Hack `postgres` Source Code: Vol I

Chapter 1 : C & Rust

1.46 Closures - Rust

One can also call them as `anonymous functions` in rust. Rust borrowed some features from differen programming paradigms and this feature is derived from `functional programming` . Java forgot its core nature i.e., `Object Oriented Programming` and went for functional programming features - anonymous functions aka lambdas to be in the race parallel to rest of the programming languages which are following different programming paradigms - procedural, objected orietned and functional like python. Rust made sure to have all the useful features from different programming paradigms along with its main goal of being alternative to C/C++ in speed and safety.

Coming back to closures...

It is not possible to store a function in a variable in rust if there is no concept of `Closures` in it. As we can store them in variable, we can actually pass them to functions as arguments.

One more thing, usually, function is not able to access the variables from the scope in which they are defined directly. We need to pass those variables from their defined scope to function scope using function arguments. But, `Closures` in rust can access those variables from the scope in which they are defined. This kind of capability is called `Environment Capturing` .

```
fn main() {

    // main-line-1
    let a:i32 = 10;
    let b:i32 = 20;

    // main-line-2
    let sum_closure = |a,b| { a+b };

    // main-line-3
    let sum:i32 = sum_closure(a,b);

    // main-line-4
    println!("Sum of  a and b is {}", sum);

}
```

`main-line-1` : Initialization of immutable variables `a` and `b` of type `i32` with values `10` and `20` respectively. Rust uses 8 bytes of stack memory for these variable to store.

`main-line-2` : Closures doesn't have any name and they follow different notation. Rust creators inclined towards this notation after seeing ruby and smalltalk closures. However, at this step, we are passing `a and b` to closure body (that looks similar to function body) by placing them between two pipes - `| |` . Rust uses its intelligence ;-) to **infer the types** of `a and b` autmatically. So, we don't have to annotate types (placing of types next to variable name). Same goes with return type. It is apparent that there is a function that accepts two variables in the same way as normal function but with different notation and doing some thing in its body but doesn't have a name. What is it? & How can we use it? One can call those type of function as `closure` and can be used by assigning it to a varable. So, we assigned the closure to a variable `sum_closure` . `sum_closure` is immutable in nature. But, how does rust know whether to borrow or move ownership of variables that are passed to closures. Rust decides that internally. Here, we are using copy types (primitive types). So, rust doesn't move ownership.

`main-line-3` : Calling closure using variable name. We can invoke closure like a function by passign arguments(if there are any). Here, we are passing `a and b` to closure using the variable to which that closure is assigned. Return values from closure is stored into `sum` of type `i32` .

`main-line-4` : printing result using `ptintln!` macro.