

1 Pengolahan Data

```
package main

import (
    "encoding/json"
    "html/template"
    "log"
    "net/http"
)

type Country struct {
    Name      string `json:"name"`
    DialCode  string `json:"dialCode"`
    IsoCode   string `json:"isoCode"`
    Flag      string `json:"flag"`
}

type TemplateData struct {
    Countries []Country
}

func main() {
    http.HandleFunc("/", countriesHandler)
    log.Println("Server started at http://localhost:8080")
    log.Fatal(http.ListenAndServe(":8080", nil))
}

func countriesHandler(w http.ResponseWriter, r *http.Request) {
    resp, err := http.Get("https://citcall.com/test/countries.json")
    if err != nil {
        http.Error(w, "Failed to fetch countries data",
            http.StatusInternalServerError)
        return
    }
    defer resp.Body.Close()

    var countries []Country
    if err := json.NewDecoder(resp.Body).Decode(&countries); err != nil {
        http.Error(w, "Failed to parse countries data",
            http.StatusInternalServerError)
        return
    }

    tmpl := template.Must(template.New("countries").Parse(htmlTemplate))
```

```

data := TemplateData{Countries: countries}
if err := tmpl.Execute(w, data); err != nil {
    http.Error(w, "Failed to render template", http.StatusInternalServerError)
    return
}
}

const htmlTemplate = `
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Countries Table</title>
    <style>
        table {
            width: 100%;
            border-collapse: collapse;
        }
        table, th, td {
            border: 1px solid black;
        }
        th, td {
            padding: 8px;
            text-align: left;
        }
        th {
            background-color: #f2f2f2;
        }
        img {
            width: 32px;
            height: 32px;
        }
    </style>
</head>
<body>
    <h2>Countries Table</h2>
    <table>
        <thead>
            <tr>
                <th>Flag</th>
                <th>Name</th>
                <th>Dial Code</th>
                <th>ISO Code</th>
            </tr>










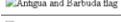

```

```

</thead>
<tbody>
  {{range .Countries}}
  <tr>
    <td></td>
    <td>{{.Name}}</td>
    <td>{{.DialCode}}</td>
    <td>{{.IsoCode}}</td>
  </tr>
  {{end}}
</tbody>
</table>
</body>
</html>
`

```

Setelah Code diatas di run:

Countries Table			
Flag	Name	Dial Code	ISO Code
	Afghanistan	+93	AF
	Aland Islands	+358	AX
	Albania	+355	AL
	Algeria	+213	DZ
	American Samoa	+1684	AS
	Andorra	+376	AD
	Angola	+244	AO
	Anguilla	+1264	AI
	Antarctica	+672	AQ
	Antigua and Barbuda	+1268	AG
	Argentina	+54	AR

2. Performance

Perbandingan

1. Efficiency:

- **Code A** menyimpan panjang tmp dalam variabel size dan kemudian menggunakan nilai ini dalam kondisi loop. Ini sedikit lebih efisien daripada **Code B** karena menghindari perhitungan ulang len(tmp) pada setiap iterasi. Namun, karena loop tidak akan pernah dieksekusi akibat kondisi yang salah (i == len(tmp) adalah false ketika i adalah 0), perbedaan efisiensi ini tidak relevan dalam praktiknya.

2. Readability:

- **Code A** memiliki sedikit keunggulan dalam keterbacaan karena menyimpan len(tmp) dalam variabel size, yang membuatnya jelas bahwa panjang tmp konstan dalam blok kode ini.
- **Code B** lebih ringkas tetapi mungkin kurang jelas karena berulang kali memanggil len(tmp) dalam kondisi loop.

3. Maintainability:

- **Code A** mungkin sedikit lebih mudah untuk dipelihara karena memisahkan logika (menyimpan len(tmp) dalam variabel) dari kondisi loop. Ini membuatnya lebih mudah untuk dipahami dan dimodifikasi logika loop di masa depan.
- **Code B** lebih pendek, yang mungkin lebih mudah untuk sebagian pengembang bekerja dengan, tetapi perhitungan berulang dari len(tmp) bisa kurang intuitif untuk memahami kode.

Kesimpulan

Code A sedikit lebih baik daripada **Code B** karena penyimpanan eksplisit dari 'len(tmp)' dalam variabel, yang membuat kode lebih mudah dibaca dan dipelihara. Meskipun kedua kode memiliki masalah logis yang sama (loop dengan kondisi 'i == size' dan 'func(j int) bool { return j > 100 }' tidak akan pernah dieksekusi), **Code A** lebih mudah dibaca dan dipahami karena memisahkan perhitungan dan perbandingan.

3. Problem Solving

“Masalah utamanya adalah bahwa fungsi-fungsi seperti `Mati`, `Terbang`, dan `Makan` tidak memanipulasi objek `Bebek` secara langsung, karena parameter `b` `Bebek` diterima sebagai nilai (by value) dan bukan sebagai referensi (by reference). Akibatnya, setiap perubahan pada objek `Bebek` di dalam fungsi-fungsi tersebut tidak akan mempengaruhi objek asli di luar fungsi. Untuk memperbaiki masalah ini, kita harus mengubah parameter `b` menjadi pointer (`Bebek`) sehingga fungsi-fungsi dapat memanipulasi objek `Bebek` asli.

```
type Bebek struct {
    energi    int
    hidup     bool
    bisaTerbang bool
    suaraTerbang string
}

func Mati(b *Bebek) {
    b.hidup = false
}

func Terbang(b *Bebek) {
    if b.energi > 0 && b.hidup && b.bisaTerbang {
        fmt.Println(b.suaraTerbang)
        b.energi -= 1
        if b.energi == 0 {
            Mati(b)
        }
    }
}

func Makan(b *Bebek) {
    if b.energi > 0 && b.hidup {
        b.energi += 1
    }
}
```

PROBLEMS 1 OUTPUT TERMINAL PORTS COMMENT

```
PS E:\gotest\testcitcal\n03> go run main.go
• Kwek kwek!
Kwek kwek!
Kwek kwek!
Energi Bebek: 2
Bebek Hidup: true
○ PS E:\gotest\testcitcal\n03> 
```

4. Cryptarithm

```
package main

import (
    "fmt"
    "strconv"
    "strings"
)

func solveCryptarithm(input string) string {
    parts := strings.Fields(input)
    if len(parts) != 5 || parts[3] != "=" {
        return "Format input tidak valid"
    }

    letters := make(map[rune]bool)
    for _, part := range parts {
        if part != "+" && part != "-" && part != "=" {
            for _, char := range part {
                if char >= 'A' && char <= 'Z' {
                    letters[char] = true
                }
            }
        }
    }

    uniqueLetters := make([]rune, 0, len(letters))
    for letter := range letters {
        uniqueLetters = append(uniqueLetters, letter)
    }

    solution := backtrack(parts, uniqueLetters, make(map[rune]int), 0)
    if solution == nil {
        return "Tidak ada solusi"
    }
    return formatSolution(parts, solution)
}

func backtrack(parts []string, letters []rune, assigned map[rune]int, index int) map[rune]int {
    if index == len(letters) {
        if isValid(parts, assigned) {
            return assigned
        }
    }
    return nil
}
```

```

    }

    for digit := 0; digit <= 9; digit++ {
        if index == 0 && digit == 0 && (strings.HasPrefix(parts[0],
string(letters[index])) ||
        strings.HasPrefix(parts[2], string(letters[index])) ||
        strings.HasPrefix(parts[4], string(letters[index]))) {
            continue
        }

        if !isDigitUsed(assigned, digit) {
            newAssigned := make(map[rune]int)
            for k, v := range assigned {
                newAssigned[k] = v
            }
            newAssigned[letters[index]] = digit
            if result := backtrack(parts, letters, newAssigned, index+1); result != nil
{
                return result
            }
        }
    }

    return nil
}

func isDigitUsed(assigned map[rune]int, digit int) bool {
    for _, v := range assigned {
        if v == digit {
            return true
        }
    }
    return false
}

func isValid(parts []string, assigned map[rune]int) bool {
    num1 := parseNumber(parts[0], assigned)
    num2 := parseNumber(parts[2], assigned)
    result := parseNumber(parts[4], assigned)

    if num1 == -1 || num2 == -1 || result == -1 {
        return false
    }

    switch parts[1] {

```

```

    case "+":
        return num1+num2 == result
    case "-":
        return num1-num2 == result
    default:
        return false
    }
}

func parseNumber(s string, assigned map[rune]int) int {
    result := 0
    for _, char := range s {
        if digit, ok := assigned[char]; ok {
            result = result*10 + digit
        } else {
            return -1
        }
    }
    return result
}

func formatSolution(parts []string, assigned map[rune]int) string {
    var solution strings.Builder
    for i, part := range parts {
        if i > 0 {
            solution.WriteString(" ")
        }
        if part == "+" || part == "-" || part == "=" {
            solution.WriteString(part)
        } else {
            for _, char := range part {
                solution.WriteString(strconv.Itoa(assigned[char]))
            }
        }
    }
    return solution.String()
}

func main() {
    inputs := []string{
        "II + II = HIU",
        "ABD - AD = DKL",
    }

    for _, input := range inputs {

```



```
    fmt.Printf("Input: %s\n", input)
    fmt.Printf("Output: %s\n\n", solveCryptarithm(input))
}
}
```

PROBLEMS 1 OUTPUT TERMINAL PORTS

● Input: II + II = HIU
Output: 99 + 99 = 198

Input: ABD - AD = DKL
Output: 312 - 32 = 280

○ PS E:\gotest\testcitcal\n04> █