

**LAPORAN PRAKTIKUM
ALGORITMA DAN PEMROGRAMAN 2**

**MODUL XI
NILAI EKSTREM**



Disusun Oleh :

Egi Umar Ferdhika / 2311102277

11-IF-06

Dosen Pengampu :

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

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. DASAR TEORI

Nilai ekstrem dalam pemrograman merujuk pada nilai terkecil (minimum) atau terbesar (maksimum) dalam suatu kumpulan data. Pencarian nilai ekstrem digunakan dalam berbagai aplikasi seperti analisis statistik, validasi data, dan optimasi. Proses ini dimulai dengan menginisialisasi nilai pertama dari data sebagai nilai sementara untuk minimum dan maksimum. Kemudian, data diiterasi untuk membandingkan setiap elemen dengan nilai sementara tersebut. Jika ditemukan elemen yang lebih kecil dari nilai minimum atau lebih besar dari nilai maksimum, nilai tersebut akan diperbarui. Algoritma pencarian nilai ekstrem di Go memiliki kompleksitas waktu $O(n)$, yang berarti pencarian dilakukan hanya dengan satu kali iterasi terhadap data.

Menggunakan struktur data seperti array dan slice untuk menyimpan data, memungkinkan pencarian nilai ekstrem dengan efisien. Fungsi pencarian nilai ekstrem dapat dibuat modular, yang meningkatkan keterbacaan dan pemeliharaan kode. Selain itu, Go menyediakan pengecekan indeks otomatis yang mengurangi kesalahan dalam akses data. Pencarian nilai ekstrem berguna dalam berbagai konteks, seperti pengolahan data sensor, analisis rentang data, dan optimasi. Secara keseluruhan,

II. GUIDED

1. Berisi source code dan output dari kegiatan praktikum yang telah dilaksanakan.

Source Code diberi penjelasan maka akan menjadi nilai ++

GUIDED 1

Sourcecode

```
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
    }

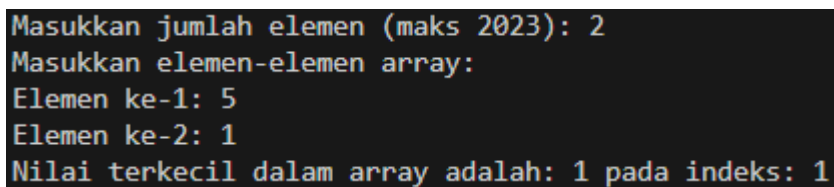
    // Memasukkan elemen-elemen array
    fmt.Println("Masukkan elemen-elemen array:")
    for i := 0; i < n; i++ {
        fmt.Print("Elemen ke-", i+1, ": ")
        fmt.Scan(&tab[i])
    }
}
```

```
}

// 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)
}
```

Screenshoot Output



```
Masukkan jumlah elemen (maks 2023): 2
Masukkan elemen-elemen array:
Elemen ke-1: 5
Elemen ke-2: 1
Nilai terkecil dalam array adalah: 1 pada indeks: 1
```

Deskripsi Program

Kode di atas adalah kode yang bertujuan untuk menemukan nilai terkecil dalam sebuah array beserta indeksinya. Program dimulai dengan mendeklarasikan tipe data `arrInt`, yaitu array tetap dengan panjang maksimum 2023 elemen bertipe integer. Fungsi `terkecil` menerima array tersebut dan jumlah elemen yang akan diproses, kemudian mengembalikan indeks elemen dengan nilai terkecil menggunakan perulangan sederhana. Fungsi ini bekerja dengan membandingkan nilai elemen array dari indeks 1 hingga $n-1$, lalu memperbarui indeks terkecil jika ditemukan elemen yang lebih kecil.

Pada fungsi `main`, program meminta pengguna memasukkan jumlah elemen array, yang harus berada dalam rentang 1 hingga 2023, untuk memastikan validitas input. Setelah jumlah elemen diterima, program meminta pengguna mengisi setiap elemen array satu per satu. Selanjutnya, fungsi `terkecil` dipanggil untuk menemukan indeks elemen terkecil, dan hasilnya (nilai terkecil beserta indeksinya) ditampilkan ke layar. Program ini sederhana, efisien, dan cocok untuk pemrosesan data dalam array yang jumlah elemennya dibatasi. Namun, penggunaan array tetap dapat dimodifikasi menjadi array dinamis (`slice`) untuk efisiensi memori, dan validasi input dapat ditingkatkan untuk menangani kasus data yang tidak valid.

GUIDED 2

Sourcecode

```
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)

}

```

Screenshoot Output

```

Masukkan jumlah mahasiswa (maks 2023): 2

Masukkan data mahasiswa ke-1
Nama: Jono
NIM: 22
Kelas: 4
Jurusan: Akuntansi
IPK: 4.0

Masukkan data mahasiswa ke-2
Nama: Fikri
NIM: 24
Kelas: 5
Jurusan: Informatika
IPK: 3.99

IPK tertinggi dari 2 mahasiswa adalah: 4.00

```

Deskripsi Program

Kode di atas adalah program untuk mencari IPK tertinggi dari data sekelompok mahasiswa. Data mahasiswa didefinisikan dalam bentuk struct bernama mahasiswa, yang memiliki atribut seperti nama, nim, kelas, jurusan, dan ipk. Program juga mendeklarasikan tipe data array arrMhs, yang mampu menyimpan hingga 2023 data mahasiswa. Fungsi ipk digunakan untuk mencari nilai IPK tertinggi dalam array mahasiswa dengan membandingkan atribut ipk setiap elemen array menggunakan perulangan. Pada fungsi main, program meminta pengguna memasukkan jumlah mahasiswa (dengan validasi bahwa jumlah harus antara 1

hingga 2023), lalu data mahasiswa diinput satu per satu melalui atributnya. Setelah data terkumpul, fungsi ipk dipanggil untuk menghitung IPK tertinggi, yang kemudian ditampilkan dalam format dua angka desimal. Program ini efektif untuk mengolah data mahasiswa dengan fokus pada atribut IPK, meskipun penggunaan array tetap dapat diganti dengan slice agar lebih fleksibel dalam pengelolaan memori.

III. UNGUIDED

1. Berisi source code dan output dari kegiatan praktikum yang telah dilaksanakan.

Source Code diberi penjelasan maka akan menjadi nilai ++

UNGUIDED 1

Sourcecode

```
package main

import (
    "fmt"
    // Nama : Egi Umar Ferdhika
    // NIM : 2311102277
)

func main() {
    var beratKelinci [1000]float64
    var N int

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

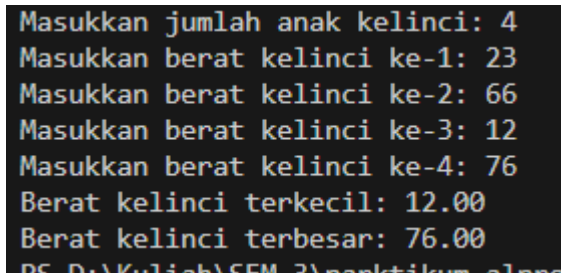
    for i := 0; i < N; i++ {
        fmt.Printf("Masukkan berat kelinci ke-%d: ",
i+1)
        fmt.Scan(&beratKelinci[i])
    }

    min := beratKelinci[0]
    max := beratKelinci[0]

    for i := 1; i < N; i++ {
        if beratKelinci[i] < min {
            min = beratKelinci[i]
        }
        if beratKelinci[i] > max {
            max = beratKelinci[i]
        }
    }
}
```

```
    fmt.Printf("Berat kelinci terkecil: %.2f\n", min)
    fmt.Printf("Berat kelinci terbesar: %.2f\n", max)
}
```

Screenshoot Output



```
Masukkan jumlah anak kelinci: 4
Masukkan berat kelinci ke-1: 23
Masukkan berat kelinci ke-2: 66
Masukkan berat kelinci ke-3: 12
Masukkan berat kelinci ke-4: 76
Berat kelinci terkecil: 12.00
Berat kelinci terbesar: 76.00
```

Deskripsi Program

Program di atas menggunakan bahasa pemrograman Go untuk menentukan berat terkecil dan terbesar dari sejumlah kelinci. Program dimulai dengan mendeklarasikan array `beratKelinci` berukuran maksimum 1000 elemen bertipe `float64`, yang digunakan untuk menyimpan berat kelinci. Pengguna diminta memasukkan jumlah kelinci (N) dan berat masing-masing kelinci satu per satu. Setelah semua data berat dimasukkan, program melakukan pencarian berat terkecil (min) dan terbesar (max) menggunakan perulangan. Nilai awal min dan max diinisialisasi dengan berat kelinci pertama, kemudian dibandingkan dengan berat kelinci lainnya. Jika ditemukan berat yang lebih kecil dari min atau lebih besar dari max, nilai tersebut diperbarui. Hasil akhirnya, yaitu berat terkecil dan terbesar, ditampilkan dalam format desimal dua angka. Program ini sederhana dan efektif untuk mengolah data berat kelinci dalam jumlah yang tidak melebihi 1000, meskipun dapat dioptimalkan dengan penggunaan array dinamis (slice) agar lebih fleksibel.

UNGUIDED 2

Sourcecode

```
package main

import (
    "fmt"
    "math"

    // Nama : Egi Umar Ferdhika
    // NIM : 2311102277
)
```



```

)

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

    fmt.Println("Inputkan berat ikan:")
    beratIkan := inputBerat(jumlahIkan)

    totalWadah := hitungWadah(jumlahIkan,
jumlahPerWadah)
    totalPerWadah := hitungTotalPerWadah(beratIkan,
jumlahPerWadah, totalWadah)
    rataRataPerWadah := hitungRataRata(totalPerWadah,
jumlahIkan, jumlahPerWadah)

    tampilkanHasil("Total ikan di wadah:",
totalPerWadah)
    tampilkanHasil("Rata-rata berat ikan di setiap
wadah:", rataRataPerWadah)
}

func inputBerat(jumlah int) []float64 {
    berat := make([]float64, jumlah)
    for i := 0; i < jumlah; i++ {
        fmt.Scan(&berat[i])
    }
    return berat
}

func hitungWadah(jumlahIkan, jumlahPerWadah int) int {
    return int(math.Ceil(float64(jumlahIkan) /
float64(jumlahPerWadah)))
}

func hitungTotalPerWadah(berat []float64,
jumlahPerWadah, totalWadah int) []float64 {
    totals := make([]float64, totalWadah)
    for i, b := range berat {
        indexWadah := i / jumlahPerWadah
        totals[indexWadah] += b
    }
    return totals
}

func hitungRataRata(totalPerWadah []float64, jumlahIkan,
jumlahPerWadah int) []float64 {
    rataRata := make([]float64, len(totalPerWadah))
    for i := range totalPerWadah {
        if (i+1)*jumlahPerWadah <= jumlahIkan {
            rataRata[i] = totalPerWadah[i] /
float64(jumlahPerWadah)
        } else {

```

```

        rataRata[i] = totalPerWadah[i] /
float64(jumlahIkan%jumlahPerWadah)
    }
    }
    return rataRata
}

func tampilkanHasil(label string, data []float64) {
    fmt.Println(label)
    for _, value := range data {
        fmt.Printf("%.2f ", value)
    }
    fmt.Println()
}

```

Screenshoot Output

```

Inputkan jumlah ikan yang mau dijual (x) dan jumlah ikan per wadah (y): 5 3
Inputkan berat ikan:
23
44
21
65
71
Total ikan di wadah:
88.00 136.00
Rata-rata berat ikan di setiap wadah:
29.33 68.00

```

Deskripsi Program

Program ini digunakan untuk menghitung total berat ikan per wadah dan rata-rata berat ikan di setiap wadah berdasarkan jumlah ikan yang akan dijual dan kapasitas maksimum ikan per wadah. Program dimulai dengan meminta pengguna untuk memasukkan jumlah ikan (jumlahIkan) dan jumlah ikan per wadah (jumlahPerWadah), diikuti dengan data berat masing-masing ikan. Berat ikan disimpan dalam slice menggunakan fungsi inputBerat. Program kemudian menghitung jumlah wadah yang diperlukan menggunakan fungsi hitungWadah, yang memanfaatkan fungsi math.Ceil untuk membulatkan ke atas jika jumlah ikan tidak habis dibagi kapasitas per wadah. Fungsi hitungTotalPerWadah menghitung total berat ikan untuk setiap wadah dengan mengelompokkan berat ikan berdasarkan indeks. Rata-rata berat ikan di setiap wadah dihitung oleh fungsi hitungRataRata, dengan mempertimbangkan ikan yang tersisa jika jumlah ikan tidak habis dibagi rata. Akhirnya, hasil total berat dan rata-rata berat per wadah ditampilkan melalui fungsi tampilkanHasil. Program ini dirancang modular dengan

fungsi-fungsi terpisah yang meningkatkan keterbacaan dan kemudahan pengelolaan kode, menjadikannya fleksibel untuk analisis distribusi berat ikan.

UNGUIDED 3

Sourcecode

```
package main

import (
    "fmt"
    // Nama : Egi Umar Ferdhika
    // NIM : 2311102277
)

type arrBalita [100]float64

func hitungMinMax(arrBerat arrBalita, n int) (min, max float64) {
    min = arrBerat[0]
    max = arrBerat[0]

    for i := 1; i < n; i++ {
        if arrBerat[i] < min {
            min = arrBerat[i]
        }
        if arrBerat[i] > max {
            max = arrBerat[i]
        }
    }
    return
}

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

func main() {
    var dataBalita arrBalita
    var n int

    fmt.Print("Masukkan banyak data berat balita: ")
    fmt.Scan(&n)

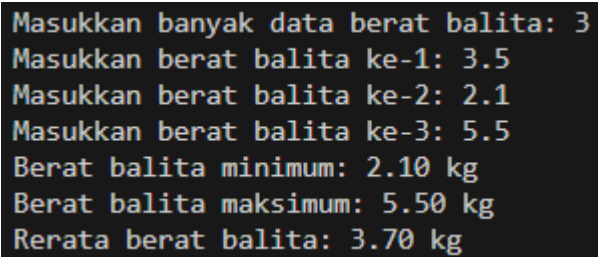
    for i := 0; i < n; i++ {
        fmt.Printf("Masukkan berat balita ke-%d: ", i+1)
        fmt.Scan(&dataBalita[i])
    }

    min, max := hitungMinMax(dataBalita, n)
    rata := rerata(dataBalita, n)

    fmt.Printf("Berat balita minimum: %.2f kg\n", min)
```

```
    fmt.Printf("Berat balita maksimum: %.2f kg\n", max)
    fmt.Printf("Rerata berat balita: %.2f kg\n", rata)
}
```

Screenshoot Output



```
Masukkan banyak data berat balita: 3
Masukkan berat balita ke-1: 3.5
Masukkan berat balita ke-2: 2.1
Masukkan berat balita ke-3: 5.5
Berat balita minimum: 2.10 kg
Berat balita maksimum: 5.50 kg
Rerata berat balita: 3.70 kg
```

Deskripsi Program

Program di atas bertujuan untuk menghitung berat minimum, berat maksimum, dan rata-rata berat balita berdasarkan data yang dimasukkan oleh pengguna. Data berat balita disimpan dalam array `arrBalita` yang mampu menampung hingga 100 elemen bertipe `float`. Fungsi `hitungMinMax` digunakan untuk mencari berat terkecil (minimum) dan terbesar (maksimum) dengan membandingkan setiap elemen array secara iteratif. Sementara itu, fungsi `rerata` menghitung rata-rata berat balita dengan menjumlahkan seluruh nilai dalam array kemudian membaginya dengan jumlah data (`n`). Pada fungsi `main`, program meminta pengguna memasukkan jumlah data berat balita, kemudian berat masing-masing balita. Setelah data dimasukkan, fungsi `hitungMinMax` dan `rerata` dipanggil untuk menghitung hasilnya, yang kemudian ditampilkan dalam format dua desimal. Program ini dirancang sederhana dan efektif untuk menganalisis berat balita, tetapi dapat disesuaikan lebih lanjut dengan penggunaan array dinamis (`slice`) untuk menangani jumlah data yang lebih fleksibel.