LAPORAN PRAKTIKUM
ALGORITMA PEMROGRAMAN 2
MODUL XI
PENCARIAN NILAI EKSTRIM PADA HIMPUNAN DATA

Oleh :

Geranada Saputra Priambudi

2311102008

IF-11-06

Dosen Pengampu :

Abednego Dwi Septiadi S.Kom., M.Kom.

PROGRAM STUDI S1 INFORMATIKA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

I. DASAR TEORI

Pencarian adalah suatu proses yang lazim dilakukan di dalam kehidupan sehari-hari. Contoh penggunaannya dalam kehidupan nyata sangatlah beragam, misalnya pencarian file di dalam directory komputer, pencarian suatu teks di dalam sebuah dokumen, pencarian buku pada rak buku, dan contoh lainnya.

Pencarian nilai ekstrem pada himpunan data adalah proses untuk menemukan nilai maksimum dan nilai minimum dalam sebuah data. Nilai maksimum adalah elemen terbesar dalam himpunan data, sedangkan nilai minimum adalah elemen terkecil. Operasi ini penting dalam banyak bidang seperti statistik, sains data, ekonomi, dan rekayasa perangkat lunak. Contoh aplikasinya adalah mencari suhu tertinggi dan terendah dalam data cuaca, atau keuntungan maksimum dan kerugian minimum dalam data keuangan.

Pada umumnya, himpunan data dapat berbentuk array, list, tabel, atau struktur data lainnya. Pencarian nilai ekstrem dilakukan dengan cara memeriksa setiap elemen dalam himpunan tersebut satu per satu. Proses ini dikenal sebagai iterasi, di mana algoritma membandingkan elemen saat ini dengan elemen sebelumnya untuk memperbarui nilai ekstrem sementara. Pendekatan ini sederhana namun sangat efektif, terutama untuk dataset yang berukuran kecil hingga menengah.

Proses pencarian nilai ekstrem memerlukan dua langkah utama. Pertama, algoritma harus memiliki asumsi awal, yaitu mengambil elemen pertama dari dataset sebagai nilai sementara untuk maksimum dan minimum. Kedua, algoritma akan memeriksa setiap elemen berikutnya, membandingkannya dengan nilai sementara. Jika elemen tersebut lebih besar dari nilai maksimum sementara, maka nilai maksimum diperbarui. Sebaliknya, jika elemen lebih kecil dari nilai minimum sementara, nilai minimum juga diperbarui. Proses ini terus berjalan hingga seluruh elemen selesai diperiksa.

Efisiensi algoritma pencarian nilai ekstrem bergantung pada jumlah elemen dalam himpunan data, sering disebut sebagai kompleksitas waktu. Algoritma ini memiliki kompleksitas waktu sebesar O(n), di mana n adalah jumlah elemen dalam dataset. Artinya, waktu yang dibutuhkan untuk menemukan nilai ekstrem akan meningkat secara linear dengan ukuran dataset. Hal ini membuat pendekatan ini efisien untuk dataset berukuran kecil atau sedang, tetapi mungkin kurang optimal untuk dataset yang sangat besar.

Selain itu, perlu dipahami bahwa pencarian nilai ekstrem hanya bekerja dengan baik pada dataset yang memiliki nilai-nilai yang terdefinisi. Jika dataset mengandung nilai-nilai yang tidak valid seperti data kosong (null) atau error, maka algoritma harus dilengkapi dengan mekanisme validasi data. Validasi ini bertujuan untuk memastikan bahwa setiap elemen yang diperiksa benar-benar dapat dibandingkan dengan elemen lainnya. Contohnya, dalam pengolahan data keuangan, nilai seperti "tidak ada data" harus diabaikan selama proses pencarian nilai ekstrem.

Salah satu tantangan dalam pencarian nilai ekstrem adalah menangani dataset dengan berbagai tipe data, seperti bilangan bulat (integer), bilangan desimal (float), atau data yang mengandung kombinasi keduanya. Dalam kasus seperti ini, dataset perlu diubah menjadi format yang seragam sebelum pencarian dilakukan. Misalnya, dalam analisis

suhu, data dalam format Celsius dan Fahrenheit harus dikonversi ke satu unit pengukuran yang sama untuk mendapatkan hasil yang konsisten.

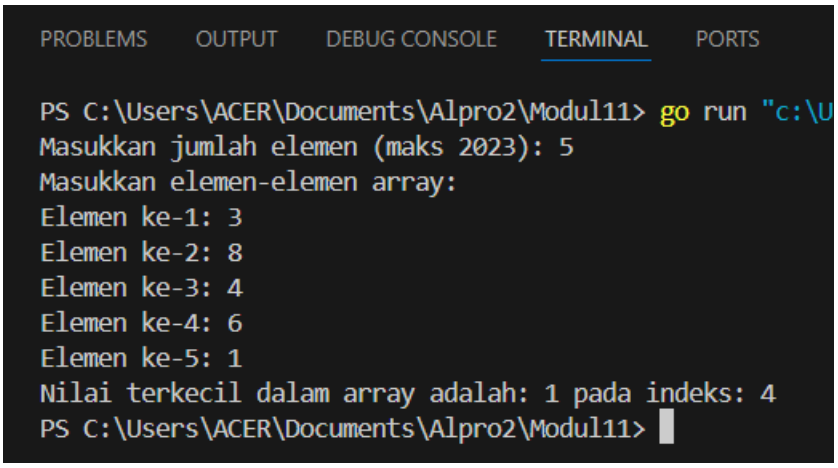
Dalam aplikasi praktis, pencarian nilai ekstrem tidak hanya memberikan informasi tentang nilai-nilai batas dalam dataset, tetapi juga membantu dalam mengidentifikasi anomali. Sebagai contoh, jika sebuah dataset suhu menunjukkan nilai maksimum yang jauh lebih tinggi dari nilai rata-rata historis, hal ini bisa menjadi indikasi adanya data outlier atau kejadian ekstrem dalam iklim. Dengan demikian, pencarian nilai ekstrem dapat membantu dalam deteksi dan analisis kejadian langka.

II. GUIDED

1. Source Code:

```
package main

import "fmt"

// Mendeklarasikan tipe data array arrInt dengan panjang 2023
type arrInt [2023]int

// Fungsi untuk mencari indeks elemen terkecil dalam array
func terkecil(tabInt arrInt, n int) int {
	var idx int = 0 // idx menyimpan indeks elemen terkecil
	var j int = 1
	for j < n {
		if tabInt[idx] > tabInt[j] {
			idx = j // Simpan indeks j jika elemen di indeks
j lebih kecil
		}
		j = j + 1
	}
	return idx
}

// Fungsi main untuk menguji fungsi terkecil
func main() {
	var n int
	var tab arrInt

	// Meminta input jumlah elemen array
	fmt.Print("Masukkan jumlah elemen (maks 2023): ")
	fmt.Scan(&n)

	// Validasi input jumlah elemen
	if n < 1 || n > 2023 {
		fmt.Println("Jumlah elemen harus antara 1 dan 2023.")
		return
	}

	// Memasukkan elemen-elemen array
	fmt.Println("Masukkan elemen-elemen array:")
	for i := 0; i < n; i++ {
		fmt.Print("Elemen ke-", i+1, ": ")
		fmt.Scan(&tab[i])
	}

	// Memanggil fungsi terkecil untuk menemukan indeks elemen
terkecil
	idxMin := terkecil(tab, n)

	// Menampilkan nilai dan indeks terkecil
	fmt.Println("Nilai terkecil dalam array adalah:",
tab[idxMin], "pada indeks:", idxMin)
}
```

Output:



Penjelasan:

Program di atas adalah implementasi pencarian elemen terkecil dalam array menggunakan bahasa Go. Program ini mendefinisikan array arrInt dengan kapasitas maksimum 2023 elemen dan menggunakan fungsi terkecil untuk mencari indeks elemen terkecil dalam array. Fungsi ini membandingkan setiap elemen dengan elemen lainnya, dimulai dari indeks ke-0 hingga indeks terakhir, dan mengembalikan indeks elemen dengan nilai terkecil. Dalam fungsi main, pengguna diminta untuk memasukkan jumlah elemen array (n) dan kemudian mengisi nilai-nilai elemen array tersebut. Setelah semua elemen dimasukkan, program memanggil fungsi terkecil dan mencetak nilai terkecil serta indeksnya. Pada contoh eksekusi, ketika pengguna memasukkan lima elemen (3, 8, 4, 6, 1), program menentukan bahwa nilai terkecil adalah 1, yang berada pada indeks ke-4 (menggunakan indeks berbasis nol). Hasil ini kemudian ditampilkan kepada pengguna.

2. **Source Code:**

```
package main

import "fmt"

// Definisi struct mahasiswa dengan atribut nama, nim, kelas,
jurusan, dan ipk
type mahasiswa struct {
      nama, nim, kelas, jurusan string
      ipk                       float64
}

// Definisi tipe data array mahasiswa dengan kapasitas maksimal
2023
type arrMhs [2023]mahasiswa

// Fungsi untuk mencari IPK tertinggi dalam array mahasiswa
func ipk(T arrMhs, n int) float64 {
```

```
	var tertinggi float64 = T[0].ipk
	var j int = 1
	for j < n {
		if tertinggi < T[j].ipk {
			tertinggi = T[j].ipk
		}
		j = j + 1
	}
	return tertinggi
}

// Fungsi main untuk mengisi data mahasiswa dan mencari IPK tertinggi
func main() {
	var n int
	var dataMhs arrMhs

	// Meminta input jumlah mahasiswa
	fmt.Print("Masukkan jumlah mahasiswa (maks 2023): ")
	fmt.Scan(&n)

	// Validasi jumlah mahasiswa yang dimasukkan
	if n < 1 || n > 2023 {
		fmt.Println("Jumlah mahasiswa harus antara 1 dan 2023.")
		return
	}

	// Mengisi data mahasiswa
	for i := 0; i < n; i++ {
		fmt.Printf("\nMasukkan data mahasiswa ke-%d\n", i+1)
		fmt.Print("Nama: ")
		fmt.Scan(&dataMhs[i].nama)
		fmt.Print("NIM: ")
		fmt.Scan(&dataMhs[i].nim)
		fmt.Print("Kelas: ")
		fmt.Scan(&dataMhs[i].kelas)
		fmt.Print("Jurusan: ")
		fmt.Scan(&dataMhs[i].jurusan)
		fmt.Print("IPK: ")
		fmt.Scan(&dataMhs[i].ipk)
	}

	// Mencari dan menampilkan IPK tertinggi
	tertinggi := ipk(dataMhs, n)
	fmt.Printf("\nIPK tertinggi dari %d mahasiswa adalah: %.2f\n", n, tertinggi)

}
```

Output:

```
PS C:\Users\ACER\Documents\Alpro2\Modul11> go run "c:\Users\ACER\
Masukkan jumlah mahasiswa (maks 2023): 5

Masukkan data mahasiswa ke-1
Nama: Gery
NIM: 2234
Kelas: A
Jurusan: Informatika
IPK: 4

Masukkan data mahasiswa ke-2
Nama: Nada
NIM: 2235
Kelas: B
Jurusan: RPL
IPK: 3.6

Masukkan data mahasiswa ke-3
Nama: Jaki
NIM: 2237
Kelas: B
Jurusan: Biomedis
IPK: 3.8

Masukkan data mahasiswa ke-4
Nama: Danu
NIM: 2239
Kelas: A
Jurusan: Informatika
IPK: 4

Masukkan data mahasiswa ke-5
Nama: Budi
NIM: 2242
Kelas: B
Jurusan: Elektro
IPK: 3.7

IPK tertinggi dari 5 mahasiswa adalah: 4.00
```

Penjelasan:

Program di atas merupakan program Go yang digunakan untuk mencari IPK tertinggi dari sejumlah mahasiswa yang datanya dimasukkan oleh pengguna. Program ini mendefinisikan sebuah struct mahasiswa yang berisi atribut nama, nim, kelas, jurusan, dan ipk. Kemudian, program menggunakan array arrMhs untuk menyimpan data mahasiswa, dengan kapasitas maksimal 2023 mahasiswa. Pengguna diminta untuk memasukkan jumlah mahasiswa yang ingin diinput, lalu diikuti dengan input data masing-masing mahasiswa. Setelah itu, fungsi ipk digunakan untuk mencari IPK tertinggi dengan membandingkan setiap nilai IPK

mahasiswa yang dimasukkan. Output dari program ini menampilkan IPK tertinggi dari mahasiswa yang diinput. Dalam contoh, input yang diberikan menghasilkan output "IPK tertinggi dari 5 mahasiswa adalah: 4.00", karena mahasiswa dengan nama Gery dan Danu memiliki IPK tertinggi yaitu 4.00.

III. UNGUIDED

1. Sebuah program digunakan untuk mendata berat anak kelinci yang akan dijual ke pasar. Program ini menggunakan array dengan kapasitas 1000 untuk menampung data berat anak kelinci yang akan dijual.

   Masukan terdiri dari sekumpulan bilangan, yang mana bilangan pertama adalah bilangan bulat N yang menyatakan banyaknya anak kelinci yang akan ditimbang beratnya. Selanjutnya N bilangan riil berikutnya adalah berat dari anak kelinci yang akan dijual.

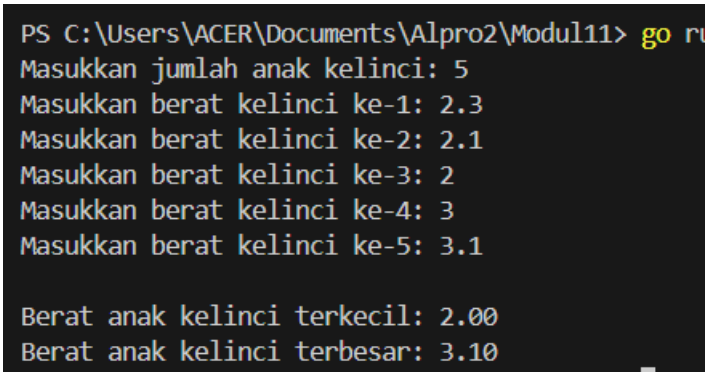
   Keluaran terdiri dari dua buah bilangan riil yang menyatakan berat kelinci terkecil dan terbesar.

   Source Code

```
package main

import "fmt"

type arrInt [1000]float64

func nilaiEkstrim(berat arrInt, jumlahKelinci int) (float64,
float64) {
	nilaiMin := berat[0]
	nilaiMax := berat[0]

	for i := 1; i < jumlahKelinci; i++ {
		if berat[i] < nilaiMin {
			nilaiMin = berat[i]
		}
		if berat[i] > nilaiMax {
			nilaiMax = berat[i]
		}
	}
	return nilaiMin, nilaiMax
}

func main() {
	var jumlahKelinci int
	var beratKelinci arrInt

	fmt.Print("Masukkan jumlah anak kelinci: ")
	fmt.Scan(&jumlahKelinci)

	if jumlahKelinci < 1 || jumlahKelinci > 1000 {
		fmt.Println("Masukan harus angka 1-1000!")
		return
	}

	for i := 0; i < jumlahKelinci; i++ {
		fmt.Printf("Masukkan berat kelinci ke-%d: ", i+1)
		fmt.Scan(&beratKelinci[i])
	}
```

```
	nilaiMin, nilaiMax := nilaiEkstrim(beratKelinci, jumlahKelinci)

	fmt.Printf("\nBerat anak kelinci terkecil: %.2f\n", nilaiMin)
	fmt.Printf("Berat anak kelinci terbesar: %.2f\n", nilaiMax)
}
```

Output



Penjelasan Program

Program di atas adalah implementasi dalam bahasa Go yang digunakan untuk mencari berat anak kelinci terkecil dan terbesar dari sejumlah input yang diberikan oleh pengguna. Program ini dimulai dengan mendeklarasikan tipe data arrInt sebagai array dengan kapasitas 1000 elemen bertipe float64, yang menyimpan berat kelinci. Fungsi nilaiEkstrim digunakan untuk mencari nilai minimum dan maksimum dari array berat kelinci berdasarkan jumlah kelinci yang dimasukkan oleh pengguna. Di dalam fungsi main, pengguna diminta untuk memasukkan jumlah anak kelinci, yang kemudian akan divalidasi agar berada dalam rentang 1 hingga 1000. Setelah itu, program meminta input berat kelinci satu per satu dan menyimpannya dalam array. Fungsi nilaiEkstrim kemudian dipanggil untuk menghitung berat kelinci terkecil dan terbesar, dan hasilnya ditampilkan dengan format dua angka di belakang koma. Output yang dihasilkan menunjukkan bahwa berat kelinci terkecil adalah 2.00 kg dan berat kelinci terbesar adalah 3.10 kg, sesuai dengan input yang diberikan oleh pengguna.

2. Sebuah program digunakan untuk menentukan tarif ikan yang akan dijual ke pasar. Program ini menggunakan array dengan kapasitas 1000 untuk menampung data berat ikan yang akan dijual.
   Masukan terdiri dari dua baris, yang mana baris pertama terdiri dari dua bilangan bulat x dan y. Bilangan x menyatakan banyaknya ikan yang akan dijual, sedangkan

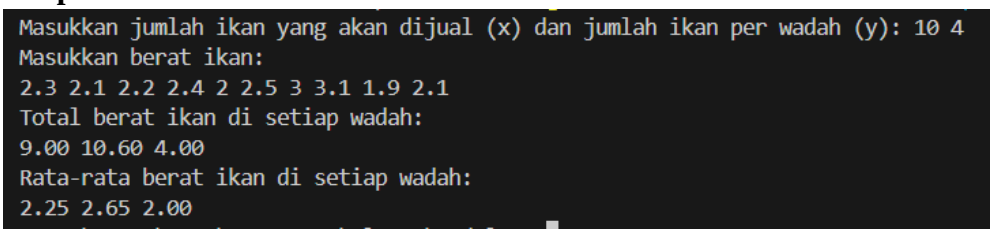
y adalah banyaknya ikan yang akan dimasukan ke dalam wadah. Baris kedua terdiri dari sejumlah x bilangan riil yang menyatakan banyaknya ikan yang akan dijual.
Keluaran terdiri dari dua baris. Baris pertama adalah kumpulan bilangan riil yang menyatakan total berat ikan di setiap wadah (jumlah wadah tergantung pada nilai x dan y, urutan ikan yang dimasukan ke dalam wadah sesuai urutan pada masukan baris ke-2). Baris kedua adalah sebuah bilangan riil yang menyatakan berat rata-rata ikan di setiap wadah.

Source Code

```
package main

import "fmt"

type Ikan struct {
    Berat float64
}

type Wadah struct {
    TotalBerat float64
    RataRata   float64
}

type arrIkan [1000]Ikan

func main() {
    var x, y int
     fmt.Print("Masukkan jumlah ikan yang akan dijual (x) dan
jumlah ikan per wadah (y): ")
    fmt.Scan(&x, &y)

    if x > 1000 {
        fmt.Println("Jumlah ikan melebihi kapasitas array.")
        return
    }
    var ikanArray arrIkan

    fmt.Println("Masukkan berat ikan:")
    for i := 0; i < x; i++ {
        fmt.Scan(&ikanArray[i].Berat)
    }

    jumlahWadah := x / y
    if x%y != 0 {
        jumlahWadah++
    }

    wadahArray := make([]Wadah, jumlahWadah)

    for i := 0; i < x; i++ {
        indexWadah := i / y
        wadahArray[indexWadah].TotalBerat += ikanArray[i].Berat
    }
```

```
    fmt.Println("Total berat ikan di setiap wadah:")
    for _, wadah := range wadahArray {
        fmt.Printf("%.2f ", wadah.TotalBerat)
    }
    fmt.Println()

    fmt.Println("Rata-rata berat ikan di setiap wadah:")
    for i := 0; i < jumlahWadah; i++ {
        var jumlahIkanPerWadah int
        if i == jumlahWadah-1 {
            jumlahIkanPerWadah = x % y
            if jumlahIkanPerWadah == 0 && x > 0 {
                jumlahIkanPerWadah = y
            }
        } else {
            jumlahIkanPerWadah = y
        }

           wadahArray[i].RataRata  =  wadahArray[i].TotalBerat  /
float64(jumlahIkanPerWadah)
        fmt.Printf("%.2f ", wadahArray[i].RataRata)
    }
    fmt.Println()
}
```

Output



Penjelasan Program

Program di atas adalah program bahasa Go untuk menghitung total berat ikan di setiap wadah dan rata-rata berat ikan di setiap wadah berdasarkan input jumlah ikan dan berat ikan. Pertama, program meminta input dua nilai, yaitu jumlah ikan yang akan dijual (x) dan jumlah ikan yang akan dimasukkan ke dalam setiap wadah (y). Setelah itu, program meminta input berat masing-masing ikan. Kemudian, program menghitung jumlah wadah yang diperlukan dengan cara membagi jumlah ikan dengan kapasitas wadah dan membulatkan hasilnya jika ada sisa ikan. Selanjutnya, program mengalokasikan ikan ke dalam wadah sesuai urutan dan menghitung total berat ikan di setiap wadah. Untuk setiap wadah, program juga menghitung rata-rata berat ikan di dalamnya dengan membagi total berat ikan dengan jumlah ikan dalam wadah tersebut. Output program terdiri dari dua baris: baris pertama menampilkan total berat ikan di setiap wadah, dan baris kedua menampilkan rata-rata berat ikan di setiap wadah.

Sebagai contoh, jika input jumlah ikan 10 dan kapasitas wadah 4 dengan berat ikan yang dimasukkan seperti yang tercantum, outputnya adalah total berat ikan per wadah: 9.00 10.60 4.00 dan rata-rata berat ikan per wadah: 2.25 2.65 2.00. Penjelasan output program ini dimulai dari pembagian jumlah ikan (x = 10) dengan kapasitas wadah (y = 4) untuk menentukan jumlah wadah yang diperlukan. Karena 10 dibagi 4 menghasilkan 2 wadah penuh dan 2 ikan sisa, maka diperlukan 3 wadah. Ikan-ikan dimasukkan ke dalam wadah sesuai urutan, di mana wadah pertama berisi ikan-ikan dengan berat 2.3, 2.1, 2.2, dan 2.4, sehingga total berat wadah pertama adalah 9.00. Wadah kedua berisi ikan dengan berat 2, 2.5, 3, dan 3.1, yang menghasilkan total berat 10.60. Wadah ketiga berisi dua ikan terakhir dengan berat 1.9 dan 2.1, yang total beratnya 4.00. Rata-rata berat ikan di setiap wadah dihitung dengan membagi total berat wadah dengan jumlah ikan di wadah tersebut. Untuk wadah pertama, rata-ratanya adalah 9.00 / 4 = 2.25, untuk wadah kedua adalah 10.60 / 4 = 2.65, dan untuk wadah ketiga adalah 4.00 / 2 = 2.00.

3. Pos Pelayanan Terpadu (posyandu) sebagai tempat pelayanan kesehatan perlu mencatat data berat balita (dalam kg). Petugas akan memasukkan data tersebut ke dalam array. Dari data yang diperoleh akan dicari berat balita terkecil, terbesar, dan reratanya.
   Buatlah program dengan spesifikasi subprogram sebagai berikut:

```
type arrBalita [100]float64
func hitungMinMax(arrBerat arrBalita; bMin, bMax *float64) {
/* I.S. Terdefinisi array dinamis arrBerat
 Proses: Menghitung berat minimum dan maksimum dalam array
 F.S. Menampilkan berat minimum dan maksimum balita */
 ...
}
function rerata (arrBerat arrBalita) real {
/* menghitung dan mengembalikan rerata berat balita dalam array
*/
 ...
}
```

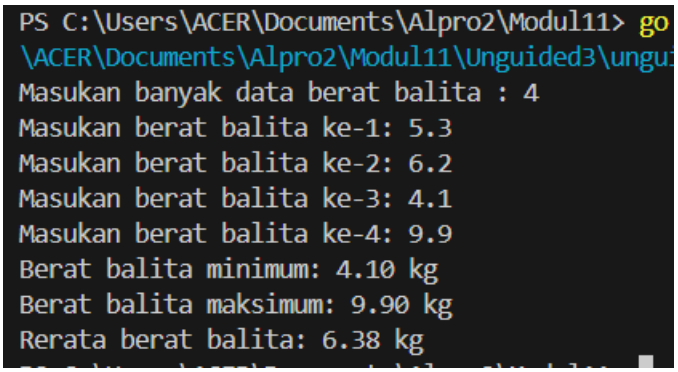
   Perhatikan sesi interaksi pada contoh berikut ini **(teks bergaris bawah adalah input/read)**

```
Masukan banyak data berat balita : 4
Masukan berat balita ke-1: 5.3
Masukan berat balita ke-2: 6.2
Masukan berat balita ke-3: 4.1
Masukan berat balita ke-4: 9.9
Berat balita minimum: 4.10 kg
Berat balita maksimum: 9.90 kg
Rerata berat balita: 6.38 kg
```

Source Code

```
package main

import "fmt"

type arrBalita [100]float64

func hitungMinMax(arrBerat arrBalita, jumlahBalita int, bMin,
bMax *float64) {
    *bMin = arrBerat[0]
    *bMax = arrBerat[0]
    for i := 0; i < jumlahBalita; i++ {
        if arrBerat[i] < *bMin {
            *bMin = arrBerat[i]
        }
        if arrBerat[i] > *bMax {
            *bMax = arrBerat[i]
        }
    }
}

func rerata(arrBerat arrBalita, jumlahBalita int) float64 {
    var total float64
    for n := 0; n < jumlahBalita; n++ {
        total += arrBerat[n]
    }
    return total / float64(jumlahBalita)
}

func main() {
    var jumlahBalita int
    fmt.Print("Masukan banyak data berat balita : ")
    fmt.Scan(&jumlahBalita)
    var arrBerat arrBalita
    for n := 0; n < jumlahBalita; n++ {
        fmt.Printf("Masukan berat balita ke-%d: ", n+1)
        fmt.Scan(&arrBerat[n])
    }

    var bMin, bMax float64
    hitungMinMax(arrBerat, jumlahBalita, &bMin, &bMax)
    rerataBerat := rerata(arrBerat, jumlahBalita)
    fmt.Printf("Berat balita minimum: %.2f kg\n", bMin)
    fmt.Printf("Berat balita maksimum: %.2f kg\n", bMax)
    fmt.Printf("Rerata berat balita: %.2f kg\n", rerataBerat)
}
```

Output



Penjelasan Program

Program Go diatas untuk mencatat dan menganalisis data berat badan balita di Posyandu. Data berat badan dimasukkan ke dalam array bertipe arrBalita yang mampu menampung hingga 100 data. Setelah pengguna menentukan jumlah data balita yang akan diinput, program meminta pengguna memasukkan berat badan masing-masing balita satu per satu. Fungsi hitungMinMax digunakan untuk menghitung berat minimum dan maksimum dengan membandingkan setiap elemen dalam array terhadap nilai awal minimum dan maksimum yang diinisialisasi dengan elemen pertama. Sementara itu, fungsi rerata menghitung rata-rata berat badan dengan menjumlahkan semua data berat badan yang ada dan membaginya dengan jumlah data yang dimasukkan. Hasil akhir berupa berat balita minimum, maksimum, dan rata-rata ditampilkan dalam format dua desimal untuk mempermudah pembacaan. Pada contoh input 4 data berat balita (5.3 kg, 6.2 kg, 4.1 kg, 9.9 kg), program membandingkan setiap nilai untuk menentukan bahwa 4.1 adalah berat terkecil dan 9.9 adalah terbesar. Rata-rata dihitung dengan rumus (5.3 + 6.2 + 4.1 + 9.9) / 4 = 6.38, sehingga menghasilkan output yang sesuai: minimum 4.10 kg, maksimum 9.90 kg, dan rata-rata 6.38 kg.