CoE 164

Computing Platforms

01b: Rust Programming Basics

SYNTAX

Rust adopts a simplified syntax to make it more readable. Although it gives more powers to the programmer, it still looks friendly compared to other “close to memory” languages.

STATEMENTS

Every statement in Rust ends with a semicolon.
Usually, a statement is contained in a single line.

Example

```rust
let mut x = 5;
println!("Hello world!");
io::stdin()
    .read_line(&mut x)
    .expect(":(");
```

COMMENTS

Single-line comments start with two slashes. Multiline comments are surrounded by /* */ “quotation marks”.

Example

```
let mut x = 5; // Comment
/*
 a
 multiline comment!
*/
```

PRELUDE

We can import libraries that we want to use using the `use` keyword. This collection of imports that usually appear on top of a Rust program is called the *prelude*.

Example

```
use std::io;
use std::env;
use std::path::{PathBuf};
```

VARIABLES

Variables are declared using the `let` keyword. All variables should be assigned an initial value before use.

Variables are named using *snake case* - all words should be in *lowercase* and possibly separated by underscores.

Example

```
let seven = 7;
let guess = "hello world!";

let a_long_name = 2.71828;
```

VARIABLES

The `mut` keyword can be added to note that that variable is *mutable* (i.e. can be replaced with a different value). However, we cannot replace it with a value of a different data type.

By default, variables are *not* mutable.

Example

```
let mut ans = 3;
let guess = String::new();

guess = "hello".to_string(); // compile error
ans = 20; // NO compile error
ans = "20"; // compile error
```

CONSTANTS

Constants are declared using the `const` keyword. They are *not* mutable and can only be initialized with a constant expression.

Constants are named using *snake case* - all words should be in *uppercase* and possibly separated by underscores.

Constants should *have* type annotations.

Example

```
let x = 3;
const PI: f64 = 3.14;

const TAU: f64 = PI * 2; // NO compile error
const PI_3: f64 = x * 3; // compile error
```

TYPE ANNOTATION: BASICS

Rust infers the data type of variables depending on its initially-assigned value. However, we can also put a *type annotation* after the name of the variable to force a variable to hold a certain data type.

Example

```
let ans: i64 = 3; // Signed integer
let is_done: bool = true; // Boolean
let mut guess: String = String::new(); //
String
let mut moles: f64 = 6.022e23; // Float
```

VARIABLE COMPARISON

Variable

- Declared using `let`
- Type annotation is optional and can be inferred.
- Can be mutable
- Can be initialized using any expression.

Constant

- Declared using `const`
- Type annotation is required
- Can never be mutable
- Can be initialized *only* with a constant value or an expression with constants.

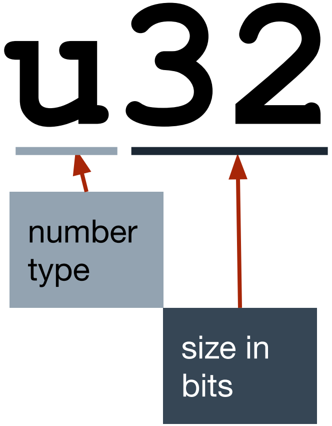
DATA TYPES

Rust has the following built-in data types:

- Signed and unsigned integers
- Floating-point numbers
- Boolean
- Character
- Tuples and arrays
- Strings

DATA TYPES: NUMERIC

Rust supports various numeric types. These are annotated with a number type and their size in bits.



u32

number type

size in bits

Unsigned integer	`u8 u16 u32 u64 u128 usize*`
Signed integer	`i8 i16 i32 i64 i128 isize*`
Floats	`f32 f64`

* The `size` size in bits refers to the size of a memory address. It can either be 32-bit (for x86) or 64-bit (for x64).

DATA TYPES: NUMERIC

Rust supports basic math operations, with the same operations expected from current programming languages.

Example

```
let mut add_i = 3 + 10;
let sub_f = 5.0 - 2.0;
let mul_u = -3i32 + 78i32;
let div_i = 10 / 3;

add_i -= 3; // add_i = add_i - 3;
```

TYPE ANNOTATION: SUFFIXES

Numeric data types can also accept suffixes as substitute for the usual type signature.

Underscores can be used in between numerals as a visual separator.

Example

```
let ans: i64 = 3; // Signed integer
let ans2 = 3i64; // Signed integer
let mut pi_approx = 3.1416f64; // Floating point
let mut e_approx = 2.718_f64; // Floating point
let c = 299_792_458; // Speed of light
```

DATA TYPES: BOOLEAN

A **boolean** has only two possible values - `true` and `false`. It is one byte in size. Note that conditional expressions return a boolean.

Example

```
let true_v2 = true;
let f = false;

let z = true_v2 || f; // logical OR
```

DATA TYPES: CHARACTER

A **character** represents a single glyph surrounded by single quotes. It is four bytes in size and can include non-ASCII characters.

Example

```
let cap_a = 'A';
let enye = 'ñ';
let emoji = '🤔';
```

DATA TYPES: TUPLES

A **tuple** is a compound data type grouping data of various types. It has a fixed length and can be mutable.

Each element of a tuple can be accessed using the dot notation and can be “destructured” to assign each element into individual variables.

Example

```
let str_pair: (&str, i64) = ("3", 3);
let mut pair_ints = (7, 3);
let first_elm = pair_ints.0;

let (fs, ss) = str_pair; // Destructuring
pair_ints.0 = -1; // Mutable assignment
```

DATA TYPES: ARRAYS

An **array** is a compound data type grouping data of same types. It has a fixed length and can be mutable.

Example

```
// Literal syntax
let a = [1, 2, 3, 4, 5];

// 3 elements of type u32
let mut b: [u32; 3] = [6, 7, 8];

// 100 elements, all 0
let visit_list = [0u32; 100];
```

DATA TYPES: ARRAYS

Individual elements can be accessed and assigned to using square brackets. There are also various methods that can be called on arrays for different tasks.

Example

```
let mut a = [1, 2, 3, 4, 5];
println!("{}", a[0]); // Prints 1

a[3] = 2;
println!("{:?}", a); // [1, 2, 3, 2, 5]

let mylen = a.len();  // size of array a is 5
```

RESOURCES

- The Rust Book

CoE 164

Computing Platforms

01b: Rust Programming Basics