```rust
struct ConNum{inner: Mutex<i32>}
unsafe impl Send for ConNum {} // send: mutex ensures
this
unsafe impl Sync for ConNum {} // sync: mutex ensures
this
struct LinkedNode<'a> {
    prev: Option<&'a mut LinkedNode<'a>>,
    next: Option<&'a mut LinkedNode<'a>>,
}
static num: ConNum = ConNum::new(0);

fn main() {
    let n1 = ConNum::new(2024);
    let n2 = n1;
    println!("n2 is {}", *n2.inner.lock().unwrap());
    // println!("n1 is {}", *n1.inner.lock().unwrap());
ownership+move: n1 is moved to n2
    // let n3 = 2023 as ConNum; strict typed: i32 and
ConNum are different types

    let mut ln1 = LinkedNode::new(); let mut ln2 =
LinkedNode::new(); let mut ln3 = LinkedNode::new();
    // ln1.next = Some(&mut ln2); ln3.prev = Some(&mut
ln2); // mutability+borrow: mutable borrow `ln2` twice

    let res = thread::spawn(|| {
        for _ in 0..1000000 {
            // let inner = num.inner.lock().unwrap();
mutability: inner need to be mutable to self-add
            let mut inner = num.inner.lock().unwrap();
// mutability+borrow: immutable reference
            *inner += 1;
            // println!("n2 is {}",
*n2.inner.lock().unwrap()); lifetime: lifetime of n2
maybe short than this thread
        }
    });

    for _ in 0..1000000 {
        let mut inner = num.inner.lock().unwrap(); //
mutability+borrow: immutable reference
        *inner += 1;
    };
}
```