LAPORAN PRAKTIKUM
KONSTRUKSI PERANGKAT BERGERAK

MODUL II

AUTOMATA DAN TABLE-DRIVEN CONSTRUCTION

Disusun Oleh :

Yogi Hafidh Maulana

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 programming adalah paradigma pemrograman yang memandang program sebagai finite-state machine (FSM) atau formal automaton yang memiliki berbagai state yang saling berkaitan dan memiliki aturan tertentu yang jelas. Ciri-ciri utama Automata-Based Construction:

a. Pemrosesan berbasis state – Setiap eksekusi program dibatasi dalam satu state tanpa adanya eksekusi tumpang-tindih antar state.
b. Perpindahan antar state eksplisit – Komunikasi antara state hanya dapat dilakukan melalui variabel global atau mekanisme eksplisit lainnya.

2. Table-Driven Construction

Table-Driven Construction adalah teknik pemrograman yang menggantikan logic statements seperti if-else atau switch-case dengan tabel yang berisi informasi yang digunakan untuk pengambilan keputusan. Keunggulan Table-Driven Construction:

a. Mengurangi kompleksitas kode ketika banyak kondisi yang perlu diproses.
b. Mempermudah perubahan aturan atau logika hanya dengan mengubah tabel.

Metode Pencarian dalam Table-Driven Construction:

a. Direct Access – Menggunakan indeks langsung untuk mengakses nilai dari tabel.
b. Indexed Access – Menggunakan tabel tambahan untuk menyimpan indeks dari tabel utama agar lebih hemat memori.
c. Stair-step Access – Menggunakan tabel untuk mencocokkan rentang nilai dengan output tertentu.

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.

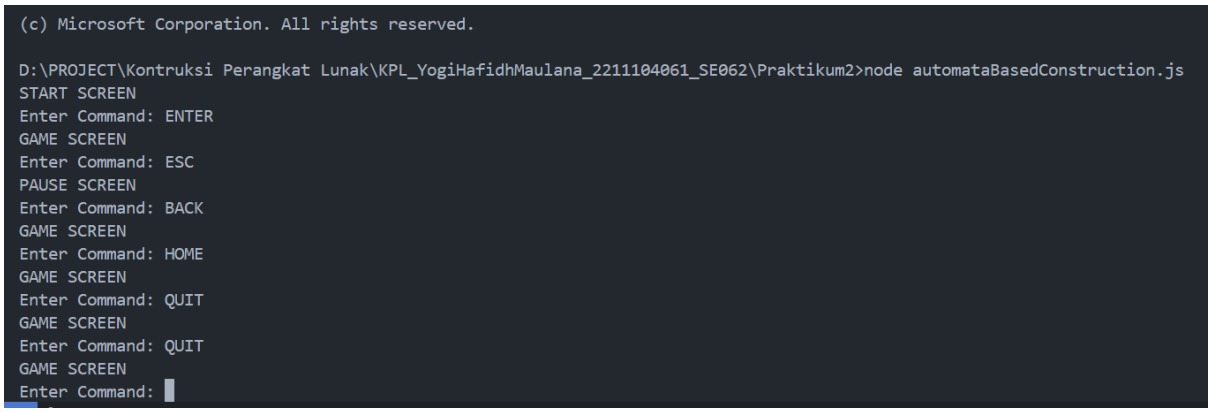
BAB II
IMPLEMENTASI (GUIDED)

A. Automata-based Construction (FSM)

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



Deskripsi Code

Program membaca input pengguna dengan readline dan mengubah state berdasarkan perintah yang diberikan. Ada empat state utama: START, GAME, PAUSE, dan EXIT. Saat pertama dijalankan, program berada di state START dan akan menampilkan pesan sesuai dengan state saat ini. Pengguna dapat memasukkan perintah seperti ENTER untuk berpindah ke state GAME, ESC untuk masuk ke PAUSE, dan QUIT untuk keluar dari program. Setiap kali pengguna memasukkan perintah, program akan memproses input tersebut dan menjalankan kembali fungsi hingga pengguna mengetik QUIT, yang akan menutup program.

B. Table-driven Construction

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));
```

Output

```
D:\PROJECT\Kontruksi Perangkat Lunak\KPL_YogiHafidhMaulana_2211104061_SE062\Praktikum2>
28
Invalid month
AB
D
```

Deskripsi Code

Program ini menerapkan Table-Driven Construction dalam dua metode: Direct Access dan Stair-step Access. Pada metode Direct Access, program menyimpan jumlah hari dalam setiap bulan dalam sebuah array, lalu mengambil nilai berdasarkan indeks bulan yang dimasukkan pengguna. Jika bulan tidak valid, program mengembalikan pesan "Invalid month". Sedangkan pada metode Stair-step Access, program menentukan nilai huruf berdasarkan rentang skor yang telah ditentukan dalam dua array: satu untuk batas skor dan satu lagi untuk nilai huruf. Program akan mengecek skor dari atas ke bawah dan mengembalikan nilai pertama yang sesuai dengan rentang yang tersedia. Teknik ini mempermudah pengambilan keputusan tanpa harus menggunakan banyak if-else atau switch-case, sehingga kode lebih rapi dan mudah dikelola.

BAB III
PENUGASAN (UNGUIDED)

Soal 1: Automata-based Construction (FSM)

Code

```
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(`\n${state} SCREEN`);

  // Menampilkan pesan saat masuk ke setiap state
  switch (state) {
    case State.START:
      console.log(
        "Permainan dimulai! Ketik 'PLAY' untuk mulai bermain atau 'EXIT' untuk keluar."
      );
      break;
    case State.PLAYING:
      console.log(
        "Anda sedang bermain. Ketik 'LOSE' jika kalah atau 'EXIT' untuk keluar."
      );
      break;
    case State.GAME_OVER:
      console.log(
        "Game Over! Ketik 'RESTART' untuk bermain lagi atau 'EXIT' untuk keluar."
      );
      break;
  }

  rl.question("Enter Command: ", (command) => {
    switch (state) {
      case State.START:
        if (command === "PLAY") {
          state = State.PLAYING;
          console.log("\nAnda memasuki mode permainan.");
        } else if (command === "EXIT") {
          state = State.EXIT;
        }
        break;
      case State.PLAYING:
        if (command === "LOSE") {
          state = State.GAME_OVER;
          console.log("\nAnda kalah! Permainan berakhir.");
        } else if (command === "EXIT") {
          state = State.EXIT;
        }
        break;
      case State.GAME_OVER:
        if (command === "RESTART") {
          state = State.START;
          console.log("\nPermainan di-restart. Kembali ke menu awal.");
        } else if (command === "EXIT") {
          state = State.EXIT;
        }
        break;
    }

    if (state !== State.EXIT) {
      runStateMachine();
    } else {
      console.log("\nEXITING GAME... Terima kasih telah bermain!");
      rl.close();
    }
  });
}

runStateMachine();
```

Output

```
D:\PROJECT\Kontruksi Perangkat Lunak\KPL_YogiHafidhMaulana_2211104061_SE062\Praktikum2\UNGUIDED>node game.js

START SCREEN
Permainan dimulai! Ketik 'PLAY' untuk mulai bermain atau 'EXIT' untuk keluar.
Enter Command: PLAY

Anda memasuki mode permainan.

PLAYING SCREEN
Anda sedang bermain. Ketik 'LOSE' jika kalah atau 'EXIT' untuk keluar.
Enter Command: LOSE

Anda kalah! Permainan berakhir.

GAME OVER SCREEN
Game Over! Ketik 'RESTART' untuk bermain lagi atau 'EXIT' untuk keluar.
Enter Command: RESTART

Permainan di-restart. Kembali ke menu awal.

START SCREEN
Permainan dimulai! Ketik 'PLAY' untuk mulai bermain atau 'EXIT' untuk keluar.
Enter Command: EXIT

EXITING GAME... Terima kasih telah bermain!

D:\PROJECT\Kontruksi Perangkat Lunak\KPL_YogiHafidhMaulana_2211104061_SE062\Praktikum2\UNGUIDED>
```

Deskripsi Code

Program membaca input pengguna melalui terminal menggunakan modul readline dan memiliki tiga state utama: START, PLAYING, dan GAME OVER, serta satu state tambahan EXIT untuk keluar dari permainan. Transisi antar state dilakukan berdasarkan input pengguna, misalnya mengetik "PLAY" untuk memulai permainan, "LOSE" untuk masuk ke state GAME OVER, dan "RESTART" untuk kembali ke state START. Setiap kali masuk ke state baru, program menampilkan pesan yang menjelaskan kondisi permainan saat ini. Program berjalan dalam loop rekursif hingga pengguna mengetik "EXIT", yang akan menutup permainan dan menghentikan program.