

LAPORAN PRAKTIKUM
ALGORITMA DAN PEMROGRAMAN 1
MODUL 16
“SKEMA PEMROSESAN SEKUENSIAL”



DISUSUN OLEH:
ANGGUN WAHYU WIDIYANA
103112480280
S1 IF-12-01
DOSEN:
Yohani Setiya Rafika Nur, M. Kom.

PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2024/2025

DASAR TEORI

Pengertian Pemrosesan sekuensial

Pemrosesan sekuensial adalah konsep dasar dalam pemrograman dimana setiap instruksi dieksekusi satu per satu secara berurutan. Setiap instruksi akan dijalankan setelah instruksi sebelumnya selesai dieksekusi. Dalam bahasa Go, pemrosesan sekuensial merupakan dasar dari eksekusi program yang dimulai dari fungsi `main()`.

Pembacaan Data Tanpa Marker pada Akhir Rangkaian Data

Pola ini memperlihatkan bahwa semua data yang diberikan pada masukan adalah data yang harus diproses. Berikut contoh polanya:

	Notasi Algoritma	Notasi dalam bahasa Go
1	input (n)	<code>fmt.Scanln(&n)</code>
2	<code>i = 1</code>	<code>i = 0</code>
3	while <code>i <= n</code> do	for <code>i < n</code> {
4	input (dat)	<code>fmt.Scan(&dat)</code>
5	{ kode untuk memproses dat }	// kode untuk memproses dat
6	<code>i = i + 1</code>	<code>i = i + 1</code>
7	endwhile	}

Pada pola di atas, terlihat bahwa variabel `dat` yang diperoleh pada baris ke-4 adalah data valid yang pasti diproses pada baris ke-5. Baris ke-5 bisa berisi potongan kode apapun yang digunakan untuk memproses variabel *dat* tersebut.

Berikut contohnya pada potongan program pencarian nilai maksimum:

	Notasi Algoritma	Notasi dalam bahasa Go
1	input (n)	<code>fmt.Scan(&n)</code>
2	<code>max = {BILANGAN_KECIL}</code>	<code>max = // BILANGAN_KECIL</code>
3	<code>i = 1</code>	<code>i = 1</code>
4	while <code>i <= n</code> do	for <code>i <= n</code> {
5	input (dat)	<code>fmt.Scan(&dat)</code>
6	if <code>dat > max</code> then	if <code>dat > max</code> {
7	<code>max = dat</code>	<code>max = dat</code>
8	endif	}
9	<code>i = i + 1</code>	<code>i = i + 1</code>
10	endwhile	}
11	output ("Data terbesar", max)	<code>fmt.Println("Data terbesar", max)</code>

Pembacaan Data dengan Marker pada Akhir Rangkaian Data

Pada pola dengan marker, terdapat data yang dipersiapkan khusus untuk menghentikan perulangan. Artinya semua data yang diberikan pada masukan adalah data yang valid, kecuali data yang terakhir, karena digunakan untuk menghentikan perulangan. Berikut contoh polanya:

	Notasi Algoritma	Notasi dalam bahasa Go
1	input (dat)	fmt. Scanln (&dat)
2	while dat != MARKER do	for dat != MARKER {
3	{kode untuk memproses dat}	// kode untuk memproses dat
4	input (dat)	fmt. Scanln (&dat)
5	endwhile	}

Apabila kita memperhatikan pola dengan marker di atas, terlihat bahwa selalu ada pengkondisian pada **while** yang akan selalu mengecek nilai *dat*. Semua nilai pada variabel *dat*, baik yang diperoleh pada baris ke-1 ataupun baris ke-4 akan selalu dicek pada baris ke-2. Apabila variabel *dat* tidak berisi *marker*, maka *dat* dapat diproses pada baris ke-3, sebaliknya apabila *dat* adalah *marker*, maka perulangan akan berhenti, dan *dat* tidak akan diproses pada baris ke-3.

Perhatikan contoh potongan program mencari nilai maksimum berikut ini.

	Notasi Algoritma	Notasi dalam bahasa Go
1	max = {BILANGAN_KECIL}	max = // BILANGAN_KECIL
2	input (dat)	fmt. Scan (&dat)
3	while dat != MARKER do	for dat != MARKER {
4	if dat > max then	if dat > max {
5	max = dat	max = dat
6	endif	}
7	input (dat)	fmt. Scan (&dat)
8	endwhile	}
9	output ("Data terbesar", max)	fmt. Println ("Data terbesar", max)

Nilai *marker* bisa nilai berapapun, biasanya diberikan pada soal atau kita biasanya bisa memberikan nilai berdasarkan asumsi.

Kemungkinan Rangkaian Data Kosong, Hanya Ada Marker

Pola dengan marker pada contoh di atas memungkinkan terjadi bahwa data pertama yang diberikan pada masukan adalah marker, artinya tidak ada satu datapun yang valid. Kemungkinan ini disebut juga rangkaian data kosong atau kasus kosong.

	Notasi Algoritma	Notasi dalam bahasa Go
1	input (dat)	fmt. Scanln (&dat)
2	if dat == MARKER then	if dat == MARKER {
3	{kode untuk data kosong}	// kode untuk data kosong
4	else	} else {
5	while dat != MARKER do	for dat != MARKER {
6	{kode untuk memproses dat}	// kode untuk memproses dat
7	input (dat)	fmt. Scanln (&dat)
8	endwhile	}
9	endif	}

Pada pola di atas ditambahkan struktur kontrol pengkondisian atau percabangan apabila ada kasus kosong yang terjadi. Berikut ini adalah contoh kode untuk mencari nilai maksimum.

	Notasi Algoritma	Notasi dalam bahasa Go
1	input (dat)	fmt. Scanln (&dat)
2	if dat == MARKER then	if dat == MARKER {
3	output ("tidak ada data")	fmt. Println ("tidak ada data")
4	else	} else {
5	max = {BILANGAN KECIL }	max = // BILANGAN KECIL
6	while dat != MARKER do	for dat != MARKER {
7	if dat > max then	if dat > max {
8	max	max = dat
9	endif	}
10	input (dat)	fmt. Scanln (&dat)
11	endwhile	}
12	output ("Data terbesar", max)	fmt. Println ("Data terbesar", max)
13	endif	}

Elemen Pertama Perlu Diproses Tersendiri/Kasus Khusus

Pada pola ini data pertama diproses terlebih dahulu secara khusus sebelum perulangan dilakukan. Apabila melihat contoh pencarian nilai maksimum di atas, terlihat bahwa nilai variabel *max* selalu diinisialisasi oleh sebuah nilai **BILANGAN KECIL** berapapun. Kekurangan dari pendekatan ini adalah kita harus mengetahui secara pasti nilai-nilai yang mungkin ada pada variabel *dat*, yang mana nilai pada variabel *dat* tersebut **TIDAK BOLEH** lebih kecil dibandingkan nilai dari **BILANGAN KECIL** yang digunakan saat inisialisasi. Sebagai contoh mencari nilai temperatur atau suhu maksimum. Pada kasus ini, berapa nilai dari **BILANGAN KECIL** yang akan digunakan? Tentu kita akan kesulitan menentukannya, karena temperatur bisa bernilai negatif. Oleh karena itu, dengan

menggunakan pola Kasus Khusus ini, penentuan **BILANGAN KECIL** tersebut data dapat terselesaikan.

	Notasi Algoritma	Notasi dalam bahasa Go
1	input (dat)	fmt. Scanln (&dat)
2	if dat == MARKER then	if dat == MARKER {
3	{kode untuk data kosong}	// kode untuk data kosong
4	else	}else{
5	{proses data pertama}	{proses data pertama}
6	input (dat)	fmt. Scan (&dat)
7	while dat != MARKER do	for dat != MARKER {
8	{kode untuk memproses dat}	// kode untuk memproses dat
9	input (dat)	fmt. Scanln (&dat)
10	endwhile	}
11	endif	}

Pada pola tersebut, terlihat data atau elemen pertama yang diperoleh pada baris ke-1 diproses pada baris ke-5, selanjutnya data ke-2 didapat pada baris ke-6.

Berikut contoh potongan program untuk mencari nilai maksimum:

	Notasi Algoritma	Notasi dalam bahasa Go
1	input (dat)	fmt. Scanln (&dat)
2	if dat == MARKER then	if dat == MARKER {
3	output ("tidak ada data")	fmt. Println ("tidak ada data")
4	else	}else{
5	max = dat	max = dat
6	input (dat)	fmt. Scan (&dat)
7	while dat != MARKER do	for dat != MARKER {
8	if dat > max then	if dat > max {
9	max	max = dat
10	endif	}
11	input (dat)	fmt. Scanln (&dat)
12	endwhile	}
13	output ("Data terbesar", max)	fmt. Println ("Data terbesar",max)
14	endif	}

Contoh-Contoh Program Sekuensial

Program menghitung luas persegi

```
package main

import "fmt"

func main() {
```

```
var sisi float64

fmt.Scanln(&sisi)

luas := sisi * sisi

fmt.Printf("Luas persegi: %.2f\n", luas)
}
```

Program Konversi Suhu

```
package main

import "fmt"

func main() {
    var celsius float64

    fmt.Scanln(&celsius)

    fahrenheit := (celsius * 9/5) + 32

    fmt.Printf("%.2f°C = %.2f°F\n", celsius, fahrenheit)
}
```

Kesimpulan

Pemrosesan sekuensial merupakan dasar dari pemrograman yang penting untuk dipahami sebelum mempelajari konsep yang lebih kompleks. Dalam Go, pemahaman tentang pemrosesan sekuensial membantu dalam membangun program yang terstruktur dan mudah dipahami.

LATIHAN SOAL

Latihan1

Diberikan sejumlah bilangan riil yang diakhiri dengan marker 9999, cari rerata dari bilangan-bilangan tersebut

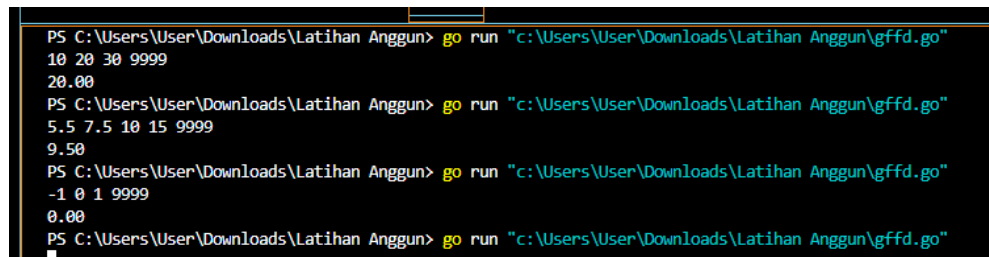
Source Code:

```
package main
import "fmt"
func main(){
    var bilangan float64
    total := 0.0
    jumlah := 0

    for {
        fmt.Scan(&bilangan)
        if bilangan == 9999 {
            break
        }
        total += bilangan
        jumlah++
    }

    if jumlah > 0 {
        rerata := total / float64(jumlah)
        fmt.Printf("%.2f\n", rerata)
    }
}
```

Output:



```
PS C:\Users\User\Downloads\Latihan Anggun> go run "c:\Users\User\Downloads\Latihan Anggun\gffd.go"
10 20 30 9999
20.00
PS C:\Users\User\Downloads\Latihan Anggun> go run "c:\Users\User\Downloads\Latihan Anggun\gffd.go"
5.5 7.5 10 15 9999
9.50
PS C:\Users\User\Downloads\Latihan Anggun> go run "c:\Users\User\Downloads\Latihan Anggun\gffd.go"
-1 0 1 9999
0.00
PS C:\Users\User\Downloads\Latihan Anggun> go run "c:\Users\User\Downloads\Latihan Anggun\gffd.go"
```

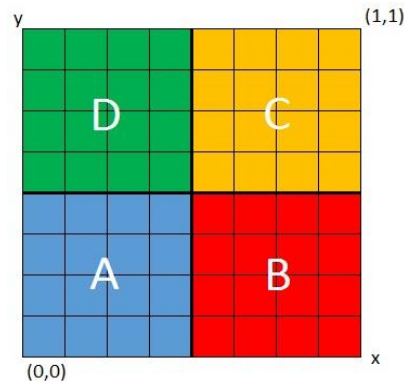
Deskripsi Program:

Saya mencoba memasukkan angka-angka diatas, maka program akan menghitung dan menampilkan rerata dari bilangan yang telah dimasukkan sebelum marker 9999.

Program yang ada di tabel source code diatas, digunakan untuk menghitung rerata dari sejumlah bilangan riil yang dimasukkan oleh pengguna. Pengguna dapat memasukkan bilangan satu per satu, dan proses input akan berlanjut hingga pengguna memasukkan angka 9999, yang berfungsi sebagai marker untuk menghentikan pengambilan input.

Latihan3

Empat daerah A, B, C, dan D yang berdekatan ingin mengukur curah hujan. Keempat daerah tersebut digambarkan pada bidang berikut:



Gambar 1. Ilustrasi denah daerah yang digunakan untuk mengukur curah hujan

Misal curah hujan dihitung berdasarkan banyaknya tetesan air hujan. Setiap tetesan berukuran 0.0001 ml curah hujan. Tetesan air hujan turun secara acak dari titik (0,0) sampai (1,1). Jika diterima input yang menyatakan banyaknya tetesan air hujan. Tentukan curah hujan untuk keempat daerah tersebut.

Buatlah program yang menerima input berupa banyaknya tetesan air hujan. Kemudian buat koordinat/titik (x, y) secara acak dengan menggunakan fungsi `rand.Float64()`. Hitung dan tampilkan banyaknya tetesan yang jatuh pada daerah A, B, C dan D. Konversikan satu tetesan berukuran 0.0001 milimeter.

Catatan: Lihat lampiran untuk informasi menggunakan paket `math/rand` untuk menggunakan `rand.Float64()` yang menghasilkan bilangan riil acak `[0..1]`.

Contoh masukan dan keluaran:

No	Masukan	Keluaran
1	10000000	Curah hujan daerah A: 250.0066 milimeter Curah hujan daerah B: 249.8981 milimeter Curah hujan daerah C: 249.9930 milimeter Curah hujan daerah D: 250.1023 milimeter

Source Code:

```
package main
import "fmt"
func main(){
```



```

rand.Seed(1) // Mengatur seed untuk hasil yang konsisten

var JmlTetes int
fmt.Scan(&JmlTetes)

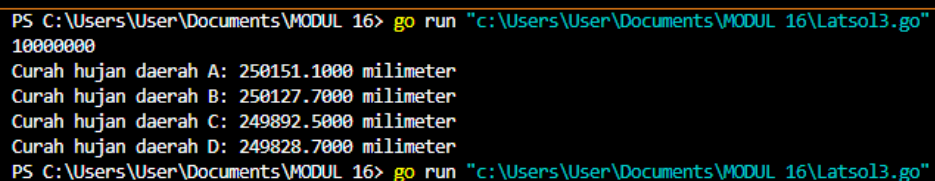
const UkTetes = 0.0001
hitung := [4]int{}

for i := 0; i < JmlTetes; i++ {
    x, y := rand.Float64(), rand.Float64()
    if x < 0.5 {
        if y < 0.5 {
            hitung[2]++ // Daerah C
        } else {
            hitung[0]++ // Daerah A
        }
    } else {
        if y < 0.5 {
            hitung[3]++ // Daerah D
        } else {
            hitung[1]++ // Daerah B
        }
    }
}

for i, c := range hitung {
    fmt.Printf("Curah hujan daerah %c: %.4f\n", 'A'+rune(i), float64(c)*UkTetes*1000)
}

```

Output:



```

PS C:\Users\User\Documents\MODUL 16> go run "c:\Users\User\Documents\MODUL 16\Latso13.go"
10000000
Curah hujan daerah A: 250151.1000 milimeter
Curah hujan daerah B: 250127.7000 milimeter
Curah hujan daerah C: 249892.5000 milimeter
Curah hujan daerah D: 249828.7000 milimeter
PS C:\Users\User\Documents\MODUL 16> go run "c:\Users\User\Documents\MODUL 16\Latso13.go"

```

Deskripsi Program:

Program Go ini untuk menghitung curah hujan di empat daerah yang berbeda berdasarkan jumlah tetesan air hujan yang jatuh. Program ini menggunakan library math/rand untuk mengacak koordinat (x,y) dari titik asal (0,0) hingga (1,1). Berdasarkan koordinat ini, program menentukan ke mana tetesan jatuh dan menghitung jumlah tetesan di masing-masing daerah.

Setelah proses simulasi selesai, program akan menampilkan hasil curah hujan untuk setiap daerah dalam satuan milimeter. Program ini sangat berguna dalam simulasi cuaca dan analisis statistik tentang distribusi hujan di wilayah-wilayah tertentu.

DAFTAR PUSTAKA

School of Computing. *Modul Praktikum 14 – Komposisi. Algoritma dan Pemrograman 1 SI Informatika*.2024