Extracted from:

Rust Brain Teasers

Exercise Your Mind

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Rust Brain Teasers

Exercise Your Mind

Herbert Wolverson

edited by Tammy Coron

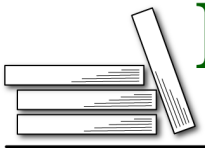
Rust Brain Teasers

Exercise Your Mind

Herbert Wolverson

The Pragmatic Bookshelf

Raleigh, North Carolina



Pragmatic Bookshelf

For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit *https://pragprog.com*.

The team that produced this book includes:

CEO: Dave Rankin
COO: Janet Furlow
Managing Editor: Tammy Coron
Development Editor: Tammy Coron
Copy Editor: L. Sakhi MacMillan
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics
Founders: Andy Hunt and Dave Thomas

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-680509-17-5
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—March 2022

To Henry, my loyal canine coding companion
of thirteen years—who sadly didn't live to see
the book's release.

Puzzle 5

How Long Is a String?

string_length/src/main.rs

```
const HELLO_WORLD : &'static str = "Halló heimur";

fn main() {
    println!("{} is {} characters long.",
        HELLO_WORLD,
        HELLO_WORLD.len()
    );
}
```

Guess the Output

Try to guess what the output is before moving to the next page.

The program will display the following output:

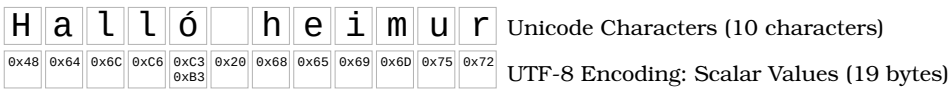
```
Halló heimur is 13 characters long.
```

Discussion

Your eyes aren't deceiving you—"Halló heimur", contains 12 characters (including the space). Let's step back and take a look at how Rust's String type works. The internal struct definition of a String is straightforward:

```
pub struct String {
    vec: Vec<u8>,
}
```

Strings are just a vector of bytes (u8), representing Unicode characters in an encoding named UTF-8. Rust automatically translates your string to UTF-8. The encoding looks like this:



Your original string, "Halló heimur" consists of 11 ASCII characters (including the space) and one Latin-1 Supplement character: the ó. ASCII characters require 1 byte to encode, Latin supplements require 2 bytes.

Rust's string encoding is smart enough to not store extra zeroes for each Unicode character. If it did, String would be a vector of char types. Rust's char is exactly 4 bytes long—the *maximum* size of a single Unicode character.[9] Char variables don't represent a single ASCII character; instead, they represent a *Unicode scalar value*. The scalar value can represent a single glyph or modification to another glyph.

String Length

String.len() counts the number of bytes in the string's backing vector. If a String was storing every character as a char, you'd expect Halló heimur to occupy 48 bytes of memory. Rust's String isn't storing characters; it's storing a byte array representing just the bytes needed to output the stored text.

9. https://doc.rust-lang.org/std/primitive.char.html#representation

Not all UTF-8 characters require all 4 bytes to render. For example, a space requires only 1 byte (0x20), while most Latin Extension characters use 2 bytes. The first byte (0xC3) indicates that the character uses the Latin Extension character region, and the second byte (0xB3 for ó) identifies the character.

The string Halló heimur contains 11 ASCII characters—each using 1 byte of memory—and occupies 11 bytes. Add 2 bytes for the ó and your string occupies 13 bytes of memory.

Counting Characters

You can correctly count the characters in Halló heimur with the following code:

```
println!("{} is {} characters long.",
    HELLO_WORLD,
    HELLO_WORLD
        .chars() // Convert to an iterator over a char sequence
        .count() // Count the characters in the sequence
);
```

When you call my_str.chars(), you're requesting an iterator that returns each element of the string represented as a char.[10] Rust correctly deduces that there are a total of 12 glyphs—or Unicode scalar values—making up the string. The iterator passes each of them to your consumer as a 4-byte char. Even if a glyph only requires 1 or 2 bytes of memory, Rust will allocate all 4 bytes for the char type. Traversing the iterator uses very little extra memory. If you call collect() on the iterator—to create a vector of char data—the vector will consume 40 bytes of memory.

Use my_str.chars() to access individual characters in a String. It's an iterator, so you can use nth, for_each and other iterator functions to find what you're looking for. For example, you can access the fourth character in a string with my_str.chars().nth(4).

Impact of UTF-8 Sizing

Unicode string sizing can be confusing at times, which can lead to surprising results in your code. You need to be aware of the distinction between characters and bytes:

- When you're validating string length, know what counts and what doesn't. For example, if you only accept usernames that are 10 characters or less, you need to decide if you mean glyphs or bytes.

10. https://doc.rust-lang.org/std/str/struct.Chars.html

- When storing strings in databases, you need to remember to allocate enough space for non-English character set strings.
- When transmitting or receiving information to/from a remote API, you need to agree on a length standard for encoding strings in transit.
- If you're writing a program for a memory constrained system, parsing Unicode string character by character can consume a *lot* more memory than you expected. The string `love: ❤` is 7 characters long, requires 12 bytes of storage in a String—and 32 bytes of memory when processed as individual characters. This may seem like a small amount of memory, but if your reader enters the entirety of *War and Peace* into your program's input box, per-character parsing may require more resources than you expected.
- When accessing individual characters in a string, it's much safer to use `chars` as opposed to directly accessing the byte array. Characters are aware of Unicode boundaries—bytes are not. Printing the first 6 bytes of "Können" will only print "Könne". Printing the first 6 characters will output the entire word.

Further Reading

Char

https://doc.rust-lang.org/std/primitive.char.html

String length

https://doc.rust-lang.org/std/string/struct.String.html#method.len

Unicode Symbol Reference

https://www.compart.com/en/unicode/

Wikipedia UTF-8

https://en.wikipedia.org/wiki/UTF-8

String Source Code

https://doc.rust-lang.org/src/alloc/string.rs.html