Hack `postgres` Source Code: Vol I

Chapter 1 : C & Rust

1.43 Smart Pointer `Rc<T>` in Rust

Rust allows shared ownership of data using a smart pointer called `Rc<T>` .This type `Rc<T>` shares ownership of data `T` created on heap memory and it creates data on heap memory only. Creation of other references which can point to data `T` on heap is done by using `clone` method of `Rc<T>` . Behind the scenens, for each clone, reference count is increased and count of all the references that are pointed to the data `T` is tracked and when there are no references to the data `T` , rust destroys data `T` on heap memory. Rc stands for `reference counting` .

```
use std::rc::Rc;

fn main(){

    //Create i32 data on heap using Rc<T>
    // `first_ref` is a reference pointing to the data on heap memory
    let first_ref:Rc<i32> = Rc::new(11);

    // Find count of references on data 11 on heap
    println!("Count of references {}", Rc::strong_count(&first_ref));

    // `second_ref` is a reference to the data on heap memory
    let second_ref:Rc<i32> = Rc::clone(&first_ref);

    // Find count of references on data 11 on heap
    println!("Count of references {}", Rc::strong_count(&second_ref));

}
```

So, 4 bytes of memory is allocated on heap memory to store number 11 which is of type `i32` using `Rc<T>` . Reference/address of the data is stored in the variable `first_ref` . Later, data's address is shared to another variable `second_ref` using `Rc::clone` operation. By the end of this process, there are two references pointing to same data on heap memory.