

**LAPORAN PRAKTIKUM
ALGORITMA DAN PEMROGRAMAN 2**

**MODUL IV
PROSEDUR**



Disusun Oleh :

Ghilbran Alfaries Pryma

2311102267

S1IF-11-05

Dosen Pengampu :

Arif Amrulloh, S.Kom., M.Kom

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. DASAR TEORI

Dasar Teori Pemrograman Prosedural dalam Go (Golang)

Pemrograman prosedural adalah salah satu paradigma pemrograman yang berfokus pada prosedur atau fungsi. Dalam paradigma ini, masalah dipecah menjadi kumpulan prosedur atau langkah-langkah logis yang disebut fungsi, dan alur program ditentukan oleh urutan instruksi yang dieksekusi secara berurutan.

Prinsip Dasar Pemrograman Prosedural

Pemrograman prosedural pada dasarnya mengacu pada pembagian masalah menjadi beberapa sub-masalah yang lebih kecil, dan setiap sub-masalah diselesaikan melalui fungsi yang terdefinisi dengan baik. Beberapa prinsip utama pemrograman prosedural adalah:

- **Modularitas:** Program dibagi menjadi bagian-bagian yang lebih kecil yang disebut modul atau fungsi.
- **Urutan Instruksi:** Program dijalankan dengan urutan instruksi secara linear.
- **Reusabilitas Kode:** Fungsi-fungsi yang dibuat bisa dipanggil berulang kali sehingga mengurangi duplikasi kode.
- **Struktur Kontrol:** Meliputi penggunaan kondisi (if-else) dan perulangan (loops).

Go (Golang) sebagai Bahasa Prosedural

Go, atau Golang, adalah bahasa pemrograman yang dikembangkan oleh Google dengan tujuan untuk menyediakan alat yang efisien dan sederhana untuk pengembangan sistem besar. Go adalah bahasa statis dan dikompilasi, yang memiliki sifat mendukung pemrograman prosedural meskipun juga mendukung paradigma pemrograman lainnya seperti pemrograman berorientasi objek dan fungsional.

a. Fungsi dalam Go:

Go mendukung pembuatan fungsi sebagai unit dasar dari program prosedural. Fungsi ini dapat dipanggil dari bagian lain dalam kode dan dapat menerima argumen serta mengembalikan nilai. Contoh dasar fungsi dalam Go:

b. Struktur Kontrol Prosedural:

Sama seperti bahasa pemrograman lainnya, Go mendukung struktur kontrol seperti pernyataan if, for, dan switch yang umum dalam pemrograman prosedural. Contoh penggunaan if dan for dalam Go:

c. Modularitas:

Salah satu fitur penting dari pemrograman prosedural adalah modularitas, yang memungkinkan fungsi-fungsi dikelompokkan menjadi modul-modul terpisah. Dalam Go, modularitas dicapai dengan memanfaatkan paket (packages). Setiap file Go termasuk dalam paket tertentu, dan kita bisa mengimpor serta menggunakan fungsi-fungsi dari paket lain.

3. Kelebihan Pemrograman Prosedural dalam Go

- Sederhana dan Mudah Dipahami: Pemrograman prosedural cenderung mudah dimengerti karena alur program yang jelas dan terstruktur.
- Pemeliharaan Mudah: Kode yang terstruktur dengan baik memudahkan dalam hal pemeliharaan dan pengembangan lebih lanjut.
- Kinerja yang Baik: Go adalah bahasa yang efisien, dan pemrograman proseduralnya mendukung kinerja yang optimal.

4. Keterbatasan Pemrograman Prosedural

- Skalabilitas Terbatas: Untuk sistem yang lebih besar dan kompleks, paradigma prosedural mungkin kurang fleksibel dibandingkan dengan pemrograman berorientasi objek.

- Reusabilitas Terbatas: Sering kali, pemrograman prosedural kurang mendukung pewarisan (inheritance) dan polimorfisme, yang merupakan kelebihan dari pemrograman berorientasi objek.

Jadi pemrograman prosedural adalah cara menyusun program dengan membagi masalah menjadi langkah-langkah atau tugas-tugas yang lebih kecil yang disebut fungsi. Setiap fungsi menyelesaikan bagian tertentu dari masalah, dan langkah-langkah ini dijalankan secara berurutan. Bahasa Go (Golang) mendukung pendekatan ini dengan menyediakan cara mudah untuk membuat fungsi, mengontrol alur program menggunakan perintah seperti if dan for, serta mengorganisir kode melalui paket. Dalam Go, fungsi dapat menerima data, memprosesnya, dan mengembalikan hasil. Contoh penggunaan pemrograman prosedural dalam Go mencakup pembuatan fungsi yang dapat digunakan kembali, yang membantu menjaga program tetap terorganisir dan mudah dimengerti.

II. GUIDED I

Sourcecode

```
package main

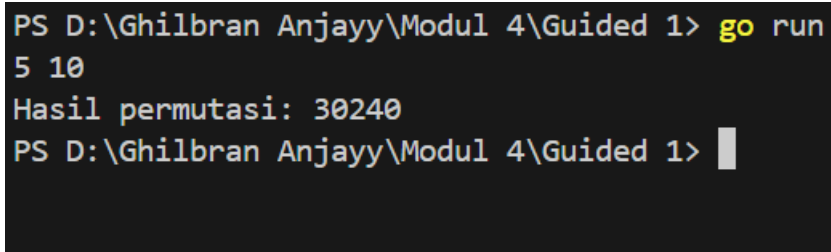
import "fmt"

func main() {
    var a, b int
    fmt.Scan(&a, &b)
    if a >= b {
        permutasi(a, b)
    } else {
        permutasi(b, a)
    }
}

func faktorial(n int, hasil *int) {
    //menggunakan pointer agar dapat mengembalikan nilai
    tanpa return
    *hasil = 1
    for i := 1; i <= n; i++ {
        *hasil *= i
    }
}

func permutasi(n, r int) {
    var hasiln, hasilnr int
    faktorial(n, &hasiln)
    faktorial(n-r, &hasilnr)
    fmt.Println("Hasil permutasi:", hasiln/hasilnr)
}
```

Screenshoot Output



```
PS D:\Ghilbran Anjarry\Modul 4\Guided 1> go run
5 10
Hasil permutasi: 30240
PS D:\Ghilbran Anjarry\Modul 4\Guided 1>
```

Deskripsi Program

Program untuk menghitung permutasi

- n adalah jumlah total elemen.
- r adalah jumlah elemen yang diambil dari total elemen.

1. Fungsi main()

- Program meminta pengguna untuk memasukkan dua angka (variabel `a` dan `b`), yang kemudian disimpan dalam dua variabel integer.
- Setelah menerima input, program memeriksa apakah nilai `a` lebih besar atau sama dengan `b`. Jika ya, fungsi `permutasi(a, b)` dipanggil; jika tidak, fungsi `permutasi(b, a)` dipanggil. Ini memastikan bahwa dalam fungsi `permutasi` selalu lebih besar atau sama dengan `r`.

2. Fungsi faktorial(`n int`, `hasil int`)

- Fungsi ini bertugas menghitung faktorial dari bilangan `n`. Faktorial dihitung dengan cara mengalikan semua bilangan dari 1 hingga `n`.
- Nilai faktorial disimpan dalam variabel `hasil` menggunakan pointer, sehingga perubahan yang dilakukan dalam fungsi ini juga mempengaruhi variabel di luar fungsi.
- Nilai awal `hasil` diinisialisasi dengan 1, lalu dalam loop `for`, dikalikan dengan setiap angka dari 1 hingga `n`.

3. Fungsi permutasi(`n, r int`)

- Fungsi ini menghitung permutasi $P(n, r)$ menggunakan dua kali panggilan ke fungsi faktorial.
- Variabel `hasiln` digunakan untuk menyimpan nilai faktorial dari `n`, dan `hasilnr` menyimpan faktorial dari $(n - r)$.
- Setelah menghitung kedua faktorial, hasil permutasi ditampilkan sebagai hasil pembagian `hasiln / hasilnr`.

GUIDED II

Sourcecode

```
package main

import "fmt"

func main() {

    var s int
    fmt.Print("Input Sisi: ")
    fmt.Scan(&s)

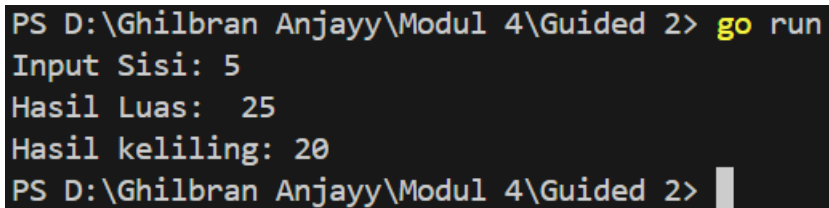
    luasPersegi(s)
    kelilingPersegi(s)

}

func luasPersegi(s int) {
    hasil := s * s
    fmt.Println("Hasil Luas: ", hasil)
}

func kelilingPersegi(s int) {
    hasil := 4 * s
    fmt.Print("Hasil keliling: ", hasil)
}
```

Screenshoot Output



```
PS D:\Ghilbran Anjarry\Modul 4\Guided 2> go run
Input Sisi: 5
Hasil Luas: 25
Hasil keliling: 20
PS D:\Ghilbran Anjarry\Modul 4\Guided 2> █
```

Deskripsi Program

Fungsi main():

- Meminta pengguna untuk memasukkan panjang sisi persegi (s).

- Memanggil dua fungsi: `luasPersegi(s)` untuk menghitung luas dan `kelilingPersegi(s)` untuk menghitung keliling persegi.

Fungsi `luasPersegi(s int)`:

- Menghitung luas persegi menggunakan rumus $Luas = s \times s$
- Menampilkan hasil luas.

Fungsi `kelilingPersegi(s int)`:

- Menghitung keliling persegi menggunakan rumus $= 4 \times s$
- Menampilkan hasil keliling.

III. UNGUIDED I

1. Soal Studi Case

Minggu Ini, mahasiswa Fakultas Informatika mendapatkan tugas dari mata kuliah matematika diskrit untuk mempelajari kombinasi dan permutasi, Jonas salah seorang mahasiswa, Iseng untult mengimplementasikannya ke dalam suatu program. Oleh karena itu bersediakah kalian membantu Jonas? (tidak tentunya ya :p)

Masukan terdiri dari empat buah bilangan asli a , b , c , dan d yang dipisahkan oleh spasi, dengan syarat $a \geq c$ dan $b \geq d$.

Keluaran terdiri dari dua baris. Baris pertama adalah hasil permutasi dan kombinasi a terhadap r , sedangkan baris kedua adalah hasil permutasi dan kombinasi b terhadap d .

Catatan: permutasi (P) dan kombinasi (C) dari n terhadap r ($n \geq r$) dapat dihitung dengan menggunakan persamaan berikut!

$$P(n, r) = \frac{n!}{(n-r)!}, \text{ sedangkan } C(n, r) = \frac{n!}{r!(n-r)!}$$

Sourcecode

```
package main

import "fmt"

// Procedure for factorial
// I.S. defined positive integer n
// F.S. result contains the factorial value of n
func factorial(n int, result *int) {
    *result = 1
    for i := 1; i <= n; i++ {
        *result *= i
    }
}

// Procedure for permutation
// I.S. defined positive integers n and r, and n >= r
// F.S. result contains the permutation value P(n, r)
func permutation(n, r int, result *int) {
    var factN, factNMinusR int
    factorial(n, &factN)
    factorial(n-r, &factNMinusR)
    *result = factN / factNMinusR
}

// Procedure for combination
// I.S. defined positive integers n and r, and n >= r
// F.S. result contains the combination value C(n, r)
func combination(n, r int, result *int) {
    var factN, factR, factNMinusR int
    factorial(n, &factN)
    factorial(r, &factR)
    factorial(n-r, &factNMinusR)
    *result = factN / (factR * factNMinusR)
}

// Main program
func main() {
    var a, b, c, d int
    var permResult, combResult int
```

```

// Input values
fmt.Print("input nilai: ")
_, err := fmt.Scan(&a, &b, &c, &d)
if err != nil {
    fmt.Println("Error input, pastikan memasukkan 4
nilai")
    return
}

// Check conditions a >= c and b >= d
if a >= c && b >= d {
    // Calculate and display permutation and combination
for a and c
    permutation(a, c, &permResult)
    fmt.Println("Permutasi:", permResult)

    combination(a, c, &combResult)
    fmt.Println("Kombinasi:", combResult)

    // Calculate and display permutation and combination
for b and d
    permutation(b, d, &permResult)
    fmt.Println("Permutasi:", permResult)

    combination(b, d, &combResult)
    fmt.Println("Kombinasi:", combResult)
} else {
    fmt.Println("tidak memenuhi kondisi")
}
}

```

Screenshoot Output

```

input nilai: 5 10 3 10
Permutasi: 60
Kombinasi: 10
Permutasi: 3628800
Kombinasi: 1
PS D:\TUGAS SEMESTER 3\Praktikum alpro 2\Modul 4\Ungu

```

Deskripsi Program

- Program ini menghitung permutasi dan kombinasi dari dua pasang bilangan bulat, dengan syarat bahwa bilangan pertama harus lebih besar atau sama dengan bilangan kedua dalam setiap pasangan.
- Menggunakan prosedur untuk faktorisasi, permutasi, dan kombinasi, dengan hasil disimpan dalam variabel yang dipass-in sebagai pointer untuk memungkinkan perubahan nilai dalam prosedur.

UNGUIDED II

2. Soal studi case

Kompetisi pemrograman tingkat nasional berlangsung ketat. Setiap peserta diberikan 8 soal yang harus dapat diselesaikan dalam waktu 5 jam saja. Peserta yang berhasil menyelesaikan soal paling banyak dalam waktu paling singkat adalah pemenangnya.

Buat program **gema** yang mencari pemenang dari daftar peserta yang diberikan. Program harus dibuat modular, yaitu dengan membuat prosedur `hitungSkor` yang mengembalikan total soal dan total skor yang dikerjakan oleh seorang peserta, melalui parameter formal. Pembacaan nama peserta dilakukan di program utama, sedangkan waktu pengerjaan dibaca di dalam prosedur.

prosedure `hitungSkor` (in/out soal, skor : integer)

Setiap baris **masuk** dimulai dengan satu string nama peserta tersebut diikuti dengan adalah 8 Integer yang menyatakan berapa lama (dalam

menit) peserta tersebut menyelesaikan soal. Jika tidak berhasil atau tidak mengirimkan jawaban maka otomatis dianggap menyelesaikan dalam waktu 5 jam 1 menit (301 menit).

Satu baris **keluaran** berisi nama pemenang, jumlah soal yang diselesaikan, dan nilai yang diperoleh. Nilai adalah total waktu yang dibutuhkan untuk menyelesaikan soal yang berhasil diselesaikan.

| No | Masukan | Keluaran |
|----|--|--------------|
| 1 | Astuti 20 50 301 301 61 71 75 10 Bertha 25 47 301 26 50 60 65 21 Selesai | Bertha 7 294 |

Keterangan:

Astuti menyelesaikan 6 soal dalam waktu 287 menit, sedangkan Bertha 7 soal dalam waktu 294 menit. Karena Bertha menyelesaikan lebih banyak, maka Bertha menang. Jika keduanya menyelesaikan sama banyak, aka pemenang adalah yang menyelesaikan dengan total waktu paling kecil.

Source code

```
package main

import (
    "bufio"
    "fmt"
    "os"
    "strconv"
    "strings"
)

func hitungSkor(waktuPengerjaan []int, soal *int, skor *int)
{
    *soal = 0
    *skor = 0
    for _, waktu := range waktuPengerjaan {
        if waktu <= 300 { // 5 jam = 300 menit
            *soal++
        }
    }
}
```

```

        *skor += waktu
    }
}

func main() {
    scanner := bufio.NewScanner(os.Stdin)
    pesertaTerbaik := ""
    soalTerbaik := 0
    skorTerbaik := 0

    for scanner.Scan() {
        input := scanner.Text()
        if strings.ToLower(input) == "selesai" {
            break
        }

        data := strings.Fields(input)
        if len(data) < 9 {
            fmt.Println("Error: Input tidak lengkap")
            continue
        }

        nama := data[0]
        waktuPengerjaan := make([]int, 8)

        for i := 0; i < 8; i++ {
            waktu, err := strconv.Atoi(data[i+1])
            if err != nil || waktu > 300 {
                waktu = 301 // Jika input tidak valid atau >
300, anggap 301 menit
            }
            waktuPengerjaan[i] = waktu
        }

        var soal, skor int
        hitungSkor(waktuPengerjaan, &soal, &skor)

        if soal > soalTerbaik || (soal == soalTerbaik &&
skor > skorTerbaik) {
            pesertaTerbaik = nama
            soalTerbaik = soal
            skorTerbaik = skor
        }
    }
}

```

```

        if pesertaTerbaik != "" {
            fmt.Printf("%s %d %d\n", pesertaTerbaik,
soalTerbaik, skorTerbaik)
        } else {
            fmt.Println("Tidak ada peserta yang valid")
        }

        if err := scanner.Err(); err != nil {
            fmt.Fprintln(os.Stderr, "Error membaca input:", err)
        }
    }
}

```

Screenshoot Output

```

Astuti 20 50 301 301 61 71 75 10
Bertha 25 47 301 26 50 60 65 21
Selesai
Bertha 7 294
PS D:\TUGAS SEMESTER 3\Praktikum alpro 2\Modul 4\Unguided 2>

```

Deskripsi Program

1. Program menerima input berupa nama peserta dan 8 waktu pengerjaan soal.
2. Waktu pengerjaan dihitung, dan jika valid (≤ 300 menit), soal tersebut dianggap dikerjakan.
3. Program terus memproses data hingga pengguna mengetik "selesai".
4. Pada akhir program, peserta terbaik ditampilkan dengan jumlah soal yang dikerjakan dan skor totalnya.

UNGUIDED III

3. Soal study case

Skiena dan Revilla dalam Programming Challenges mendefinisikan sebuah deret bilangan. Deret dimulai dengan sebuah bilangan bulat n . Jika bilangan n saat itu genap, maka suku berikutnya adalah $1/2n$, tetapi jika ganjil maka suku berikutnya bernilai $3n+1$. Rumus yang sama digunakan terus menerus untuk mencari suku berikutnya. Deret berakhir ketika suku terakhir bernilai 1. Sebagai contoh Jika dimulai dengan $n=22$, maka deret bilangan yang diperoleh adalah:

22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

Untuk suku awal sampai dengan 1000000, diketahui deret selalu mencapai suku dengan nilai 1.

Buat program sklena yang akan mencetak setiap suku dari deret yang dijelaskan di atas untuk nilai suku awal yang diberikan. Pencetakan deret harus dibuat dalam prosedur cetak Deret yang mempunyai 1 parameter formal, yaitu nilai dari suku awal.

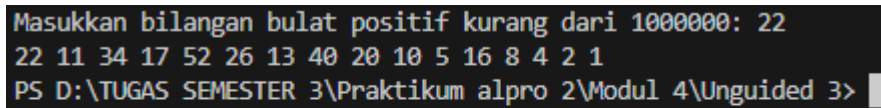
prosedure cetak Deret(int: integer)

Masukan berupa satu bilangan integer positif yang lebih kecil dari 1000000.

Keluaran terdiri dari satu baris saja. Setiap suku dari deret tersebut dicetak dalam baris yang dan dipisahkan oleh sebuah spasi.

| No | Masukan | Keluaran |
|----|---------|--|
| 1 | 22 | 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1 |

Screenshoot Output



```
Masukkan bilangan bulat positif kurang dari 1000000: 22
22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
PS D:\TUGAS SEMESTER 3\Praktikum alpro 2\Modul 4\Unguided 3>
```

Deskripsi program

Program ini mencetak deret Collatz berdasarkan input bilangan bulat positif kurang dari 1.000.000. Aturannya:

- Jika bilangan genap, bagi dengan 2.
- Jika ganjil, kalikan 3 lalu tambah 1. Proses berulang sampai mencapai angka 1.

Program juga memvalidasi input agar hanya menerima bilangan bulat positif dalam rentang yang diizinkan. Jika input tidak valid, program akan menampilkan pesan kesalahan.

DAFTAR PUSTAKA

1. Donovan, A. A., & Kernighan, B. W. (2015). *The Go Programming Language*. Addison-Wesley Professional.
2. Pike, R., Thompson, K., & Griesemer, R. (2010). *The Go Programming Language Specification*. Google Inc. Retrieved from https://golang.org/doc/go_spec.html
3. W. Richard Stevens, Stephen A. Rago (2013). *Advanced Programming in the UNIX Environment*. Addison-Wesley.
4. Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional.
5. Kernighan, B. W., & Ritchie, D. M. (1988). *The C Programming Language* (2nd ed.). Prentice Hall.