

LAPORAN PRAKTIKUM
ALGORITMA DAN PEMROGRAMAN 2
MODUL: 5
MATERI: REKURSIF



DISUSUN OLEH:

NAMA: JESIKA METANIA RAHMA ARIFIN

NIM: 103112400080

KELAS: 12 IF 01

DOSEN:

Dimas Fanny Hebrisianto Permadi S.ST, M.Kom

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025/2026

DASAR TEORI

REKURSIF

Apa itu recursive function? Recursive function adalah sebuah function yang memanggil/mengeksekusi dirinya sendiri. Recursive function bisa dikatakan salah satu yang bisa kita gunakan untuk melakukan perulangan. Ketika menulis kode aplikasi, terkadang ada kasus dimana akan lebih mudah jika dilakukan dengan recursive function. Contoh penggunaan sederhana recursive function adalah ketika melakukan operasi factorial.

Saat membuat recursive function, kita harus memastikan function tersebut dapat berhenti. Jika tidak, maka akan terkena error **stack overflow** atau melebihi limit stack karena function terus memanggil dirinya sendiri. Oleh karena itu, biasanya recursive function tidak langsung memanggil dirinya sendiri tetapi bergantung pada kondisi tertentu.

Berikut ini adalah contoh recursive function yang digunakan untuk melakukan operasi factorial.

```
func doFactorial(value int) int {
    if value == 1 {
        return value
    }
    return value * doFactorial(value-1)
}
```

Function tersebut menerima **value** dengan tipe integer dan mengembalikan data dengan tipe integer. Seperti yang sudah dijelaskan sebelumnya bahwa recursive function harus dapat berhenti, maka kita membuat pengkondisian untuk memeriksa jika **value** bernilai 1 maka kita langsung mengembalikannya tanpa memanggil function lagi. Sebaliknya, jika **value** tidak sama dengan satu maka **value** akan dikalikan dengan nilai kembalian dari function **doFactorial** dengan mengisi parameter **value** dikurang 1. Pengurangan ini bertujuan agar parameter yang dikirimkan pada eksekusi function berikutnya adalah berisi nilai pada urutan sebelumnya. Sehingga, jika kita memanggil function **doFactorial** dengan parameter 5 maka parameter berikutnya akan menjadi 4, kemudian 3, kemudian 2, dan 1. Akhirnya recursive function tidak akan dipanggil lagi.

Contoh rekursif menggunakan for-Loop:

```
func doFactorialWithForLoop(value int) int {
    result := 1
    for i := value; i > 0; i-- {
        result *= i
    }
    return result
}
```

beberapa kelebihan dan kekurangan menggunakan recursive function.

Kelebihan:

- Kode mudah ditulis
- Kurangi pemanggilan fungsi yang tidak perlu
- Sangat berguna ketika menerapkan solusi yang sama
- Rekursi mengurangi panjang kode
- Sangat berguna dalam memecahkan masalah struktur data

Kekurangan:

- Recursive function umumnya lebih lambat daripada fungsi non-rekursif
- Terkadang memerlukan memori yang lebih untuk menyimpan hasil sementara dalam stack
- Terkadang sulit untuk menganalisis atau memahami kodennya
- Tidak terlalu efisien dalam konsep kompleksitas waktu
- Kemungkinan out of memory jika kondisi recursive tidak diperiksa dengan benar

Contoh sour code full:

```
package main

import "fmt"

func doFactorial(value int) int {
    if value == 1 {
        return 1
    }
    return value * doFactorial(value-1)
}

func doFactorialWithForLoop(value int) int {
    result := 1
    for i := value; i > 0; i-- {
        result *= i
    }
    return result
}

func main() {
    resultRecursive := doFactorial(5)
    fmt.Println(resultRecursive)

    resultForLoop := doFactorialWithForLoop(5)
    fmt.Println(resultForLoop)
}
```

GUIDED

GUIDED 1

```
package main

import "fmt"

// Fungsi Iteratif untuk menghitung pangkat (base^exp)
func PangkatIteratif(base, exp int) int {
    hasil := 1
    for i := 0; i < exp; i++ {
        hasil *= base
    }
    return hasil
}

//Fungsi iteratif untuk menghitung faktorial (n!)
func faktorialIteratif(n int) int {
    hasil := 1
    for i := 2; i <= n; i++ {
        hasil *= i
    }
    return hasil
}
func main() {
    var base, exp, n int

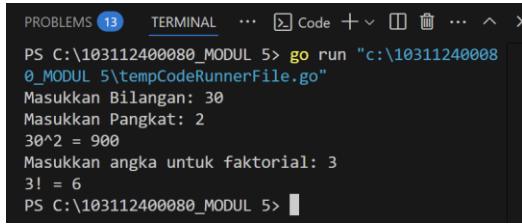
    // Input Pangkat
    fmt.Print("Masukkan Bilangan: ")
    fmt.Scanln(&base)
    fmt.Print("Masukkan Pangkat: ")
    fmt.Scanln(&exp)

    fmt.Printf("%d^%d = %d\n", base, exp, PangkatIteratif(base, exp))

    //Input faktorial
    fmt.Print("Masukkan angka untuk faktorial: ")
    fmt.Scanln(&n)

    fmt.Printf("%d! = %d\n", n, faktorialIteratif(n))
}
```

SOURCODE:



The screenshot shows a terminal window with the following text:
PROBLEMS 13 TERMINAL ... Code + v ⌂ ⌃ ⌄ ⌅ ⌆ ⌇ ⌈ ⌉ ⌋ ⌊ ⌊ ⌁ ⌂ ⌃ ⌄ ⌅ ⌆ ⌇ ⌈ ⌉ ⌋ ⌊ ⌊ ⌁
PS C:\103112400080_MODUL 5> go run "c:\103112400080_0_MODUL 5\tempCodeRunnerFile.go"
Masukkan Bilangan: 30
Masukkan Pangkat: 2
 $30^2 = 900$
Masukkan angka untuk faktorial: 3
 $3! = 6$
PS C:\103112400080_MODUL 5>

DESKRIPSI PROGRAM:

Program ini adalah program yang dibuat untuk menghitung bilangan berpangkat dan factorial menggunakan iterative

GUIDED 2

```
package main

import "fmt"

func pangkatRekursif(base, exp int) int {
    if exp == 0 {
        return 1
    }
    return base * pangkatRekursif(base, exp-1)
}

func faktorialRekursif(n int) int {
    if n == 0 || n == 1 {
        return 1
    }
    return n * faktorialRekursif(n-1)
}

func main() {
    var base, exp, n int
    // Input pangkat
    fmt.Print("Masukkan bilangan: ")
    fmt.Scanln(&base)
    fmt.Print("Masukkan pangkat: ")
    fmt.Scanln(&exp)

    fmt.Printf("%d^%d = %d\n", base, exp, pangkatRekursif(base, exp))

    // Input faktorial
    fmt.Print("Masukkan angka untuk faktorial: ")
    fmt.Scanln(&n)
```

```

    fmt.Printf("%d! = %d\n", n, faktorialRekursif(n))
}

```

SOURCECODE

```

PROBLEMS 13 TERMINAL ... ☰ Code + ⌂ ⌁ ⌈ ⌉ ⌈ ⌉ ⌈ ⌉
PS C:\103112400080_MODUL 5> go run "c:\103112400080_MODUL 5\GuidedRekursif.go"
Masukkan bilangan: 10
Masukkan pangkat: 5
10^5 = 100000
Masukkan angka untuk faktorial: 7
7! = 5040
PS C:\103112400080_MODUL 5>

```

Program ini adalah program yang di buat untuk menghitung bilangan berpangkat dan factorial menggunakan rekursif.

UNGUIDED

UNGUIDED 1

Deret fibonacci adalah sebuah deret dengan nilai suku ke-0 dan ke-1 adalah 0 dan 1, dan nilai suku ke-n selanjutnya adalah hasil penjumlahan dua suku sebelumnya. Secara umum dapat diformulasikan $f_n = f_{n-1} + f_{n-2}$. Berikut ini adalah contoh nilai deret fibonacci hingga suku ke-10. Buatlah program yang mengimplementasikan fungsi rekursif pada deret fibonacci tersebut.

```

// JESIKA METANIA RAHMA ARIFIN
//103112400080
package main

import (
    "fmt"
)

func fibonacci(n int) int {
    if n == 0 {
        return 0
    } else if n == 1 {
        return 1
    } else {
        return fibonacci(n-1) + fibonacci(n-2)
    }
}

```

```
func main() {
    var n int
    fmt.Scan(&n)

    fmt.Println("n\t", "Sn")
    for i := 0; i <= n; i++ {
        fmt.Printf("%d\t%d\n", i, fibonacci(i))
    }
}
```

SOURCECODE:

```
PROBLEMS 13 TERMINAL ... ☒ Code + ✎ 🖌 ... ^ >
PS C:\103112400080_MODUL 5> go run "c:\103112400080_MODUL 5\Unguided1.go"
10
n      Sn
0      0
1      1
2      1
3      2
4      3
5      5
6      8
7      13
8      21
9      34
10     55
PS C:\103112400080_MODUL 5>
```

DESKRIPSI PROGRAM:

Program ini adalah program yang di buat untuk menghitung nilai suku ke 0 dan ke 1 dan nilai suku ke n selanjutnya adalah hasil penjumlahan dua suku sebelumnya.

UNGUIDED 2

Buatlah sebuah program yang digunakan untuk menampilkan pola bintang berikut ini dengan menggunakan fungsi rekursif. N adalah masukan dari user.

```
// JESIKA METANIA RAHMA ARIFIN
//103112400080

package main

import (
    "fmt"
)

func printStars(n int) {
    if n == 0 {
        return
    }
```

```

        }
        fmt.Println("*")
        printStars(n - 1)
    }
func printPattern(n, i int) {
    if i > n {
        return
    }
    printStars(i)
    fmt.Println()
    printPattern(n, i+1)
}

func main() {
    var n int
    fmt.Scan(&n)
    printPattern(n, 1)
}

```

SOURCECODE:

```

PROBLEMS 14 TERMINAL ... Code + ⌂ ⌄ ... ^ >
0_MODUL 5\Unguided2.go"
5
*
**
***
****
*****
PS C:\103112400080_MODUL 5> go run "c:\10311240008
0_MODUL 5\Unguided2.go"
1
*
PS C:\103112400080_MODUL 5> go run "c:\10311240008
0_MODUL 5\Unguided2.go"
3
*
**
***
PS C:\103112400080_MODUL 5>

```

DESKRIPSI PROGRAM:

Program ini adalah program yang di buat untuk menampilkan pola bintang kecil dengan menggunakan fungsi rekursif.

UNGUIDED 3

Buatlah program yang mengimplementasikan rekursif untuk menampilkan faktor bilangan dari suatu N, atau bilangan yang apa saja yang habis membagi N.

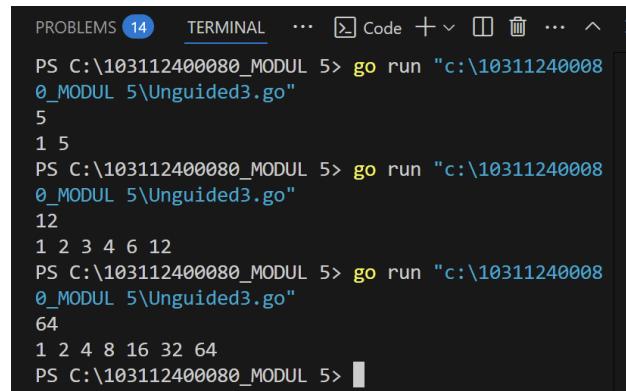
Masukan terdiri dari sebuah bilangan bulat positif N.

Keluaran terdiri dari barisan bilangan yang menjadi faktor dari N (terurut dari 1 hingga N ya).

```
// JESIKA METANIA RAHMA ARIFIN  
//103112400080
```

```
package main  
  
import (  
    "fmt"  
)  
  
func faktorBilangan(n, i int) {  
    if i > n {  
        return  
    }  
    if n%i == 0 {  
        fmt.Print(i, " ")  
    }  
    faktorBilangan(n, i+1)  
}  
  
func main() {  
    var n int  
    fmt.Scan(&n)  
    faktorBilangan(n, 1)  
    fmt.Println()  
}
```

SOURCECODE:



```
PROBLEMS 14 TERMINAL ... ☰ Code + √ ⌂ ⌄ ... ^ >  
PS C:\103112400080_MODUL 5> go run "c:\103112400080_MODUL 5\Unguided3.go"  
5  
1 5  
PS C:\103112400080_MODUL 5> go run "c:\103112400080_MODUL 5\Unguided3.go"  
12  
1 2 3 4 6 12  
PS C:\103112400080_MODUL 5> go run "c:\103112400080_MODUL 5\Unguided3.go"  
64  
1 2 4 8 16 32 64  
PS C:\103112400080_MODUL 5>
```

DESKRIPSI PROGRAM:

Program ini adalah program yang dibuat untuk menampilkan faktor dari suatu bilangan menggunakan implementasi rekursif.

KESIMPULAN

Fungsi rekursif di bahasa Go (Golang) merupakan fungsi yang memanggil diri sendiri selama proses eksekusinya. Rekursi sering dipakai untuk menyelesaikan persoalan yang bisa dibagi menjadi subpersoalan yang lebih kecil dengan pola serupa.

Kelebihan Rekursi di Golang:

- Mudah dan Anggun – Rekursi memfasilitasi penyelesaian masalah dengan kode yang lebih singkat dan lebih mudah dimengerti, terutama dalam contoh seperti perhitungan Fibonacci, faktorial, dan penelusuran dalam struktur data seperti pohon.
- Mengurangi Kode yang Sama – Jika dibandingkan dengan metode iteratif, rekursi seringkali lebih mudah dipahami dalam menangani masalah yang memiliki karakteristik rekursif bawaan.

Kelemahan Rekursi di Golang:

- Memori Stack Overhead – Tiap kali rekursi dipanggil, akan mengonsumsi tambahan memori stack, yang bisa mengakibatkan stack overflow jika kedalamannya berlebihan.
- Tidak Efisien – Rekursi yang tidak dioptimalkan (contohnya, tanpa memoization atau teknik pemrograman dinamis) bisa mengakibatkan pengulangan perhitungan yang memperlambat pelaksanaan.

REFERENSI

<https://blog.ruangdeveloper.com/golang-recursive-function/>