Yajush Vyas vyasyajush@gmail.com 8160467125

```
package main

import "fmt"

func main() {
    cnp := make(chan func(), 10)
    for i := 0; i < 4; i++ {
        go func() {
            for f := range cnp {
                f()
            }
        }()
    }
    cnp <- func() {
        fmt.Println("HERE1")
    }
    fmt.Println("Hello")
}
```

1. **Explaining how the highlighted constructs work?**
   The code creates a channel ‘cnp’ to hold up to 10 functions and starts four goroutines, each waiting to receive and execute functions from the channel. When a function that prints "HERE1" is sent to the channel, one of the goroutines picks it up and executes it.

2. **Giving use-cases of what these constructs could be used for.**
   Channels and goroutines in Go are useful for:
   1. Parallel Processing: Running multiple independent tasks at once, like processing images concurrently.
   2. I/O Operations: Handling network or file I/O without blocking the main program flow.
   3. Worker Pools: Managing a pool of workers to process tasks from a queue, such as image or data processing.
   4. Concurrency Control: Synchronizing access to shared resources to avoid race conditions.

3. **What is the significance of the for loop with 4 iterations?**
   The for loop with 4 iterations starts 4 goroutines, enabling concurrent execution of tasks. This setup allows the program to process multiple functions in parallel, improving performance and load distribution.

4. **What is the significance of make(chan func(), 10)?**
   make(chan func(), 10) creates a buffered channel that can hold up to 10 functions, allowing multiple functions to be sent without blocking the sender, thus enabling efficient and smooth concurrent processing.

5. **Why is “HERE1” not getting printed?**
   The functions sent to the channel are not being processed because the goroutines that are supposed to execute them are likely waiting indefinitely for more functions to be received from the channel.