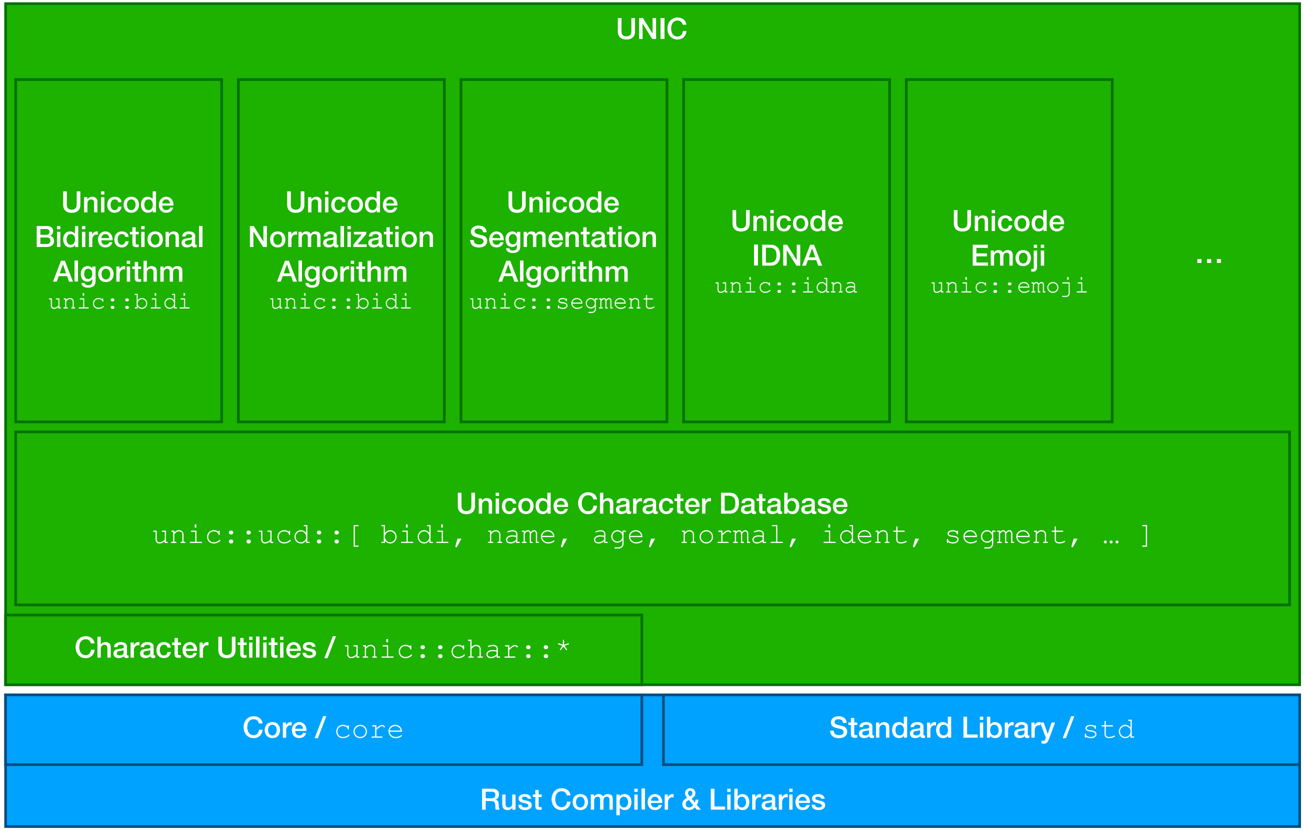
- **Rust Project**
  – String algorithms needed for a compiler

- **Servo Project**
  – Basic string algorithms needed for a rendering engine

- **Locale-aware API**
  – Actually not much available yet
  – WIP by Mozilla, et al.

UNIC Experiment

UNIC: Unicode and i18n Crates for Rust



UNIC
Unicode Bidirectional Algorithm unic::bidi
Unicode Normalization Algorithm unic::bidi
Unicode Segmentation Algorithm unic::segment
Unicode IDNA unic::idna
Unicode Emoji unic::emoji
...
Unicode Character Database unic::ucd::[bidi, name, age, normal, ident, segment, …]
Character Utilities / unic::char::*
Core / core
Standard Library / std
Rust Compiler & Libraries

What’s this?

Hello سلام

How about this?

Hello سلام

A Case of Missing Bidi Context

How about Locale Context?

Hello سلام

Hello سلام

Summary: Programming Languages

- **Machine language**

Summary: Programming Languages

- **Machine language**
- **Procedural**
  - **GOTO**

Summary: Programming Languages

- Machine language
- Procedural
  - GOTO
- Functional

Summary: Programming Languages

- Machine language
- Procedural
    - GOTO
- Functional
- Garbage collection

Summary: Programming Languages

- Machine language
- Procedural
    - GOTO
- Functional
- Garbage collection
- Strict memory management

Summary: Unicode & i18n

- **Byte == Char**

Summary: Unicode & i18n

- Byte == Char
- Contextual Charset

Summary: Unicode & i18n

- Byte == Char
- Contextual Charset
- Separation of text encoding & font encoding

Summary: Unicode & i18n

- Byte == Char
- Contextual Charset
- Separation of text encoding & font encoding
- Unified encoding

Summary: Unicode & i18n

- Byte == Char
- Contextual Charset
- Separation of text encoding & font encoding
- Unified encoding
- Contextual Local

Summary: Unicode & i18n

- Byte == Char
- Contextual Charset
- Separation of text encoding & font encoding
- Unified encoding
- Contextual Local
- ???

HOPE

Additional Resources

- **Rust Community**
  - rust-lang.org
  - doc.rust-lang.org
  - play.rust-lang.org
  - users.rust-lang.org
  - reddit.com/r/rust/
  - rustup.rs
  - crates.io
  - unicode-rs.github.io
  - newrustacean.com
- **Servo, the Parallel Browser Engine Project**
  - servo.org
- **UNIC: Unicode and Internationalization Crates for Rust**
  - https://github.com/open-i18n/rust-unic

질문 ?

質問 ?

שְׁאֵלוֹת?

سؤال؟

پرسش؟

Questions?

प्रश्न?

Quora