

**LAPORAN PRAKTIKUM  
KONSTRUKSI PERANGKAT LUNAK**

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



**Disusun Oleh:**

**Dewi Atika Muthi / 2211104042**

**SE-06-02**

**Asisten Praktikum:**

**Muhamad Taufiq Hidayat**

**Dosen Pengampu:**

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

**PROGRAM STUDI S1 SOFTWARE ENGINEERING**

**FAKULTAS INFORMATIKA**

**TELKOM UNIVERSITY PURWOKERTO**

**2025**

# BAB I

## PENDAHULUAN

### A. DASAR TEORI

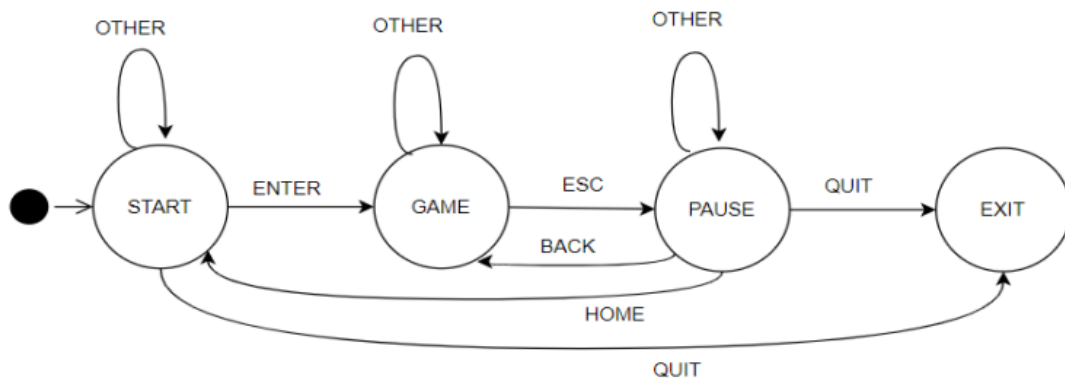
**Automata Construction** atau Automata-based programming adalah paradigma pemrograman di mana program dianggap sebagai finite-state machine (FSM) atau automaton formal lainnya dengan berbagai state yang memiliki aturan tertentu. Indikator utama dalam Automata-based programming meliputi:

1. Pemisahan eksekusi program berdasarkan state.
2. Perpindahan antar state dilakukan secara eksplisit dan tersimpan dalam variabel global.

**Table-driven Construction** adalah metode yang menggunakan tabel untuk mencari informasi dibandingkan dengan menggunakan pernyataan logika (if dan case). Terdapat tiga metode utama dalam table-driven:

1. **Direct Access** : Menggunakan key langsung untuk mencari nilai dalam tabel.
2. **Indexed Access** : Memanfaatkan tabel tambahan untuk mengurangi penggunaan memori.
3. **Stair-step Access** : Menggunakan rentang nilai untuk menentukan hasil.

**Gambar ilustrasi:**



*Gambar 1.1 State Diagram Automata-based Programming*

### B. MAKSUD DAN TUJUAN

Maksud dan tujuan dari praktikum ini adalah:

1. Menguasai implementasi Automata-based Construction.
2. Menguasai penerapan Table-driven Construction dalam.
3. Memahami konsep finite-state machine dan tabel pencarian dalam pemrograman.

## BAB II IMPLEMENTASI

### A. PRAKTIKUM (GUIDED)

#### 1. Automata-based Construction

##### Soal Studi Case

Simulasi sederhana dari finite-state machine untuk mengontrol tampilan layar dengan berbagai state.

##### Sourcecode

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

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

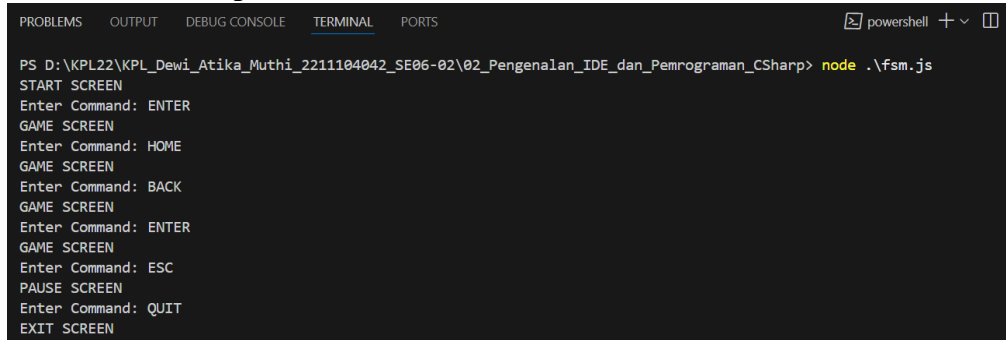
const State = {
  START: "START",
  GAME: "GAME",
  PAUSE: "PAUSE",
  EXIT: "EXIT"
};

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();
```

## Screenshoot Output



```
PS D:\KPL22\KPL_Dewi_Atika_Muthi_2211104042_SE06-02\02_Pengenalan_IDE_dan_Pemrograman_CSharp> node .\fsm.js
START SCREEN
Enter Command: ENTER
GAME SCREEN
Enter Command: HOME
GAME SCREEN
Enter Command: BACK
GAME SCREEN
Enter Command: ENTER
GAME SCREEN
Enter Command: ESC
PAUSE SCREEN
Enter Command: QUIT
EXIT SCREEN
```

## Deskripsi Program

- Program ini berjalan dalam loop hingga pengguna memasukkan “QUIT”.
- Menggunakan readline untuk membaca input dari terminal.
- State berpindah berdasarkan input pengguna.

## Rincinya:

Modul `readline` digunakan untuk membaca input dari terminal dan memberikan interaksi berbasis teks, `readline.createInterface` digunakan untuk membuat antarmuka yang memungkinkan program menerima input pengguna dari terminal. Objek `State` berisi daftar state yang ada dalam program:

- `START` : Halaman awal sebelum permainan dimulai.
- `GAME` : Keadaan ketika permainan sedang berlangsung.
- `PAUSE` : Keadaan permainan yang sedang dijeda.
- `EXIT` : Keadaan untuk keluar dari program.

Variabel `state` digunakan untuk menyimpan state aktif dalam program. Program dimulai dari state `START`. Fungsi `runStateMachine()` bertugas mencetak state aktif ke layar dan meminta input dari pengguna, input pengguna digunakan untuk menentukan transisi state.

Switch berisi logika transisi antar state:

- `START`:
  - o Jika pengguna memasukkan "ENTER", state berubah ke `GAME`.
  - o Jika pengguna memasukkan "QUIT", state berubah ke `EXIT`.
- `GAME`:
  - o Jika pengguna memasukkan "ESC", state berubah ke `PAUSE`.
- `PAUSE`:
  - o Jika pengguna memasukkan "BACK", state kembali ke `GAME`.
  - o Jika pengguna memasukkan "HOME", state kembali ke `START`.
  - o Jika pengguna memasukkan "QUIT", state berubah ke `EXIT`.

Dan dalam kondisi `if-else`, jika state belum mencapai `EXIT`, fungsi `runStateMachine()` akan dipanggil kembali untuk terus menjalankan program.

Jika state EXIT, program menampilkan "EXIT SCREEN" dan menutup antarmuka readline.

Selanjutnya, program langsung dijalankan dengan memanggil fungsi `runStateMachine()`.

## 2. Table-driven Construction

### a. Direct Access

#### Soal Studi Case

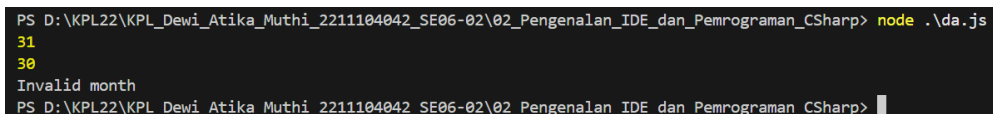
Menentukan jumlah hari dalam bulan tertentu berdasarkan input bulan dalam angka.

#### Sourcecode

```
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(3)); // Output: 31
console.log(getDaysPerMonth(4)); // Output: 30
console.log(getDaysPerMonth(13)); // Output: Invalid month
```

#### Screenshoot Output (3, 4, 13)



```
PS D:\KPL22\KPL_Dewi_Atika_Muthi_2211104042_SE06-02\02_Pengenalan_IDE_dan_Pemrograman_CSharp> node .\da.js
31
30
Invalid month
PS D:\KPL22\KPL_Dewi_Atika_Muthi_2211104042_SE06-02\02_Pengenalan_IDE_dan_Pemrograman_CSharp>
```

#### Deskripsi Program

Program ini menggunakan metode Direct Access untuk menentukan jumlah hari dalam suatu bulan berdasarkan input angka bulan (1-12). Program menyimpan data jumlah hari dalam array dan mengaksesnya langsung berdasarkan indeks yang sesuai. Jika input berada di luar rentang 1-12, program akan mengembalikan "Invalid month".

#### Rincinya:

Fungsi `getDaysPerMonth(month)` menerima satu parameter `month`, yang merepresentasikan nomor bulan (1 hingga 12). Array `daysPerMonth` berisi jumlah hari dalam masing-masing bulan dari Januari (index 0) hingga Desember (index 11). Februari memiliki 28 hari, tanpa mempertimbangkan tahun kabisat.

Lalu kita mengembalikan jumlah hari (`return`), `month - 1` digunakan karena array menggunakan indeks mulai dari 0, sedangkan bulan dimulai dari 1. Jika `month` berada dalam rentang 1-12, fungsi akan mengembalikan nilai dari array `daysPerMonth`.

Jika `month` di luar rentang tersebut (misalnya 13 atau 0), ekspresi `daysPerMonth[month - 1]` akan menghasilkan `undefined`, dan operator OR (`||`) akan mengembalikan string "Invalid month" sebagai fallback.

## b. Stair-step Access

### Soal Studi Case

Menentukan nilai huruf berdasarkan skor mahasiswa dengan menggunakan rentang nilai yang telah ditentukan.

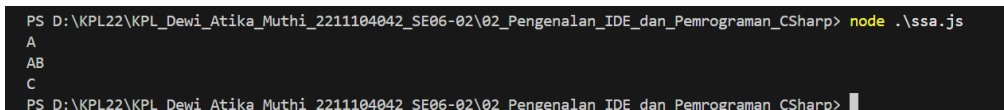
### Sourcecode

```
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(87)); // Output: A
console.log(getGradeByScore(75)); // Output: AB
console.log(getGradeByScore(55)); // Output: C
```

### Screenshoot Output



```
PS D:\KPL22\KPL_Dewi_Atika_Muthi_2211104042_SE06-02\02_Pengenalan_IDE_dan_Pemrograman_CSharp> node .\ssa.js
A
AB
C
PS D:\KPL22\KPL_Dewi_Atika_Muthi_2211104042_SE06-02\02_Pengenalan_IDE_dan_Pemrograman_CSharp>
```

### Deskripsi Program

Program ini menggunakan metode Stair-step Access untuk menentukan nilai huruf berdasarkan skor mahasiswa. Program membandingkan skor input dengan daftar rentang nilai yang telah ditentukan, lalu mengembalikan nilai huruf yang sesuai. Skor tertinggi mendapat nilai "A", sedangkan skor terendah mendapat "E".

### Rincinya:

Fungsi `getGradeByScore(studentScore)` menerima satu parameter `studentScore`, yaitu nilai siswa yang akan dikonversi ke dalam bentuk huruf. `grades` berisi daftar nilai huruf dari A (tertinggi) hingga E (terendah). `rangeLimit` berisi nilai minimum yang diperlukan untuk mendapatkan masing-masing grade. Grade yang ada dalam program:

$\geq 80 \rightarrow A$ ,  $\geq 70 \rightarrow AB$ ,  $\geq 65 \rightarrow B$ ,  $\geq 60 \rightarrow BC$ ,  $\geq 50 \rightarrow C$ ,  $\geq 40 \rightarrow D$ ,  $< 40 \rightarrow E$

Lalu loop `for` akan berjalan melalui `rangeLimit`. Jika `studentScore` lebih besar atau sama dengan nilai pada `rangeLimit[i]`, maka nilai huruf yang sesuai dari `grades[i]` akan dikembalikan. Karena rentang sudah diurutkan dari yang tertinggi ke yang terendah, saat kondisi pertama terpenuhi, fungsi langsung mengembalikan nilai.

## B. LATIHAN (UNGUIDED)

### Soal 1: Automata-based Construction (FSM)

Sebuah game memiliki tiga state utama:

1. START (awal permainan)
2. PLAYING (sedang bermain)
3. GAME OVER (permainan berakhir)

Aturan transisi antar state:

- Dari START, jika pemain mengetik "PLAY", permainan masuk ke state PLAYING.
- Dari PLAYING, jika pemain mengetik "LOSE", permainan masuk ke state GAME OVER.
- Dari GAME OVER, jika pemain mengetik "RESTART", permainan kembali ke state START.
- Pemain bisa keluar kapan saja dengan mengetik "EXIT".

### Sourcecode

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

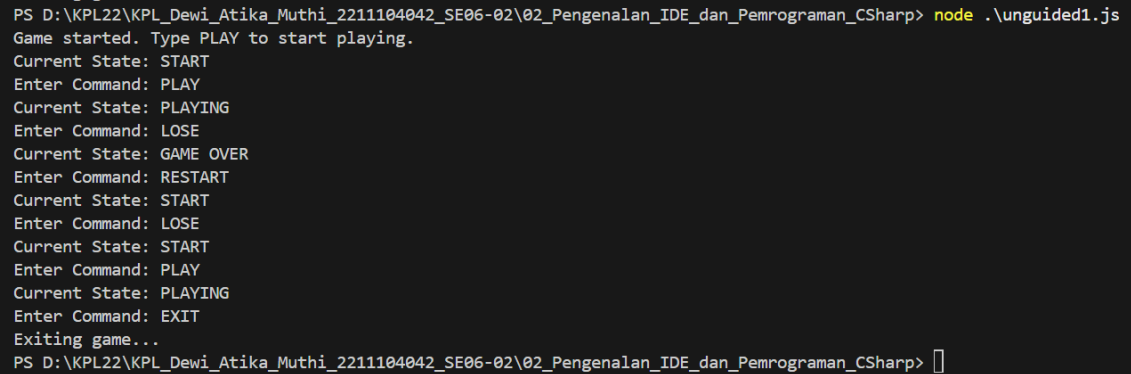
let state = "START";

function gameLoop() {
  console.log("Current State:", state);
  readline.question("Enter Command: ", (command) => {
    command = command.toUpperCase();

    switch (state) {
      case "START":
        if (command === "PLAY") state = "PLAYING";
        else if (command === "EXIT") {
          console.log("Exiting game...");
          readline.close();
          return;
        }
        break;
      case "PLAYING":
        if (command === "LOSE") state = "GAME OVER";
        else if (command === "EXIT") {
          console.log("Exiting game...");
          readline.close();
          return;
        }
        break;
      case "GAME OVER":
        if (command === "RESTART") state = "START";
        else if (command === "EXIT") {
          console.log("Exiting game...");
          readline.close();
          return;
        }
        break;
    }
    gameLoop();
  });
}
```

```
}  
  
console.log("Game started. Type PLAY to start playing.");  
gameLoop();
```

## Screenshot Output



```
PS D:\KPL22\KPL_Dewi_Atika_Muthi_2211104042_SE06-02\02_Pengenalan_IDE_dan_Pemrograman_CSharp> node .\unguided1.js  
Game started. Type PLAY to start playing.  
Current State: START  
Enter Command: PLAY  
Current State: PLAYING  
Enter Command: LOSE  
Current State: GAME OVER  
Enter Command: RESTART  
Current State: START  
Enter Command: LOSE  
Current State: START  
Enter Command: PLAY  
Current State: PLAYING  
Enter Command: EXIT  
Exiting game...  
PS D:\KPL22\KPL_Dewi_Atika_Muthi_2211104042_SE06-02\02_Pengenalan_IDE_dan_Pemrograman_CSharp> 
```

## Deskripsi Program

Program ini menerapkan Finite-State Machine (FSM) untuk menangani transisi antara tiga state dalam sebuah game. Program menerima input dari pengguna untuk berpindah state dan berakhir ketika pemain mengetik "EXIT".

### Rincinya:

Program ini menggunakan Finite State Machine (FSM) untuk mengelola alur permainan berbasis teks dengan tiga state utama:

1. START → "PLAY" untuk masuk ke PLAYING, "EXIT" untuk keluar.
2. PLAYING → "LOSE" untuk masuk ke GAME OVER, "EXIT" untuk keluar.
3. GAME OVER → "RESTART" untuk kembali ke START, "EXIT" untuk keluar.

Menggunakan readline untuk menerima input dan rekursi (`gameLoop()`) agar game terus berjalan hingga pemain keluar.



## **BAB III**

### **KESIMPULAN DAN SARAN**

#### **A. KESIMPULAN**

Dari praktikum ini, kita memahami bahwa Automata-based Construction menggunakan pendekatan finite-state machine untuk mengelola status eksekusi program. Sedangkan Table-driven Construction memungkinkan pemrosesan data yang lebih efisien dengan tabel dibandingkan logika pemrograman biasa. Implementasi kedua metode ini dalam JavaScript menunjukkan bagaimana desain program dapat disederhanakan dan dioptimalkan untuk berbagai kasus penggunaan.

#### **B. REFERENSI**

Modul 2: "Automata dan Table-driven Construction".  
McConnell, S. (2004). Code Complete, Second Edition. Microsoft Press.  
Tanenbaum, A. S. (2006). Structured Computer Organization. Pearson.