LAPORAN PRAKTIKUM
KONSTRUKSI PERANGKAT LUNAK

MODUL II

AUTOMATA DAN TABLE-DRIVEN CONSTRUCTION

Disusun Oleh :

Dimas Cahyo Margono

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

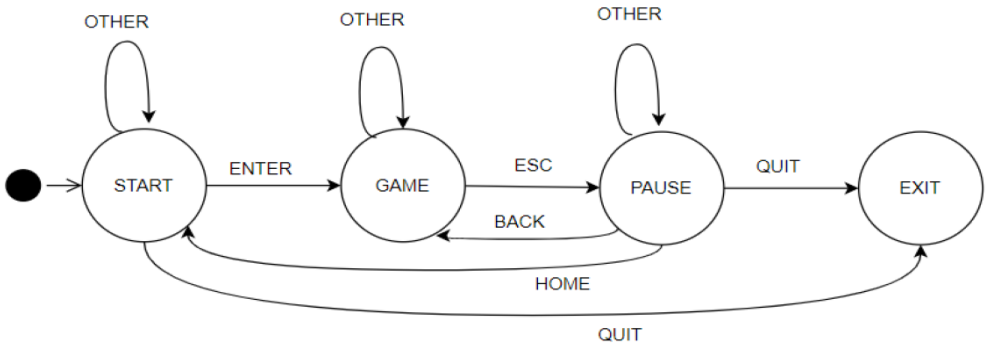
A. DASAR TEORI

2.1 Automata Construction

Automata Construction atau Automata-based programming adalah salah satu paradigma pemrograman dimana program dianggap seperti finite-state machine (FSM) atau formal automaton lainnya yang memiliki berbagai state-state yang saling berkaitan dan memiliki aturan tertentu yang jelas. Berikut ini indikator utama dalam Automata-based programming:

1. Jangka waktu eksekusi program dipisahkan dengan jelas pada state yang ada dan tidak terjadinya eksekusi yang overlaping pada state state yang ada.
2. Semua komunikasi antara state-state yang ada (perpindahan antar state) hanya dapat dilakukan secara eksplisit yang disimpan pada suatu global variable.

2.1.1 Implementasi



Gambar 4. 1 State Diagram.

Gambar diatas merupakan state diagram dari contoh source code Automata-based programming dibawah ini.

```
 using system;
public class Program
{
enum State {START, GAME, PAUSE, EXIT};
public static void Main()
{
State state = State.START;
string[] screenName = {"START", "GAME", "PAUSE",
"EXIT"};
while (state != State.EXIT)
{
```

```
Console.WriteLine(screenName[(int)state] + " SCREEN");
Console.Write("Enter Command : ");
string command = Console.ReadLine();
switch (state)
{
case State.START:
if(command == "ENTER")
state = State.GAME;
else if (command == "QUIT")
state = State.EXIT;
else
state = State.START;
break;
case State.GAME:
if(command == "ESC")
state = State.PAUSE;
else
state = State.GAME;
break;
case State.PAUSE:
if(command == "BACK")
state = State.GAME;
else if (command == "HOME")
state = State.START;
else if (command == "QUIT")
state = State.EXIT;
else
state = State.PAUSE;
break;
}
}
Console.WriteLine("EXIT SCREEN");
}
}
```

2.2 Table-driven Construction

Table-driven Construction adalah skema yang memungkinkan mencari informasi menggunakan table dibandingkan menggunakan logic statements (if dan case) untuk mengetahuinya. Hampir semua hal yang dapat ditangani oleh if dan case, dapat diubah menjadi tabel-driven sebagai gantinya. Dalam kasus sederhana, pernyataan logika lebih mudah dan lebih langsung. Saat logic statements sudah menjadi begitu kompleks, penggunaan table-driven akan menjadi semakin menarik.

Table-driven memiliki dua hal yang perlu diperhatikan sebelum diimplementasikan. Pertama kita perlu menentukan bagaimana mencari entri pada tabel. Kedua beberapa tipe data tidak bisa digunakan untuk mencari entri pada table secara langsung. Secara umum cara pencarian entri pada table-driven dibagi menjadi tiga yaitu :

1. Direct Access

2. Indexed Access
3. Stair-Step Access

2.1.2.1 Direct Access

Secara sederhana, Direct Access adalah metode table-driven yang secara langsung yang mengubah kondisi pada logic statement menjadi key dari table yang digunakan. Untuk lebih jelasnya perhatikan potongan kode berikut.

```
public static int GetDaysPerMonth(int month){
int[] daysPerMonth = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
return daysPerMonth[month - 1];
}
```

2.1.2.2 Indexed Access

Indexed Access adalah metode table-driven yang mengunakan table lain untuk menyimpan index dari table utama, hal ini bertujuan untuk mengurangi penggunan memori penyimpanan dari program apabila ukuran satu dari entri table utama besar dan memiliki nilai entri yang berulang.

2.1.2.3 Stair-step Access

Salah satu metode dalam table-driven yang berguna untuk mempermudah pencarian entri apabila terdapat kasus dimana nilai yang didapatkan berdasarkan range yang telah ada seperti contoh index nilai mahasiswa. Untuk lebih jelasnya perhatikan potongan kode berikut.

```
public static string GetGradeByScore(double studentScore){
string[] grade = { "A", "AB", "B", "BC", "C", "D", "E" };
double[] rangeLimit = { 80.0, 70.0, 65.0, 60.0, 50.0, 40.0, 0.0};
int maxGradeLevel = grade.Length - 1;
string studentGrade = "E";
int gradeLevel = 0;
while((studentGrade == "E") && (gradeLevel < maxGradeLevel)){
if( studentScore > rangeLimit[gradeLevel])
studentGrade = grade[gradeLevel];
gradeLevel = gradeLevel + 1;
}
return studentGrade;
}
```

BAB II
IMPLEMENTASI (GUIDED)

1. App.js

```
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 G:\PRAKTIKUM KPL\pertemuan2> node app.js
START SCREEN
Enter Command: ENTER
GAME SCREEN
Enter Command: ESC
PAUSE SCREEN
Enter Command: HOME
START SCREEN
Enter Command: QUIT
EXIT SCREEN
```

Penjelasan

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

2. Table-driven_Construction.js

```
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(7)); // Output: 28
console.log(getDaysPerMonth(13)); // Output: Invalid month
```

Output

```
PS G:\PRAKTIKUM KPL\pertemuan2> node Table-driven_Construction.js
31
Invalid month
PS G:\PRAKTIKUM KPL\pertemuan2>
```

Penjelasan

- Fungsi getDaysPerMonth mengembalikan jumlah hari dalam bulan yang diberikan.
- Menggunakan array daysPerMonth untuk menyimpan jumlah hari tiap bulan.
- Jika input bulan tidak valid, fungsi mengembalikan "Invalid month".

3. Stair-step_Access.js

```
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(80)); // Output: D
```

Output

```
PS G:\PRAKTIKUM KPL\pertemuan2> node Stair-step_Access.js
AB
A
PS G:\PRAKTIKUM KPL\pertemuan2>
```

Penjelasan

- Fungsi getGradeByScore menentukan nilai berdasarkan skor siswa.
- Menggunakan array grades untuk menyimpan nilai huruf dan rangeLimit sebagai batas skor.

- Loop for mengecek dari batas tertinggi ke terendah untuk menentukan nilai.

- Jika skor lebih besar atau sama dengan rangeLimit[i], maka mengembalikan nilai dari grades[i].

- Jika tidak ada kecocokan, fungsi mengembalikan `"E"`.

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”**.

Sourcecode

```
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(`${state} SCREEN`);
    rl.question("Enter Command: ", (command) => {
        switch (state) {
            case State.START:
                if (command === "PLAY") state = State.PLAYING;
                else if (command === "EXIT") state = State.EXIT;
                break;
            case State.PLAYING:
                if (command === "LOSE") state = State.GAME_OVER;
                break;
            case State.GAME_OVER:
                if (command === "RESTART") state = State.START;
                else if (command === "EXIT") state = State.EXIT;
                break;
        }
        if (state !== State.EXIT) {
            runStateMachine();
        } else {
            console.log("EXIT SCREEN");
            rl.close();
        }
    });
}

runStateMachine();

```

Output

```
PS G:\PRAKTIKUM KPL\pertemuan2> node unguided.js
START SCREEN
Enter Command: PLAY
PLAYING SCREEN
Enter Command: LOSE
GAME OVER SCREEN
Enter Command: RESTART
START SCREEN
Enter Command: EXIT
PLAYING SCREEN
Enter Command: LOSE
GAME OVER SCREEN
Enter Command: RESTART
START SCREEN
Enter Command: EXIT
EXIT SCREEN
PS G:\PRAKTIKUM KPL\pertemuan2>
```

Penjelasan

Kode di atas merupakan implementasi Finite State Machine (FSM) dalam JavaScript untuk mensimulasikan transisi status dalam sebuah permainan sederhana. Program ini menggunakan modul `readline` untuk membaca input dari pengguna di terminal. Terdapat tiga state utama: START, PLAYING, dan GAME OVER, dengan tambahan state EXIT untuk mengakhiri permainan. Transisi antar state terjadi berdasarkan input pengguna, seperti mengetik `"PLAY"` untuk memulai permainan dari state START, `"LOSE"` untuk berpindah ke GAME OVER, dan `"RESTART"` untuk kembali ke START. Program akan terus berjalan dalam loop rekursif hingga pengguna memasukkan `"EXIT"`, yang menyebabkan permainan berhenti dan antarmuka readline ditutup.