Mutex

```
package main
import (
    "fmt"
    "sync"
)
var mu sync.Mutex
var chain string
func main() {
    chain = "main"
    A()
    fmt.Println(chain)
}
func A() {
    mu.Lock()
    defer mu.Unlock()
    chain = chain + " --> A"
    B()
}
func B() {
    chain = chain + " --> B"
    C()
}
func C() {
    mu.Lock()
    defer mu.Unlock()
    chain = chain + " --> C"
}
```

A: 不能编译

B: 输出main --> A --> B --> C

C: 输出main

D: panic

RWMutex

```
package main
import (
    "fmt"
    "sync"
    "time"
)
var mu sync.RWMutex
var count int
func main() {
    go A()
    time.Sleep(2 * time.Second)
```

```
    mu.Lock()
    defer mu.Unlock()
    count++
    fmt.Println(count)
}
func A() {
    mu.RLock()
    defer mu.RUnlock()
    B()
}
func B() {
    time.Sleep(5 * time.Second)
    C()
}
func C() {
    mu.RLock()
    defer mu.RUnlock()
}
```

A: 不能编译

B: 输出1

C: 程序hang住

D: panic

Waitgroup

```
package main
import (
    "sync"
    "time"
)
func main() {
    var wg sync.WaitGroup
    wg.Add(1)
    go func() {
        time.Sleep(time.Millisecond)
        wg.Done()
        wg.Add(1)
    }()
    wg.Wait()
}
```

A: 不能编译

B: 无输出，正常退出

C: 程序hang住

D: panic

Mutex

```
package main
import (
    "fmt"
    "sync"
)
type MyMutex struct {
    count int
    sync.Mutex
}
func main() {
    var mu MyMutex
    mu.Lock()
    var mu2 = mu
    mu.count++
    mu.Unlock()
    mu2.Lock()
    mu2.count++
    mu2.Unlock()
    fmt.Println(mu.count, mu2.count)
}
```

A: 不能编译

B: 输出1, 1

C: 输出1, 2

D: panic

Map

```
package main
import (
    "fmt"
    "sync"
)
func main() {
    var m sync.Map
    m.LoadOrStore("a", 1)
    m.Delete("a")
    fmt.Println(m.Len())
}
```

A: 不能编译

B: 输出1

C: 输出0

D: panic