

**LAPORAN PRAKTIKUM  
KONSTRUKSI PERANGKAT BERGERAK**

**MODUL II  
AUTOMATA DAN TABLE-DRIVEN CONSTRUCTION**



**Disusun Oleh :**

**Ade Fatkhul Anam**

**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**

Automata-based programming adalah paradigma pemrograman di mana program dianggap sebagai finite-state machine (FSM) yang memiliki beberapa state yang saling berhubungan dengan aturan transisi tertentu. Prinsip utama dari automata-based programming adalah pemisahan yang jelas antara eksekusi program dalam setiap state serta perpindahan antar state yang hanya dapat dilakukan secara eksplisit melalui variabel global.

Table-driven construction adalah pendekatan pemrograman yang menggantikan penggunaan pernyataan logika seperti if dan switch dengan tabel data untuk menentukan hasil yang diinginkan. Terdapat tiga metode utama dalam table-driven construction, yaitu Direct Access, Indexed Access, dan Stair-step Access.

### **B. MAKSUD DAN TUJUAN**

- Memahami konsep dan implementasi automata-based programming menggunakan finite-state machine (FSM).
- Memahami penerapan table-driven construction dalam pemrograman.
- Mampu mengimplementasikan aturan transisi antar state dalam FSM dengan bahasa pemrograman.
- mempraktikkan metode table-driven untuk pengolahan data berdasarkan tabel.

## BAB II IMPLEMENTASI (GUIDED)

### 1. Automata-based Construction (FSM)

Source code:

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

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

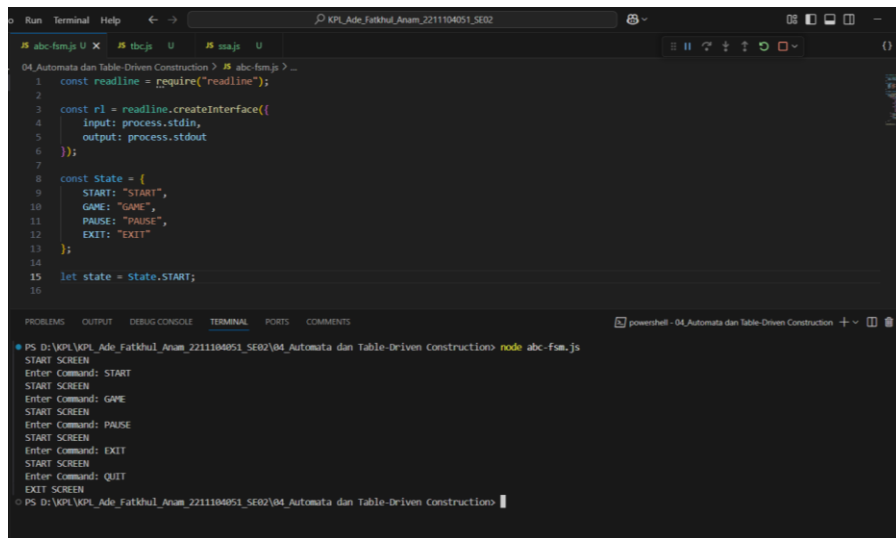
const State = {
  START: "START",
  GAME: "GAME",
  PAUSE: "PAUSE",
  EXIT: "EXIT"
};

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:



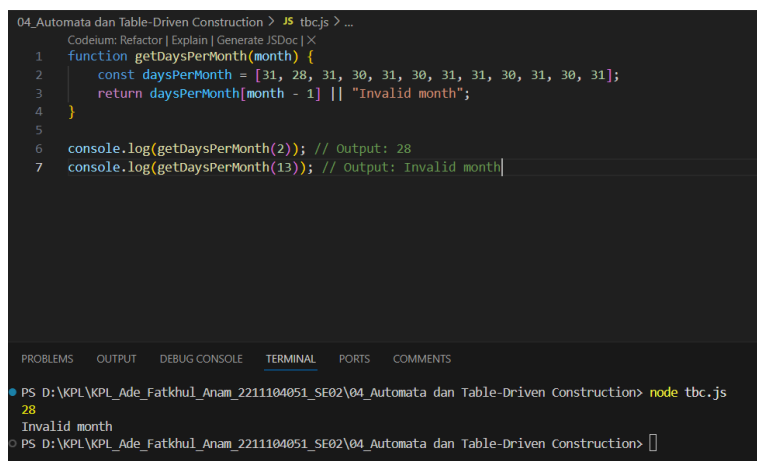
The screenshot shows a VS Code editor with a file named `abc-fsm.js` open. The code defines a state machine with states `START`, `GAME`, `PAUSE`, and `EXIT`. The terminal output shows the program running and responding to user input commands: `START SCREEN`, `Enter Command: START`, `START SCREEN`, `Enter Command: GAME`, `START SCREEN`, `Enter Command: PAUSE`, `START SCREEN`, `Enter Command: EXIT`, `START SCREEN`, `Enter Command: QUIT`, and `EXIT SCREEN`.

## 2. Table-driven Construction

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:



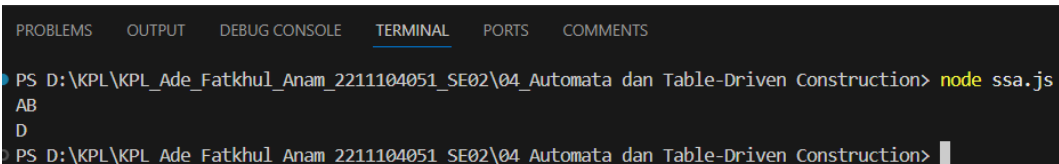
The screenshot shows a VS Code editor with a file named `tbc.js` open. The code defines a function `getDaysPerMonth` that returns the number of days in a given month. The terminal output shows the program running and displaying the results of the function calls: `28` and `Invalid month`.

### 3. Stair-step Access:

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:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS  
PS D:\KPL\KPL_Ade_Fatkhul_Anam_2211104051_SE02\04_Automata dan Table-Driven Construction> node ssa.js  
AB  
D  
PS D:\KPL\KPL_Ade_Fatkhul_Anam_2211104051_SE02\04_Automata dan Table-Driven Construction> |
```

#### 4. Penjelasan

Penjelasan Implementasi dari Codingan di Atas Kode yang diberikan merupakan implementasi finite-state machine (FSM) dalam permainan sederhana dengan tiga state utama: START, PLAYING, dan GAME OVER. Berikut adalah aturan transisinya:

- Dari START, pemain bisa masuk ke PLAYING dengan perintah "PLAY".
- Dari PLAYING, permainan berakhir (GAME OVER) jika pemain mengetik "LOSE".
- Dari GAME OVER, permainan bisa diulang dengan perintah "RESTART".
- Pemain dapat keluar dari permainan kapan saja dengan mengetik "EXIT".

Kode ini menggunakan perulangan untuk menjaga permainan tetap berjalan hingga pemain memilih untuk keluar. Transisi antar state dikontrol melalui struktur switch-case berdasarkan input yang diberikan oleh pengguna.

### BAB III

#### 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**".

Jawab:

Source 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"
};

let state = State.START;

function runStateMachine() {
  console.log(`Current State: ${state}`);
  rl.question("Enter Command: ", (command) => {
    switch (state) {
```

```
        case State.START:
            if (command === "PLAY") state = State.PLAYING;
            else if (command === "EXIT") {
                console.log("Exiting Game...");
                return rl.close();
            }
            break;
        case State.PLAYING:
            if (command === "LOSE") state = State.GAME_OVER;
            else if (command === "EXIT") {
                console.log("Exiting Game...");
                return rl.close();
            }
            break;
        case State.GAME_OVER:
            if (command === "RESTART") state = State.START;
            else if (command === "EXIT") {
                console.log("Exiting Game...");
                return rl.close();
            }
            break;
    }
    runStateMachine();
});
}
```

```
runStateMachine();
```



## Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

PS D:\KPL\KPL_Ade_Fatkul_Anam_2211104051_SE02\04_Automata dan Table-Driven Construction> node gameFSM.js
Current State: START
Enter Command: PLAY
Current State: PLAYING
Enter Command: LOSE
Current State: GAME OVER
Enter Command: RESTART
Current State: START
Enter Command: EXIT
Exiting Game...
PS D:\KPL\KPL_Ade_Fatkul_Anam_2211104051_SE02\04_Automata dan Table-Driven Construction> |
```

## Penjelasan:

- Tambahkan state baru "PAUSED" yang memungkinkan pemain menghentikan sementara permainan dengan mengetik "PAUSE" saat di PLAYING.
- Dari "PAUSED", pemain bisa kembali ke "PLAYING" dengan perintah "RESUME" atau kembali ke "START" dengan perintah "HOME".
- Pastikan sistem tetap menangani perintah "EXIT" kapan saja untuk keluar dari permainan.
- Tambahkan validasi input agar jika pemain mengetik perintah yang tidak dikenal, sistem menampilkan pesan kesalahan dan meminta input ulang.