```go
//this code is achieved with golang 1.17
package main

import (
    "fmt"
    "log"
    "math/rand"
    "os"
    "strconv"
    "time"
)

//fabLimit should be >=1 for previous checking
func outputFabTill(fabLimit int){
    fabHead :=1
    fabSecond :=0
    fmt.Print(fabSecond)
    for fabHead<fabLimit{
        fmt.Printf(" %d",fabHead)
        fabHead+=fabSecond
        fabSecond=fabHead-fabSecond
    }
    fmt.Printf("\n")
}

func main() {
    //some initialization checking
    checkStart,err:= strconv.Atoi(os.Args[1]);if err!=nil{
        log.Fatalf("parsing checkStart(first number failed, int expected,%v",err)
    }
    checkEnd,err:= strconv.Atoi(os.Args[2]);if err!=nil{
        log.Fatalf("parsing checkEnd(second number) failed, int expected,%v",err)
    }
    numToCheck,err:= strconv.Atoi(os.Args[3]);if err!=nil{
        log.Fatalf("parsing numToCheck(third number) failed, int expected,%v",err)
    }
    if checkStart<=0||checkEnd<=0||numToCheck<=0{
        log.Fatalf("0/minus input detected,please check input")
    }
    //swap if not in right order
    if checkStart>checkEnd{
        checkStart+=checkEnd
        checkEnd=checkStart
        checkStart-=checkEnd
    }
    //used to save all passed tests
    passedTests:=[]int{}
    for currentChecking:=checkStart;currentChecking<=checkEnd;currentChecking++{
        if numToCheck%currentChecking==0{
            passedTests=append(passedTests,currentChecking)
        }
    }
    //failed to pass, exit
    if len(passedTests)==0{
        panic("numToCheck failed to pass a single task, exiting")
    }
    //output randomly in passed testes
    rand.Seed(time.Now().UnixNano())
    outputFabTill(passedTests[rand.Intn(len(passedTests))])
}
```

Plain text below

```
package main

import (
  "fmt"
  "log"
  "math/rand"
  "os"
  "strconv"
  "time"
)

//fabLimit should be >=1 for previous checking
func outputFabTill(fabLimit int){
  fabHead :=1
  fabSecond :=0
  fmt.Print(fabSecond)
  for fabHead<fabLimit{
    fmt.Printf(" %d",fabHead)
    fabHead+=fabSecond
    fabSecond=fabHead-fabSecond
  }
  fmt.Printf("\n")
}

func main() {
  //some initialization checking
  checkStart,err:= strconv.Atoi(os.Args[1]);if err!=nil{
    log.Fatalf("parsing checkStart(first number failed, int expected ,%v",err)
  }
  checkEnd,err:= strconv.Atoi(os.Args[2]);if err!=nil{
    log.Fatalf("parsing checkEnd(second number) failed, int expected ,%v",err)
  }
  numToCheck,err:= strconv.Atoi(os.Args[3]);if err!=nil{
    log.Fatalf("parsing numToCheck(third number) failed, int expected ,%v",err)
  }
  if checkStart<=0||checkEnd<=0||numToCheck<=0{
    log.Fatalf("0/minus input detected,please check input")
  }
  //swap if not in right order
  if checkStart>checkEnd{
    checkStart+=checkEnd
```

```
        checkEnd=checkStart
        checkStart-=checkEnd
    }
    //used to save all passed tests
    passedTests:=[]int{}
    for currentChecking:=checkStart;currentChecking<=checkEnd;currentChecking++{
        if numToCheck%currentChecking==0{
            passedTests=append(passedTests,currentChecking)
        }
    }
    //failed to pass, exit
    if len(passedTests)==0{
        panic("numToCheck failed to pass a single task, exiting")
    }
    //output randomly in passed testes
    rand.Seed(time.Now().UnixNano())
    outputFabTill(passedTests[rand.Intn(len(passedTests))])
}
```