2-1: Control Flow and Pattern Matching

Artem Pavlov, TII, Abu Dhabi, 15.04.2025

if branches

```
if cond1 {
    // first branch
} else if cond2 {
    // second branch
} else {
    // final branch
}
```

- cond1 and cond2 are expressions which evaluate to bool
- else and else if branches can be omitted
- You can have multiple of else if branches

if as an expression

```
let x: &str = if a % 2 == 0
    "even"
} else {
    "odd"
};
```

- if branches are expressions and can evaluate to values of any type
- Return type of each branch must match

loop

```
let res: u64 = loop {
    // do stuff
    if cond1 {
        continue;
    }
    // do stuff
    if cond2 {
        break 42u64;
    }
};
```

- continue skips over remaining code and starts new iteration of the innermost loop
- break stops execution of the innermost loop and returns value from it

while loops

```rust
let mut i: u32 = 0;
while i < 10 {
    if i % 2 != 0 {
        continue;
    }
    println!("{i}");
    i += 1;
}
```

- while loops execute as long as the condition is true
- You can use break and continue
- break can not return values

for loops

```
let numbers = [10, 20, 30, 40, 50];
for number in numbers {
    println!("{number}");
}
```

```
for i in 0..10 {
    println!("{i}");
}
for i in (0..10).rev() {
    println!("{i}");
}
```

- for loops iterate over “iterators”
- Collections (e.g. vectors, arrays) can act as iterators
- Ranges (0..10) is a special built-in syntax for creating iterators over specified range
- continue and break can be used with for loops
- break can not return values

Loop labels

```
'outer_loop: loop {
    'next_loop: while cond1 {
        if vals.is_empty() {
            break;
        }
        for val in vals {
            if val.is_stop() {
                continue 'next_loop;
            }
            if val.is_bad() {
                break 'outer_loop;
            }
        }
    }
}
```

- You can break and continue outer loops by using loop labels

Arrays

- Contain a number elements of the same type
- Array type is written as [T; N], where T is a type, and N is size known at compile time
- For example, [u8; 16], [u32; 128]
- Can be constructed as let x: [u8; 3] = [1, 2, 3]; and let x: [u32; 16] = [42; 16];
- Access to elements is performed using indexing operator, i.e. array[3]

Tuples

- A tuple is a collection of values of different types
- Tuples have type (T1, T2, ..), where T1 and T2 potentially different types
- For example, (u8, u32), (u32, u64, u128)
- Tuples also can be “empty”, i.e. ()
- Can be constructed as let x: (u8, u32) = (1, 1024);
- Values can be extracted from the tuple using tuple indexing t.0, t.1

Structs

```
// A unit struct
struct Unit;

//A struct with two fields
struct Foo {
    a: u32,
    b: u64,
}

// "Tuple" struct
struct Bar(Foo, u64);
```

```
let a: Unit = Unit;
let b = Foo { a: 42, b: 64 };
let c = Bar(b, 13);
```

Enums

```
enum MaybeBool {
    True,
    False,
    Dunno,
}

enum Primitives {
    Empty,
    Width(u32),
    Point { x: u32, y: u32 },
    Circle { radius: u32 },
}
```

```
let a: MaybeBool = MaybeBool::True;
use MaybeBool::Dunno;
let b: MaybeBool = Dunno;

use Primitives::Circle;
let p1: Primitives = Primitives::Empty;
let p2: Primitives = Primitives::Width(1);
let p3: Primitives = Circle { radius: 42 };
```

Derivation of standard traits

- Rust allows you to derive commonly used traits for newly defined types
- Deriving a trait adds a standard trait implementation
- Implemented trait adds new functionality for the type

Derivable std traits

- Default: create a “default’ instance of a type
- Debug: allows to format a value using the {:?} formatter
- Hash: allows to compute a hash from &T
- Clone: allows to create T from &T via a copy
- Copy: gives a type 'copy semantics' instead of 'move semantics'
- Eq, PartialEq, Ord, PartialOrd: allows to compare values of a type

Pattern matching

```rust
let x: Option<u32> = get_x();
let new_x = match x {
    None => None,
    Some(i) => Some(i + 1),
};

if let Some(i) = x {
    println!("x is Some({i})");
} else {
    println!("x is None");
}
```

```rust
while let Some(inner_x) = x {
    if inner_x == 0 {
        break;
    }
    let new_x = process(inner_x);
    if new_x > 100 {
        x = Some(new_x);
    } else {
        x = None;
    }
}
```

Error handling

```rust
use std::fs::File;

fn main() {
    let greeting_file_result = File::open("hello.txt");

    let greeting_file = match greeting_file_result {
        Ok(file) => file,
        Err(error) => panic!("Problem opening the file: {:?}", error),
    };
}
```

```rust
use std::fs::File;

fn main() {
    let greeting_file = File::open("hello.txt").unwrap();
}
```

Processing error types

```rust
use std::fs::File;
use std::io::ErrorKind;

fn main() {
    let greeting_file_result = File::open("hello.txt");

    let greeting_file = match greeting_file_result {
        Ok(file) => file,
        Err(error) => match error.kind() {
            ErrorKind::NotFound => match File::create("hello.txt") {
                Ok(fc) => fc,
                Err(e) => panic!("Problem creating the file: {:?}", e),
            },
            other_error => {
                panic!("Problem opening the file: {:?}", other_error);
            }
        },
    };
}
```

Error propagation (the manual way)

```rust
use std::fs::File;
use std::io::{self, Read};

fn read_username_from_file() -> Result<String, io::Error> {
    let username_file_result = File::open("hello.txt");

    let mut username_file = match username_file_result {
        Ok(file) => file,
        Err(e) => return Err(e),
    };

    let mut username = String::new();

    match username_file.read_to_string(&mut username) {
        Ok(_) => Ok(username),
        Err(e) => Err(e),
    }
}
```

Error propagation with ?

```rust
use std::fs::File;
use std::io::{self, Read};

fn read_username_from_file() -> Result<String, io::Error> {
    let mut username_file = File::open("hello.txt")?;
    let mut username = String::new();
    username_file.read_to_string(&mut username)?;
    Ok(username)
}
```

? and Option

```rust
fn add(a: Option<u32>, b: Option<u32>) -> Option<u32> {
    let res = a? + b?;
    Some(res)
}
```

static and const

```
static STATIC_STR: &str = "hello world!";

const MY_BACKDOOR_KEY: [u8; 16] = [42; 16];

const PAGE_SIZE_LOG2: u8 = 14;
const PAGE_SIZE: usize = 1 << PAGE_SIZE_LOG2;
const PAGE_SIZE_IN_U32S: usize = PAGE_SIZE / std::mem::size_of::<u32>();

static DEFAULT_PAGE: &[u8; PAGE_SIZE] = &[0; PAGE_SIZE];
```