LAPORAN PRAKTIKUM
KONSTRUKSI PERANGKAT LUNAK

MODUL II
AUTOMATA DAN TABLE-DRIVEN CONSRUCTION

Disusun Oleh :

Arzario Irsyad Al Fatih

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

BAB I
PENDAHULUAN

A. DASAR TEORI

1. Automata-Based Construction

Automata-Based Construction adalah metode pengembangan perangkat lunak yang menggunakan konsep teori automata untuk memodelkan perilaku sistem. Dalam metode ini, alur logika aplikasi dipetakan menggunakan finite state machine (FSM) sehingga pengambilan keputusan otomatis dapat dilakukan dengan mudah. Automata-Based Construction umumnya digunakan dalam pembuatan parser, pengenalan pola, dan sistem kendali otomatis. Keunggulan metode ini antara lain mampu menangani masalah kompleks dan mempermudah visualisasi kondisi serta transisi sistem.

Poin Penting Automata-Based Construction:

a. Menggunakan konsep automata dan FSM untuk pengambilan keputusan otomatis.
b. Digunakan dalam parser, pengenalan pola, dan sistem kendali otomatis.
c. Mengatasi masalah kompleks dan memudahkan visualisasi transisi sistem.

2. Table-Driven Construction

Table-Driven Construction merupakan metode pengembangan perangkat lunak yang menggambarkan perilaku sistem berdasarkan input dan kondisi tertentu menggunakan tabel data. Dengan memisahkan logika pengambilan keputusan dari kode utama, metode ini dapat mengurangi kompleksitas kode. Table-Driven Construction sering digunakan dalam pemrosesan transaksi dan interpreter bahasa pemrograman. Metode ini memiliki beberapa keunggulan, seperti fleksibilitas dalam modifikasi logika, mendukung modularitas, dan mengurangi risiko kesalahan pemrograman.

Poin Penting Table-Driven Construction:

a. Menggunakan tabel data untuk pengambilan keputusan.
b. Mengurangi kompleksitas kode dan mendukung modularitas.
c. Fleksibel dan mudah dimodifikasi.

B. MAKSUD DAN TUJUAN

1. MAKSUD

Modul ini bertujuan untuk memberikan pemahaman dan keterampilan dalam menerapkan Automata-Based Construction dan Table-Driven Construction dalam pengembangan perangkat lunak. Dengan menggunakan pendekatan ini, diharapkan peserta dapat memahami konsep pemrograman berbasis state dan optimasi pengambilan keputusan melalui tabel, yang dapat diterapkan dalam berbagai jenis sistem perangkat lunak.

2. TUJUAN

Setelah mempelajari dan mempraktikkan modul ini, peserta diharapkan mampu:

a. Memahami konsep Automata-Based Construction sebagai paradigma pemrograman berbasis finite-state machine (FSM).
b. Mampu mengimplementasikan Automata-Based Construction dalam bahasa node.js menggunakan struktur enum dan switch-case untuk menangani transisi state.
c. Memahami konsep Table-Driven Construction sebagai alternatif pengambilan keputusan tanpa menggunakan if-else atau switch-case secara langsung.
d. Mampu mengimplementasikan Table-Driven Construction dengan berbagai metode pencarian data, seperti Direct Access, Indexed Access, dan Stair-step Access dalam.
e. Menguasai teknik optimasi dalam pemrograman dengan memilih pendekatan yang sesuai berdasarkan kompleksitas logika yang dihadapi.
f. Mengembangkan solusi perangkat lunak yang lebih efisien dan terstruktur, terutama dalam sistem yang membutuhkan pengelolaan banyak state atau kondisi

BAB II

IMPLEMENTASI (GUIDED)

1. Automata-Based Construction

a. 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();
```

b. Output

```
E02\04_Automata dan Table-Driven Construction\GUIDED\automataBasedConstruction.js"
START SCREEN
Enter Command: ENTER
GAME SCREEN
Enter Command: ESC
PAUSE SCREEN
Enter Command: BACK
GAME SCREEN
Enter Command: HOME
GAME SCREEN
Enter Command: QUIT
GAME SCREEN
Enter Command:
```

c. Penjelasan

Kode di atas menerapkan Automata-Based Construction menggunakan Finite State Machine (FSM) dalam Node.js untuk mengendalikan alur logika aplikasi interaktif berbasis terminal. Modul readline digunakan untuk membaca input pengguna dan mengatur antarmuka input/output. Objek State menyimpan kondisi aplikasi (START, GAME, PAUSE, EXIT), dan variabel state menginisialisasi kondisi awal menjadi START. Fungsi runStateMachine menggunakan struktur switch-case untuk mengubah state berdasarkan input pengguna, dengan dukungan perintah seperti ENTER, QUIT, ESC, BACK, dan HOME. Fungsi ini berjalan secara rekursif sampai state berubah menjadi EXIT, lalu menutup antarmuka input/output dengan rl.close. Pendekatan ini efektif dalam memetakan logika kondisi dan transisi sistem secara otomatis.

2. Table-Driven Construction

a. Code

```
// Direct Access
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));
console.log(getDaysPerMonth(13));

// Stair-step Access
```

```
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));
console.log(getGradeByScore(45));
```

b. Output

```
E02\04_Automata dan Table-Driven Construction\GUIDED\tableDrivenConstruction.js"
28
Invalid month
AB
D
```

c. Penjelasan

Kode di atas terdiri dari dua fungsi: getDaysPerMonth dan getGradeByScore. Fungsi getDaysPerMonth menggunakan metode Direct Access dengan mengakses array daysPerMonth yang berisi jumlah hari setiap bulan dalam setahun berdasarkan indeks (dimulai dari 0), lalu mengembalikan jumlah hari sesuai parameter month yang diberikan atau menampilkan Invalid month jika input tidak valid. Fungsi getGradeByScore menggunakan metode Stair-step Access untuk mengonversi nilai siswa menjadi grade dengan mencocokkan nilai studentScore terhadap array rangeLimit menggunakan perulangan for. Jika nilai memenuhi kondisi, fungsi mengembalikan grade yang sesuai dari array grades. Jika tidak ada kecocokan, grade E dikembalikan secara default. Hasil output kode menunjukkan jumlah hari di bulan Februari (28), pesan kesalahan untuk input bulan yang tidak valid, serta grade nilai siswa berdasarkan skor yang diberikan.

BAB III

PENUGASAN (UNGUIDED)

1. Soal 1 Unguided

a. Code

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

    transition(action) {
        switch (this.state) {
            case "START":
                if (action === "PLAY") {
                    this.state = "PLAYING";
                } else {
                    console.log("Aksi tidak valid.");
                }
                break;
            case "PLAYING":
                if (action === "LOSE") {
                    this.state = "GAME OVER";
                } else {
                    console.log("Aksi tidak valid.");
                }
                break;
            case "GAME OVER":
                if (action === "RESTART") {
                    this.state = "START";
                } else {
                    console.log("Aksi tidak valid.");
                }
                break;
        }

        if (action === "EXIT") {
            console.log("Keluar dari permainan.");
            return false; // Menghentikan permainan
        }

        return true; // Lanjutkan permainan
    }

    async run() {
        console.log("Game dimulai. Ketik PLAY, LOSE, RESTART, atau
EXIT.");

        while (true) {
            console.log("State saat ini:", this.state);
```

```
            const action = await this.getInput("Masukkan aksi: ");
            if (!this.transition(action.toUpperCase())) break;
        }
    }

    getInput(promptText) {
        return new Promise((resolve) => {
            const readline = require("readline").createInterface({
                input: process.stdin,
                output: process.stdout,
            });
            readline.question(promptText, (input) => {
                readline.close();
                resolve(input);
            });
        });
    }
}

// Jalankan game
const game = new GameFSM();
game.run();
```

b. Output

```
E02\04_Automata dan Table-Driven Construction\UNGUIDED\GameFSM.js"
Game dimulai. Ketik PLAY, LOSE, RESTART, atau EXIT.
State saat ini: START
Masukkan aksi: PLAY
State saat ini: PLAYING
Masukkan aksi: LOSE
State saat ini: GAME OVER
Masukkan aksi: RESTART
State saat ini: START
Masukkan aksi: EXIT
Aksi tidak valid.
Keluar dari permainan.
```

c. Penjelasan

Kode JavaScript di atas mengimplementasikan finite state machine (FSM) untuk game sederhana dengan tiga state utama: START, PLAYING, dan GAME OVER. Class GameFSM memiliki metode transition(action) untuk mengatur perubahan state berdasarkan input pemain, serta menghentikan permainan jika pemain mengetik EXIT. Metode run() menggunakan asynchronous input dengan readline agar pengguna dapat berinteraksi di terminal secara real-time. Saat dijalankan, pemain dapat memasukkan

perintah PLAY untuk mulai bermain, LOSE untuk kalah dan masuk ke state GAME OVER, RESTART untuk mengulang permainan dari awal, dan EXIT untuk keluar. Pendekatan asynchronous pada metode getInput() memastikan input pengguna diproses secara efektif, menjadikan program responsif dan interaktif.