LAPORAN PRAKTIKUM
KONSTRUKSI PERANGKAT LUNAK

MODUL II

AUTOMATA DAN TABLE-DRIVEN CONSTRUCTION

Disusun Oleh :
Satria Ariq Adelard
Dompas/2211104033 S1SE-06-02

Asisten Praktikum :
Muhamad Taufiq Hidayat

Dosen Pengampu :
Riyan Dwi Yulian Prakoso, S.Kom., M.Kom.

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK
DIREKTORAT TELKOM KAMPUS PURWOKERTO
2025

PENDAHULUAN

A. GUIDED

1. Automata-based Construction (FSM)

Membangun PLC atau pemrograman berdasarkan Automata adalah salah satu model pemrograman di mana program dianggap sebagai status terbatas (FSM) atau automata resmi lainnya dengan status berbeda satu sama lain dan memiliki aturan yang jelas. Berikut adalah indikator utama pemrograman otomatisasi:

1. Waktu implementasi program dipisahkan dengan jelas dalam keadaan saat ini dan tidak melakukan kebetulan dengan keadaan yang ada.

2. Komunikasi apa pun antara keadaan saat ini (beralih antar negara) hanya dapat dilakukan dengan disimpan dengan jelas dalam variabel global.

```
const readline = require("readline");

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

const State = {
    START: "START",
    PLAYING: "PLAYING",
    GAME_OVER: "GAME OVER",
    EXIT: "EXIT"
};

let state = State.START;

function runStateMachine() {
    console.log(`Current State: ${state}`);

    rl.question("Enter Command: ", (command) => {
        switch (state) {
            case State.START:
                if (command.toUpperCase() === "PLAY") {
                    state = State.PLAYING;
                } else if (command.toUpperCase() === "EXIT") {
                    state = State.EXIT;
                }
                break;
            case State.PLAYING:
                if (command.toUpperCase() === "LOSE") {
                    state = State.GAME_OVER;
                } else if (command.toUpperCase() === "EXIT") {
                    state = State.EXIT;
                }
                break;
            case State.GAME_OVER:
                if (command.toUpperCase() === "RESTART") {
                    state = State.START;
                } else if (command.toUpperCase() === "EXIT") {
                    state = State.EXIT;
                }
                break;
        }

        if (state !== State.EXIT) {
            runStateMachine();
        } else {
            console.log("Game Exited.");
            rl.close();
        }
    });
}

runStateMachine();
```

2. **Table-driven Construction**

Building On the Board adalah metode yang memungkinkan informasi untuk menemukan informasi dengan menggunakan tabel alih -alih instruksi logis seolah -olah dan dalam kasus. Pendekatan ini sangat berguna untuk mengatur data yang lebih terstruktur dan efektif, terutama ketika logika digunakan lebih rumit. Dalam kasus sederhana, penggunaan instruksi logis mungkin lebih realistis, tetapi ketika jumlah kondisi meningkat, pendekatan berbasis tabel dapat menyederhanakan kode, meningkatkan kemampuan membaca dan memfasilitasi pemeliharaan.

, disutradarai oleh meja, ada dua hal yang perlu dipertimbangkan sebelum dilakukan. Pertama kita harus menentukan cara menemukan item dalam tabel. Dua jenis data tidak dapat digunakan untuk menemukan input secara langsung dalam tabel. Secara umum, cara untuk menemukan item dalam tabel dibagi menjadi tiga, khususnya:

1. Akses langsung.
2. Akses diindeks.
3. Kunjungitangga.

```
Codeium: Refactor | Explain | Generate JSDoc | ×
function getDaysPerMonth(month) {
    const daysPerMonth = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31];
    return daysPerMonth[month - 1] || "Invalid month";
}

console.log(getDaysPerMonth(2)); // Output: 28
console.log(getDaysPerMonth(13)); // Output: Invalid month
```

3. **Stair-step Access:**

Salah satu metode dalam table-driven yang berguna untuk mempermudah pencarian entri apabila terdapat kasus dimana nilai yang didapatkan berdasarkan range yang telah ada seperti contoh index nilai mahasiswa. Untuk lebih jelasnya perhatikan potongan kode berikut.

```
function getGradeByScore(studentScore) {
    const grades = ["A", "AB", "B", "BC", "C", "D", "E"];
    const rangeLimit = [80, 70, 65, 60, 50, 40, 0];

    for (let i = 0; i < rangeLimit.length; i++) {
        if (studentScore >= rangeLimit[i]) {
            return grades[i];
        }
    }
    return "E";
}

console.log(getGradeByScore(75)); // Output: AB
console.log(getGradeByScore(45)); // Output: D
```

BAB II
PENUGASAN (UNGUIDED)

Soal 1: Automata-based Construction (FSM)

Sebuah game memiliki tiga state utama:

- START (awal permainan)
- PLAYING (sedang bermain)
- GAME OVER (permainan berakhir)

Aturan transisi antar state:

1. Dari START, jika pemain mengetik “PLAY”, permainan masuk ke state PLAYING.
2. Dari PLAYING, jika pemain mengetik “LOSE”, permainan masuk ke state GAME OVER.
3. Dari GAME OVER, jika pemain mengetik “RESTART”, permainan kembali ke state START.
4. Pemain bisa keluar kapan saja dengan mengetik “EXIT”.

Source Codenya:

```
class Game {
    constructor() {
        this.state = "START";
    }

    transition(command) {
        switch (this.state) {
            case "START":
                if (command === "PLAY") {
                    this.state = "PLAYING";
                    console.log("Game dimulai!");
                }
                break;
            case "PLAYING":
                if (command === "LOSE") {
                    this.state = "GAME OVER";
                    console.log("Game Over!");
                }
                break;
            case "GAME OVER":
                if (command === "RESTART") {
                    this.state = "START";
                    console.log("Game di-restart. Kembali ke START.");
                }
                break;
        }

        if (command === "EXIT") {
            console.log("Keluar dari permainan.");
            process.exit();
        }
    }

    start() {
        const readline = require("readline").createInterface({
            input: process.stdin,
            output: process.stdout
        });

        console.log("Permainan dimulai. Ketik PLAY untuk bermain, EXIT untuk keluar.");

        readline.on("line", (input) => {
            this.transition(input.trim().toUpperCase());
            console.log("State saat ini:", this.state);
        });
    }
}

const game = new Game();
game.start();
```

OutPutnya:

```
PS D:\PraktikumKPL\KPL_Satria_Ariq_Adelard_Dompas_2211104033_SE062\Praktikum2\UNGUIDED> node ariq.js

=== START SCREEN ===
Enter Command: PLAY

=== PLAYING SCREEN ===
Enter Command: LOST
Invalid command! Type 'LOSE' or 'EXIT'.

=== PLAYING SCREEN ===
Enter Command: RESTART
Invalid command! Type 'LOSE' or 'EXIT'.

=== PLAYING SCREEN ===
Enter Command: EXIT

Exiting game. Goodbye!
PS D:\PraktikumKPL\KPL_Satria_Ariq_Adelard_Dompas_2211104033_SE062\Praktikum2\UNGUIDED>
```