Rust Programming Notes

Prajval

Contents

1. Types

1.1 Basic Types

1. integer: i8 u8 i16 u16 i32 u32 i64 u64 i128 u128
2. float: f32, f64
3. bool
4. char

1.2 Compound Types

1.2.1 Tuples - can mix data type

```
let x: (i32, bool, char)= (1, true, 'z')
```

1.2.2 Struct

```
struct Student {
        id: String,
        age: u32,
}
```

1.2.3 Option<T>

```
Option<T> is either Some(T) or None
```

1.2.4 Result<T, E>

```
Result<T, E> is either Ok(T) or Err(E)
```

2. Functions

2.1 Function defination

Functions in rust are defined using the fn keyword.

fn fn_name(args): -> ret_type {

`body ...`

}

Example:

```
fn foo(number: f32): -> Option<f32> {
    let log: f32 = if number == 0.0 {
        None
    } else {
        Some(number.log2())
    };
}
```

2.2 Function calling

3. Loops

loops are also expression.

3.1 loop

```rust
let mut x =0;
let y = loop {
    if x == 10 {
        break 42;
    }
    x += 1;
};
// y evaluates to 42

// nested loops
'outer: loop {
    'inner: loop {
        break 'inner;
    }
    break 'outer;
}
```

3.2 while loop

```rust
while x != 10 {
    x += 1;
}
```

3.3 for loop

```rust
for x in 1..=10 {
    println!("{x}");
}
```

3.4 match

match also can be used as a expression

```rust
let opt = 5;
match opt {
    1..=5 => print!("{} ", opt),
        7 => println!("four"),
        _ => print!("None "), // match anything
}
```

4. Common Collections

- Array
- Vec - vector
- VecDeque - vector double ended queue
- String
- Linkedlist
- HashMap
- BTreeMap
- HashSet
- BTreeSet
- BinaryHeap

4.1 Vec

A contiguous growable array type, written Vec but pronounced 'vector'

4.2 VecDeque

4.3 Linkedlist

4.4 HashMap

5. Ownership

- At any point of time there is only one owner, and checks for it at compile time
- copy type don't follow Ownership rules.
- all basic types are copy types
- struct can be copy type if you derive copy and clone and all the members in the struct are copy type

5.1 Ownership with functions

- if we pass a non copy type to a function, we can no longer use it after the call

5.2 Ownership with borrowing

shared xor mutablity: We can either shared with read only permissions or can be owned by one and mutable by the same owner.

- shared but read only or owned by one and mutable

1. shared borrow -> imutable reference (&)
   - shared borrows are copy
2. exclusive borrow -> mutable reference (&mut)
   - exclusive borrows are not copy

Type	Requirements	Access
T	Exactly one owner	All (owned)
&T	Only shared borrows can exist	Read-only
&mut T	Only once exclusive borrow at a time	Read-write, not owned

6. Lifetimes