CoE 164

Computing Platforms

02c: Rust Selection Constructs

SELECTION CONSTRUCTS

There are constructs available in Rust to more precisely handle conditions.

- `match`
- `if let`
- `while let`

MATCHING

The `match` construct enables comparison of a value against a series of patterns. When the value matches it, the expressions or statements under the match are executed.

The construct can support any data type as long as it is *exhaustive* - that is, it handles all possible values of that data type.

Example

```
let num = 12;

match num {
    0 => {
        println!("zero");
    }
    1 => {
        println!("one");
    }
    other => {
        println!("others");
    }
}
```

MATCHING: CATCH-ALL

Since match expressions should be exhaustive, there should be a provision for a *catch-all* statement in case none of the enumerated patterns have been matched.

Any name can be substituted as a placeholder for the value. An underscore can be used in case the value will be unused in the expression.

Example

```
let u = "else";

match u {
    "admin" => {
        println!("admin role");
    }
    "user" => {
        println!("user role");
    }
    _ => {
        println!("unknown");
    }
}
```

MATCHING: IF MATCH

Match statements can return expressions that can consequently be assigned to a variable. Note that each match should *implicitly* return some value and all matches are enumerated.

Example

```
let num = 12;

let num_str = match num {
    0 => {
        "zero"
    }
    1 => {
        "one"
    }
    other => {
        "others"
    }
};
```

MATCHING: ENUMS

Enums with associated data can be used in statements inside its respective match block.

```
let my_user = UserType::SuperAdmin;

let type_id = match my_user {
    UserType::SuperAdmin => 0,
    UserType::Admin(is_super, chown) => 1,
    UserType::User { chown: chown } => 2,
    UserType::Unknown => 3,
};
```

Example

MATCHING: ENUMS

Struct enums can use the shorthand syntax if the corresponding field name and variable name in the statement are the same. Tuple enums can have variable names to label each of its elements for use in the block.

```
let my_user = UserType::User { chown: 0o755 };

let chown_read = match my_user {
    UserType::User { chown } => chown | 0o400,
    _ => 0o400,
};
```

Example

IF LET

The `if let` construct enables matching whether some data is of a certain enum variant. If the enum holds some data, the data can be accessed inside the `if let` block.

`If let` blocks can be mixed with normal `if else` blocks.

```rust
let my_user = UserType::User { chown: 0o755 };

if let UserType::User { chown } = my_user {
    println!("User with permission: {chown}");
}
else {
    println!("Not a user!");
}
```

Example

WHILE LET

The `while let` is similar to the `if let` block except that the loop will be executed as long as a statement matches a certain enum variant.

```
let mut stk = vec![1, 2, 3];

while let Some(x) = stk.pop() {
    println!("{x}");
}
```

Example

RESOURCES

- The Rust Book

CoE 164

Computing Platforms

02c: Rust Selection Constructs