LAPORAN PRAKTIKUM

KONSTRUKSI PERANGKAT LUNAK

MODUL 4

AUTOMATA DAN TABLE-DRIVEN CONSTRUCTION

Disusun Oleh :

Aji Prasetyo Nugroho / 2211104049

S1SE-06-2

Asisten Praktikum :

Muhammad Taufiq Hidayat

Dosen Pengampu :

Riyan Dwi Yulian Prakoso, S.Kom., M.Kom.

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. LANDASAN TEORI

1. Automata

1.1 Definisi Automata

Automata adalah model matematika yang digunakan untuk merepresentasikan sistem yang dapat berpindah dari satu keadaan (state) ke keadaan lainnya berdasarkan input tertentu. Automata digunakan dalam berbagai bidang, seperti teori bahasa formal, pemrograman, dan desain kompilator.

1.2 Jenis-Jenis Automata

Automata terdiri dari beberapa jenis utama:

1. Finite State Automata (FSA): Model paling sederhana yang hanya memiliki sejumlah keadaan terbatas.
2. Pushdown Automata (PDA): Memiliki tambahan stack sebagai memori untuk menangani bahasa yang lebih kompleks.
3. Turing Machine (TM): Model automata yang lebih kuat dengan tape yang tidak terbatas, sering digunakan untuk memodelkan komputasi umum.

1.3 Finite State Automata (FSA)

FSA terdiri dari:

1. State (Q): Kumpulan keadaan.
2. Alphabet (Σ): Kumpulan simbol input.
3. Transition Function (δ): Fungsi yang menentukan perubahan keadaan.
4. Start State (q0): Keadaan awal.
5. Final State (F): Kumpulan keadaan akhir.

FSA dibagi menjadi:

1. Deterministic Finite Automaton (DFA): Setiap input memiliki satu transisi keadaan.
2. Non-Deterministic Finite Automaton (NFA): Input dapat memiliki beberapa kemungkinan transisi.

2. Table-Driven Construction

2.1 Definisi Table-Driven Construction

Table-Driven Construction adalah metode implementasi automata menggunakan tabel transisi untuk menentukan pergerakan antar state berdasarkan input yang diberikan. Metode ini sering digunakan dalam pembuatan lexer dalam kompilator dan sistem kendali berbasis aturan.

2.2 Komponen Table-Driven Construction

1. Tabel Transisi: Matriks yang mendefinisikan perpindahan state berdasarkan input tertentu.
2. Pointer State Saat Ini: Menunjukkan state aktif dalam automata.
3. Loop Pemrosesan Input: Iterasi yang membaca input dan melakukan transisi berdasarkan tabel.

2.3 Implementasi Table-Driven Construction

Table-Driven Construction dapat diimplementasikan dengan langkah-langkah berikut:

1. Mendefinisikan daftar state dan input.
2. Membuat tabel transisi yang berisi hubungan antara state dan input.
3. Menjalankan loop yang membaca input dan berpindah state berdasarkan tabel transisi.
4. Menentukan apakah automata berakhir di final state atau tidak.

2.4 Kelebihan dan Kekurangan

1. Kelebihan:
    1) Memisahkan logika transisi dari kode utama.
    2) Mudah diubah dan diperluas tanpa mengubah kode program utama.
2. Kekurangan:
    1) Memerlukan penyimpanan lebih banyak untuk tabel transisi.
    2) Kurang fleksibel dalam menangani aturan kompleks dibandingkan implementasi berbasis kode langsung.

B. GUIDED

1. App.js

Source Code

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

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

const State = {
    START: "START",
    GAME: "GAME",
    PAUSE: "PAUSE",
    EXIT: "EXIT"
};

let state = State.START;

function runStateMachine() {
    console.log(`${state} SCREEN`);
    rl.question("Enter Command: ", (command) => {
        switch (state) {
            case State.START:
                if (command === "ENTER") state = State.GAME;
                else if (command === "QUIT") state = State.EXIT;
                break;
            case State.GAME:
                if (command === "ESC") state = State.PAUSE;
                break;
            case State.PAUSE:
                if (command === "BACK") state = State.GAME;
                else if (command === "HOME") state = State.START;
                else if (command === "QUIT") state = State.EXIT;
                break;
        }
        if (state !== State.EXIT) {
            runStateMachine();
        } else {
            console.log("EXIT SCREEN");
            rl.close();
        }
    });
}

runStateMachine();
```

Output

```
PS D:\Praktikum KPL\Praktikum_2> node app.js
START SCREEN
Enter Command: BACK
START SCREEN
Enter Command: EXIT
START SCREEN
Enter Command: ESC
START SCREEN
Enter Command: QUIT
EXIT SCREEN
```

2. month.js

Source Code

```
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
```

Output

```
PS D:\Praktikum KPL\Praktikum_2> node month.js
28
Invalid month
```

3. studentscore.js

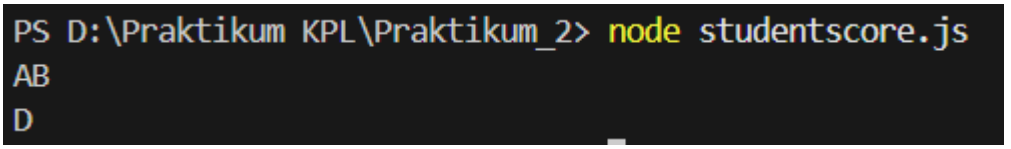
Source Code

```
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
```

Output



C. 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 Code

```
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();
```

Output

```
PS D:\Praktikum KPL\Praktikum_2> node tugas.js
Permainan dimulai. Ketik PLAY untuk bermain, EXIT untuk keluar.
> PLAY
Game dimulai!
State saat ini: PLAYING
LOSE
Game Over!
State saat ini: GAME OVER
RESTART
Game di-restart. Kembali ke START.
State saat ini: START
PLAY
Game dimulai!
State saat ini: PLAYING
EXIT
Keluar dari permainan.
```

Penjelasan Program

Program di atas adalah implementasi dari finite state machine (FSM) untuk simulasi permainan sederhana menggunakan JavaScript. Objek Game memiliki tiga state utama: "START", "PLAYING", dan "GAME OVER", serta mendukung transisi antara state berdasarkan input yang diberikan oleh pengguna. Metode transition(command) menangani perubahan state berdasarkan perintah seperti "PLAY" untuk memulai permainan, "LOSE" untuk mengakhiri permainan, dan "RESTART" untuk kembali ke keadaan awal. Selain itu, ada perintah "EXIT" yang memungkinkan pengguna keluar dari permainan dengan menggunakan process.exit().

Metode start() menggunakan modul readline untuk membaca input dari pengguna secara interaktif melalui terminal. Saat program dijalankan, pengguna diminta untuk memasukkan perintah, dan program akan merespons dengan mengubah state serta mencetak status permainan saat ini. Dengan pendekatan ini, program menunjukkan bagaimana automata dapat digunakan dalam desain sistem berbasis aturan, seperti permainan atau mesin status lainnya.