Laporan Milestone 1
Tugas Besar
IF2224-Teori Bahasa Formal dan Otomata

Disusun oleh:

Muhammad Alfansya	13523005
Orvin Andika Ikhsan Abhista	13523017
Dzaky Aurelia Fawwaz	13523065
Ferdin Arsenarendra Purtadi	13523117

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2025

I. Landasan Teori

Bahasa pemrograman adalah bahasa yang selalu digunakan untuk melakukan operasi-operasi yang melibatkan komputasi. Terkait bahasa pemrograman ini, proses untuk mengeksekusi bahasa pemrograman tersebut tidaklah mudah. Terdapat banyak proses yang perlu dilakukan oleh *compiler* supaya kode yang ditulis benar dan bisa memberikan output yang sesuai dengan bahasa pemrograman yang ditulis.

1.1 Lexical Analysis

Analisis leksikal (lexical analysis) adalah tahap pertama dalam proses kompilasi atau interpretasi bahasa pemrograman. Pada tahap ini, *compiler* membaca kode sumber mentah yang pada dasarnya hanyalah rangkaian karakter dan mengubahnya menjadi rangkaian satuan makna yang disebut token. Token merupakan unit terkecil yang memiliki arti dalam sebuah program, misalnya kata kunci (keyword), nama variabel (identifier), operator, angka, atau tanda baca.

Tujuan utama dari analisis leksikal adalah untuk mempermudah tahap berikutnya, yaitu parsing. Alih-alih berurusan langsung dengan karakter mentah, tahap parsing bekerja dengan kumpulan token yang sudah terstruktur, sehingga lebih mudah memahami susunan logis dari program. Pada proses ini, analisis leksikal juga membuang bagian yang tidak diperlukan seperti spasi, tab, dan komentar, serta dapat mendeteksi kesalahan seperti simbol yang tidak dikenal sejak awal.

Proses ini dilakukan oleh komponen yang disebut lexer. Lexer membaca kode sumber dari kiri ke kanan dan menggunakan pola tertentu untuk mengenali jenis token yang berbeda-beda. Ketika lexer menemukan urutan karakter yang cocok dengan salah satu pola tersebut, ia membuat sebuah token yang biasanya berisi dua hal utama: jenis token dan nilai token

1.2 Token

Token adalah unit terkecil bermakna dalam bahasa pemrograman, terdiri dari type (jenis token) dan value (nilai token). Dalam Pascal-S, terdapat 18 jenis token utama, seperti yang tercantum dalam spesifikasi tugas besar (lihat Tabel 1.1).

Tabel 1.2.1 Daftar Jenis Token Pascal-S pada Spesifikasi Milestone 1

Daftar Token			
No	Tipe (type)	Nilai (value)	Keterangan
1	**KEYWORD**	`program, var, begin, end, if, then, else, while, do, for, to, downto, integer, real,`	Kata kunci yang sudah didefinisikan oleh bahasa Pascal-S dan memiliki fungsi khusus dalam struktur program.

		`boolean, char, array, of, procedure, function, const, type`	
2	**IDENTIFIER**	`x, y, z, sum, avg, count`	Nama yang didefinisikan oleh pengguna, misalnya nama variabel, prosedur, atau fungsi.
3	**ARITHMETIC_OPERATOR**	`+, -, *, /, div, mod`	-
4	**RELATIONAL_OPERATOR**	`=, <>, <, <=, >, >=`	-
5	**LOGICAL_OPERATOR**	`and, or, not`	-
6	**ASSIGN_OPERATOR**	`:=`	Operator penugasan yang digunakan untuk memberi nilai ke variabel
7	**NUMBER**	`22, 3, 2018`	Bilangan berupa integer atau ril
8	**CHAR_LITERAL**	`'a', 'b', 'c'`	-
9	**STRING_LITERAL**	`'tbfo', 'seru sekali'`	-
10	**SEMICOLON**	`;`	-
11	**COMMA**	`,`	-
12	**COLON**	`:`	-
13	**DOT**	`.`	-
14	**LPARENTHESIS**	`(`	-
15	**RPARENTHESIS**	`)`	-
16	**LBRACKET**	`[`	-
17	**RBRACKET**	`]`	-
18	**RANGE_OPERATOR**	`..`	-

1.3 Lexeme

Lexeme adalah urutan karakter dalam kode sumber yang sesuai dengan pola tertentu dan dikenali sebagai satu token oleh lexer. Misalnya, dalam kode a := 5, lexeme untuk token IDENTIFIER adalah a, untuk token ASSIGN adalah :=, dan untuk token NUMBER adalah 5. Proses pengenalan lexeme dilakukan berdasarkan aturan DFA, yang memetakan urutan karakter ke jenis token yang sesuai.

1.4 DFA

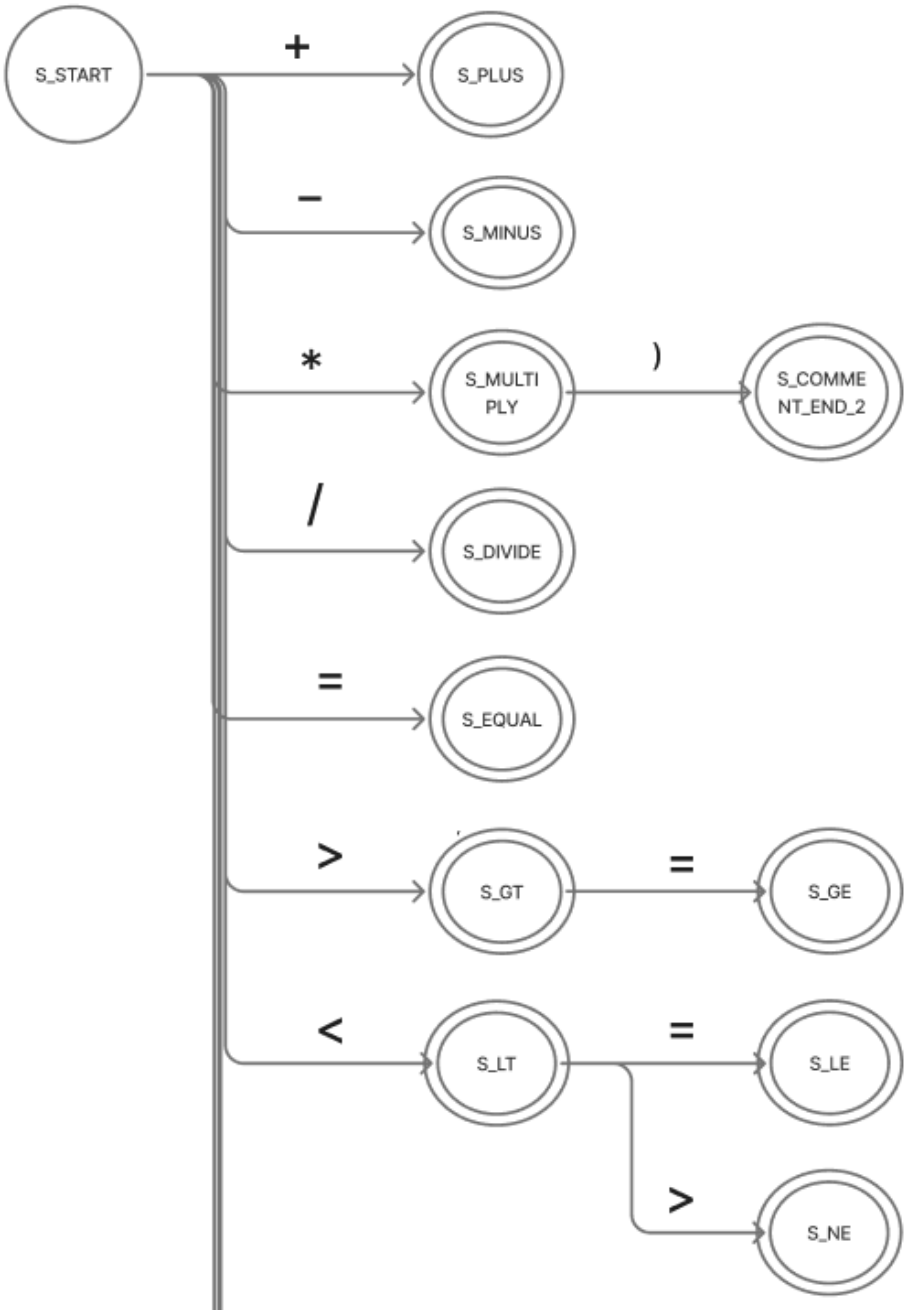
Deterministic Finite Automata (DFA) adalah model komputasi berbasis teori otomata yang digunakan untuk mengenali pola dalam regular language. DFA terdiri dari:

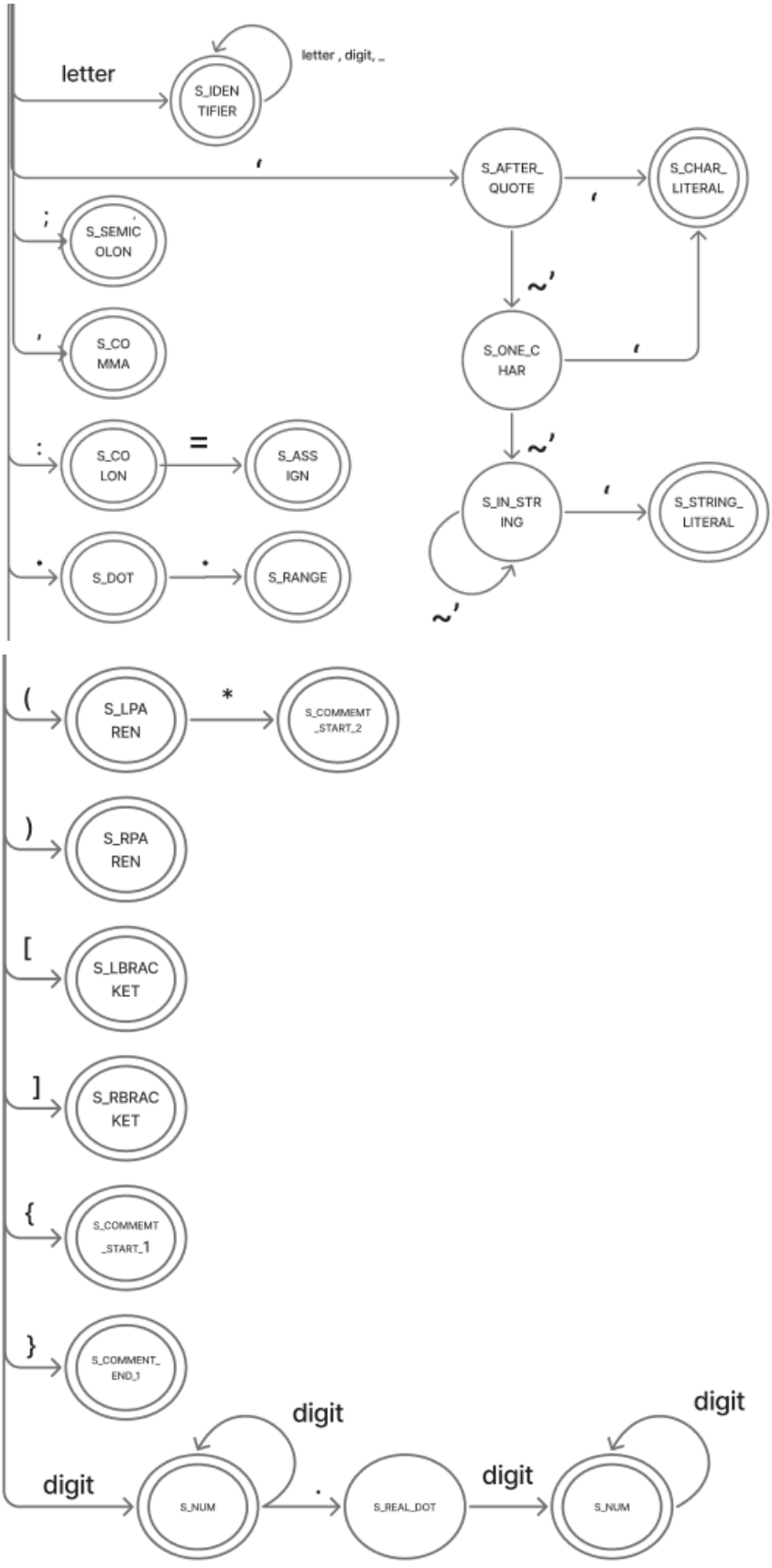
- Himpunan Keadaan (States): Kumpulan keadaan yang mewakili tahap pengenalan karakter.
- Abjad Input (Alphabet): Kumpulan karakter yang mungkin, seperti huruf, angka, dan simbol.
- Fungsi Transisi (Transition Function): Aturan yang menentukan perpindahan dari satu keadaan ke keadaan lain berdasarkan input karakter.
- Keadaan Awal (Start State): Keadaan awal saat lexer mulai memproses kode.
- Himpunan Keadaan Akhir (Final States): Keadaan yang menandakan pengenalan token selesai.

Berbeda dengan Nondeterministic Finite Automata (NFA) yang memungkinkan transisi ganda atau epsilon, DFA memiliki satu transisi pasti untuk setiap input, membuatnya efisien untuk implementasi lexer. Dalam konteks Pascal-S, DFA digunakan untuk mengenali pola token seperti keyword, identifier, dan operator berdasarkan file aturan terpisah (.txt atau .json).

II. Perancangan & Implementasi

2.1. Diagram DFA


S_START
+
S_PLUS
−
S_MINUS
*
S_MULTI PLY
)
S_COMME NT_END_2
/
S_DIVIDE
=
S_EQUAL
>
S_GT
=
S_GE
<
S_LT
=
S_LE
>
S_NE


letter
letter , digit, _
S_IDEN TIFIER
'
S_AFTER_ QUOTE
'
S_CHAR_ LITERAL
;
S_SEMIC OLON
~'
'
S_CO MMA
S_ONE_C HAR
'
~'
:
S_CO LON
=
S_ASS IGN
S_IN_STR ING
'
S_STRING_ LITERAL
~'
.
S_DOT
.
S_RANGE
(
S_LPA REN
*
S_COMMEMT _START_2
)
S_RPA REN
[
S_LBRAC KET
]
S_RBRAC KET
{
S_COMMEMT _START_1
}
S_COMMENT_ END_1
digit
digit
digit
S_NUM
.
S_REAL_DOT
digit
S_NUM

Proses DFA tersebut dimulai dari state awal S_START, di mana setiap karakter pertama yang dibaca akan menentukan jalur transisi berikutnya. Misalnya, jika karakter pertama adalah simbol aritmatika seperti +, -, *, atau /, maka automata akan berpindah ke state yang mewakili operator tersebut dan langsung mengenali tokennya. Jika karakter pertama adalah tanda relasional seperti < atau >, DFA akan menelusuri transisi tambahan untuk membedakan antara operator tunggal (<, >) dan operator ganda (<=, >=, <>).

Ketika DFA membaca huruf, ia berpindah ke state yang mengenali identifier atau keyword. Selama karakter berikutnya masih berupa huruf atau digit, automata tetap berada di state tersebut. Setelah menemukan karakter yang bukan huruf atau digit, token dihasilkan. Token tersebut bisa berupa identifier (misalnya nama variabel) atau kata kunci (seperti begin, if, while), tergantung hasil pencocokan terhadap lookup table yang berisi daftar keyword. Jika karakter pertama yang dibaca adalah digit, DFA akan berpindah ke state number, di mana ia akan tetap di sana selama karakter berikutnya masih berupa digit. Hasil akhirnya adalah token bertipe number literal

DFA juga mengenali string literal yang diawali dan diakhiri dengan tanda kutip. Setelah memasuki state string, automata akan terus membaca karakter hingga menemukan tanda kutip penutup yang sama, lalu menghasilkan token bertipe string literal. Di sisi lain, simbol-simbol tunggal seperti (,), [,], ;, ,, dan . juga memiliki state tersendiri yang langsung menghasilkan token setelah satu karakter dibaca.

2.2. Implementasi

Penerjemahan DFA ke dalam JSON

Berikut file rules (lexical_rules.json) yang dibuat berdasarkan diagram DFA yang telah dibuat :

```
{
  "start_state": "S_START",
  "char_classes": {
    "letter":
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_",
    "digit": "0123456789",
    "whitespace": " \t\n\r"
  },
  "states": {
    "S_START": {
      "is_final": false,
      "transitions": [
        { "class": "whitespace", "next_state": "S_START" },
        { "class": "letter", "next_state": "S_IDENTIFIER" },
        { "class": "digit", "next_state": "S_NUM" },
        { "chars": "+", "next_state": "S_PLUS" },
        { "chars": "-", "next_state": "S_MINUS" },
        { "chars": "*", "next_state": "S_MULTI" },
        { "chars": "/", "next_state": "S_DIVIDE" },
        { "chars": "=", "next_state": "S_EQUAL" },
```

```
        { "chars": "<", "next_state": "S_LT" },
        { "chars": ">", "next_state": "S_GT" },
        { "chars": ":", "next_state": "S_COLON" },
        { "chars": ";", "next_state": "S_SEMICOLON" },
        { "chars": ",", "next_state": "S_COMMA" },
        { "chars": ".", "next_state": "S_DOT" },
        { "chars": "'", "next_state": "S_STRING" },
        { "chars": "(", "next_state": "S_LPAREN" },
        { "chars": ")", "next_state": "S_RPAREN" },
        { "chars": "[", "next_state": "S_LBRACKET" },
        { "chars": "]", "next_state": "S_RBRACKET" },
        { "chars": "{", "next_state": "S_LBRACE" },
        { "chars": "}", "next_state": "S_RBRACE" }
      ]
    },

    "S_IDENTIFIER": {
      "is_final": true,
      "token_type": "IDENTIFIER",
      "transitions": [
        { "class": "letter", "next_state": "S_IDENTIFIER" },
        { "class": "digit", "next_state": "S_IDENTIFIER" }
      ]
    },

    "S_NUM": {
      "is_final": true,
      "token_type": "NUMBER",
      "transitions": [
        { "class": "digit", "next_state": "S_NUM" },
        { "chars": ".", "next_state": "S_REAL_DOT" }
      ]
    },
    "S_REAL_DOT": {
      "is_final": false,
      "transitions": [
        { "class": "digit", "next_state": "S_REAL_AFTER_DOT" }
      ]
    },
    "S_REAL_AFTER_DOT": {
      "is_final": true,
      "token_type": "NUMBER",
      "transitions": [
        { "class": "digit", "next_state": "S_REAL_AFTER_DOT" }
      ]
    },

    "S_PLUS": { "is_final": true, "token_type":
"ARITHMETIC_OPERATOR", "transitions": [] },
    "S_MINUS": { "is_final": true, "token_type":
"ARITHMETIC_OPERATOR", "transitions": [] },
```

```
    "S_MULTI": { "is_final": true, "token_type":
"ARITHMETIC_OPERATOR", "transitions": [] },
    "S_DIVIDE": { "is_final": true, "token_type":
"ARITHMETIC_OPERATOR", "transitions": [] },
    "S_EQUAL": { "is_final": true, "token_type":
"RELATIONAL_OPERATOR", "transitions": [] },
    "S_SEMICOLON": { "is_final": true, "token_type": "SEMICOLON",
"transitions": [] },
    "S_COMMA": { "is_final": true, "token_type": "COMMA",
"transitions": [] },
    "S_RPAREN": { "is_final": true, "token_type": "RPARENTHESIS",
"transitions": [] },
    "S_LBRACKET": { "is_final": true, "token_type": "LBRACKET",
"transitions": [] },
    "S_RBRACKET": { "is_final": true, "token_type": "RBRACKET",
"transitions": [] },
    "S_RBRACE": { "is_final": true, "token_type": "COMMENT_END",
"transitions": [] },
    "S_LBRACE": {
        "is_final": false,
        "transitions": [
            { "chars": "}", "next_state": "S_COMMENT_BRACE_END" },
            { "default": true, "next_state": "S_LBRACE" }
        ]
    },
    "S_COMMENT_BRACE_END": { "is_final": true, "token_type":
"COMMENT", "transitions": [] },

    "S_COLON": {
      "is_final": true,
      "token_type": "COLON",
      "transitions": [
        { "chars": "=", "next_state": "S_ASSIGN" }
      ]
    },
    "S_ASSIGN": { "is_final": true, "token_type":
"ASSIGN_OPERATOR", "transitions": [] },

    "S_LT": {
      "is_final": true,
      "token_type": "RELATIONAL_OPERATOR",
      "transitions": [
        { "chars": "=", "next_state": "S_LE" },
        { "chars": ">", "next_state": "S_NE" }
      ]
    },
    "S_LE": { "is_final": true, "token_type":
"RELATIONAL_OPERATOR", "transitions": [] },
    "S_NE": { "is_final": true, "token_type":
"RELATIONAL_OPERATOR", "transitions": [] },
```

```
    "S_GT": {
      "is_final": true,
      "token_type": "RELATIONAL_OPERATOR",
      "transitions": [
        { "chars": "=", "next_state": "S_GE" }
      ]
    },
    "S_GE": { "is_final": true, "token_type":
"RELATIONAL_OPERATOR", "transitions": [] },

    "S_DOT": {
        "is_final": true,
        "token_type": "DOT",
        "transitions": [
            { "chars": ".", "next_state": "S_RANGE" }
        ]
    },
    "S_RANGE": { "is_final": true, "token_type": "RANGE_OPERATOR",
"transitions": [] },

    "S_STRING": {
      "is_final": false,
      "transitions": [
        { "chars": "'", "next_state": "S_STRING_END" },
        { "default": true, "next_state": "S_STRING" }
      ]
    },
    "S_STRING_END": { "is_final": true, "token_type":
"STRING_LITERAL", "transitions": [] },

    "S_LPAREN": {
      "is_final": true,
      "token_type": "LPARENTHESIS",
      "transitions": [
        { "chars": "*", "next_state": "S_COMMENT_STAR_START" }
      ]
    },
    "S_COMMENT_STAR_START": {
        "is_final": false,
        "transitions": [
            { "chars": "*", "next_state":
"S_COMMENT_STAR_ATTEMPT_END"},
            { "default": true, "next_state": "S_COMMENT_STAR_START"
}
        ]
    },
    "S_COMMENT_STAR_ATTEMPT_END": {
        "is_final": false,
        "transitions": [
            { "chars": ")", "next_state": "S_COMMENT_STAR_END" },
            { "default": true, "next_state": "S_COMMENT_STAR_START"
```

```
}
        ]
    },

    "S_COMMENT_STAR_END": { "is_final": true, "token_type":
"COMMENT", "transitions": [] },

    "S_COMMENT_MULTI": {
      "is_final": false,
      "transitions": [
        { "chars": "*", "next_state": "S_COMMENT_MULTI_END" },
        { "default": true, "next_state": "S_COMMENT_MULTI" }
      ]
    },
    "S_COMMENT_MULTI_END": {
      "is_final": false,
      "transitions": [
        { "chars": ")", "next_state": "S_START" },
        { "default": true, "next_state": "S_COMMENT_MULTI" }
      ]
    }
  }
}
```

Selanjutnya dari rules ini, dibuat sebuah program yang akan membaca file input (.pas) dengan memanfaatkan rules yang tersedia di lexical_rules.json.

```
class Lexer:
    def __init__(self, rules_file):
        """Memuat aturan DFA dari file JSON."""
        self.rules = self._load_rules(rules_file)
        self.start_state = self.rules['start_state']
        self.char_classes = self.rules['char_classes']
        self.states = self.rules['states']

    def _load_rules(self, rules_file):
        try:
            with open(rules_file, 'r') as f:
                return json.load(f)
        except FileNotFoundError:
            print(f"Error: File aturan '{rules_file}' tidak
ditemukan.")
            sys.exit(1)

    def _find_next_state(self, current_state_name, char):
        state_info = self.states[current_state_name]

        for transition in state_info.get('transitions', []):
            if 'class' in transition:
```

```
                if char in self.char_classes[transition['class']]:
                    return transition['next_state']
            elif 'chars' in transition:
                if char in transition['chars']:
                    return transition['next_state']
            elif 'default' in transition:
                return transition['next_state']

        return None

    def tokenize(self, source_code):
        tokens = []
        current_pos = 0

        while current_pos < len(source_code):
            if source_code[current_pos] in
self.char_classes['whitespace']:
                current_pos += 1
                continue

            current_state_name = self.start_state
            last_accepted_state = None
            last_accepted_pos = -1

            temp_pos = current_pos
            while temp_pos < len(source_code):
                char = source_code[temp_pos]
                next_state_name =
self._find_next_state(current_state_name, char)
                if next_state_name is None:
                    break

                current_state_name = next_state_name
                temp_pos += 1

                if self.states[current_state_name].get('is_final',
False):
                    last_accepted_state = current_state_name
                    last_accepted_pos = temp_pos

            if last_accepted_state:
                token_type =
self.states[last_accepted_state]['token_type']
                lexeme = source_code[current_pos:last_accepted_pos]

                if token_type == 'COMMENT':
                    comment_lexeme =
source_code[current_pos:last_accepted_pos]
                    if comment_lexeme.startswith('{'):
                        tokens.append(('COMMENT_START', '{'))
                        tokens.append(('COMMENT_END', '}'))
```

```
                    elif comment_lexeme.startswith('(*'):
                        tokens.append(('COMMENT_START', '(*'))
                        tokens.append(('COMMENT_END', '*)'))

                    current_pos = last_accepted_pos
                    continue

                if token_type == 'IDENTIFIER':
                    token_type = LOOKUP_TABLE.get(lexeme.lower(),
'IDENTIFIER')

                tokens.append((token_type, lexeme))
                current_pos = last_accepted_pos
            else:
                unknown_char = source_code[current_pos]
                print(f"Error: Karakter tidak dikenal ->
'{unknown_char}' di posisi {current_pos}")
                tokens.append(('UNKNOWN', unknown_char))
                current_pos += 1

        tokens.append(('EOF', ''))
        return tokens
```

Class Lexer merupakan komponen inti dari lexical analyzer yang mengimplementasikan Deterministic Finite Automata (DFA) untuk melakukan tokenisasi kode Pascal-S. Pada saat inisialisasi, class ini memuat aturan DFA dari file JSON eksternal yang berisi definisi start state, char classes (kelas karakter seperti huruf, digit, dan whitespace), serta seluruh state beserta transisinya. Method _load_rules bertanggung jawab untuk membaca file aturan DFA dan menangani error jika file tidak ditemukan. Method _find_next_state berfungsi untuk menentukan state berikutnya berdasarkan karakter input saat ini, dengan memeriksa apakah karakter tersebut cocok dengan class tertentu, karakter spesifik, atau menggunakan transisi default.

Method utama tokenize melakukan proses scanning karakter demi karakter pada source code menggunakan algoritma longest match (maximal munch). Proses ini dimulai dari start state dan terus membaca karakter hingga tidak ada transisi yang valid lagi, sambil mencatat state final terakhir yang diterima. Setelah menemukan token yang valid, lexer menentukan tipe token berdasarkan state final yang dicapai. Untuk membedakan antara keyword dan identifier, lexer memanfaatkan lookup table yang berisi daftar keyword dan operator reserved dalam Pascal-S. Identifier yang cocok dengan entri di lookup table akan diklasifikasikan ulang sebagai keyword atau operator, sementara yang tidak cocok tetap sebagai identifier biasa. Lexer juga menangani kasus khusus seperti komentar (yang menghasilkan token COMMENT_START dan COMMENT_END) serta karakter yang tidak dikenal dengan memberikan pesan error yang informatif.

Berikut adalah Lookup Table yang digunakan :

```
LOOKUP_TABLE = {
    "program": "KEYWORD",
```

```
    "var": "KEYWORD",
    "begin": "KEYWORD",
    "end": "KEYWORD",
    "if": "KEYWORD", "then": "KEYWORD", "else": "KEYWORD", "while":
"KEYWORD",
    "do": "KEYWORD", "for": "KEYWORD", "to": "KEYWORD", "downto":
"KEYWORD",
    "integer": "KEYWORD", "real": "KEYWORD", "boolean": "KEYWORD",
"char": "KEYWORD",
    "array": "KEYWORD", "of": "KEYWORD", "procedure": "KEYWORD",
"function": "KEYWORD",
    "const": "KEYWORD", "type": "KEYWORD",
    "div": "ARITHMETIC_OPERATOR", "mod": "ARITHMETIC_OPERATOR",
    "and": "LOGICAL_OPERATOR", "or": "LOGICAL_OPERATOR", "not":
"LOGICAL_OPERATOR"
}
```

III. Pengujian

3.1 Pengujian 1 : Program Dasar dengan Token Standar

Input
test > milestone-1 > pengujian1.pas 1 program Hello; 2 var 3 a, b: integer; 4 begin 5 a := 5; 6 b := a + 10; 7 writeln('Result = ', b); 8 end.
Output

Input

```
PS C:\Users\ASUS\tbfo\JWA-Tubes-IF2224> python src/main.py
Masukkan nama file Pascal (.pas): pengujian1
--- Hasil Tokenisasi untuk pengujian1.pas ---
KEYWORD(program)
IDENTIFIER(Hello)
SEMICOLON(;)
KEYWORD(var)
IDENTIFIER(a)
COMMA(,)
IDENTIFIER(b)
COLON(:)
KEYWORD(integer)
SEMICOLON(;)
KEYWORD(begin)
IDENTIFIER(a)
ASSIGN_OPERATOR(:=)
NUMBER(5)
SEMICOLON(;)
IDENTIFIER(b)
ASSIGN_OPERATOR(:=)
IDENTIFIER(a)
ARITHMETIC_OPERATOR(+)
NUMBER(10)
SEMICOLON(;)
IDENTIFIER(writeln)
LPARENTHESIS(()
STRING_LITERAL('Result = ')
COMMA(,)
IDENTIFIER(b)
RPARENTHESIS())
SEMICOLON(;)
KEYWORD(end)
DOT(.)
EOF()
----------------------------------------
```

Penjelasan

Lexer berhasil mengenali semua jenis token dasar yang diharapkan, termasuk KEYWORD, IDENTIFIER, NUMBER, ASSIGN_OPERATOR, ARITHMETIC_OPERATOR, SEMICOLON, LPARENTHESIS, RPARENTHESIS, COMMA, STRING_LITERAL, dan DOT.
Token EOF() menunjukkan akhir file, yang sesuai dengan spesifikasi untuk menandai penyelesaian proses scanning.
Whitespace dan tata letak kode diabaikan dengan benar, sesuai fungsi lexer untuk menyederhanakan input menjadi daftar token.

3.2 Pengujian 2 : Penggunaan Operator Kompleks dan Logika

Input

test > milestone-1 > pengujian2.pas

```
1  program LogicTest;
2  var
3    x, y: boolean;
4  begin
5    x := true;
6    y := not x and (x or false);
7    writeln('Logic = ', y);
8  end.
```

Output

Input

```
PS C:\Users\ASUS\tbfo\JWA-Tubes-IF2224> python src/main.py
Masukkan nama file Pascal (.pas): pengujian2
--- Hasil Tokenisasi untuk pengujian2.pas ---
KEYWORD(program)
IDENTIFIER(LogicTest)
SEMICOLON(;)
KEYWORD(var)
IDENTIFIER(x)
COMMA(,)
IDENTIFIER(y)
COLON(:)
KEYWORD(boolean)
SEMICOLON(;)
KEYWORD(begin)
IDENTIFIER(x)
ASSIGN_OPERATOR(:=)
IDENTIFIER(true)
SEMICOLON(;)
IDENTIFIER(y)
ASSIGN_OPERATOR(:=)
LOGICAL_OPERATOR(not)
IDENTIFIER(x)
LOGICAL_OPERATOR(and)
LPARENTHESIS(()
IDENTIFIER(x)
LOGICAL_OPERATOR(or)
IDENTIFIER(false)
RPARENTHESIS())
SEMICOLON(;)
IDENTIFIER(writeln)
LPARENTHESIS(()
STRING_LITERAL('Logic = ')
COMMA(,)
IDENTIFIER(y)
RPARENTHESIS())
SEMICOLON(;)
KEYWORD(end)
DOT(.)
EOF()
----------------------------------------
```

Penjelasan

Lexer berhasil mengenali token kompleks seperti LOGICAL_OPERATOR (not, and, or), IDENTIFIER (true, false), dan kombinasi ekspresi logika dalam tanda kurung.

Token KEYWORD(boolean) dikenali dengan benar sebagai tipe data, menunjukkan dukungan lexer untuk tipe data boolean sesuai spesifikasi Pascal-S .

Struktur ekspresi not x and (x or false) dipecah menjadi token individual dengan urutan yang tepat, menunjukkan bahwa DFA mengenali pola logika dengan baik. Tidak ada kesalahan leksikal, dan EOF() menandakan penyelesaian proses scannin

3.3 Pengujian 3: Array dan Range Operator

Input

test > milestone-1 > pengujian3.pas

```
1  program ArrayTest;
2  var
3    arr: array[1..5] of integer;
4  begin
5    arr[1] := 10;
6    writeln('Array[1] = ', arr[1]);
7  end.
```

Output

```
PS C:\Users\ASUS\tbfo\JWA-Tubes-IF2224> python src/main.py
Masukkan nama file Pascal (.pas): pengujian3
--- Hasil Tokenisasi untuk pengujian3.pas ---
KEYWORD(program)
IDENTIFIER(ArrayTest)
SEMICOLON(;)
KEYWORD(var)
IDENTIFIER(arr)
COLON(:)
KEYWORD(array)
LBRACKET([)
NUMBER(1)
RANGE_OPERATOR(..)
NUMBER(5)
RBRACKET(])
KEYWORD(of)
KEYWORD(integer)
SEMICOLON(;)
KEYWORD(begin)
IDENTIFIER(arr)
LBRACKET([)
NUMBER(1)
RBRACKET(])
ASSIGN_OPERATOR(:=)
NUMBER(10)
SEMICOLON(;)
IDENTIFIER(writeln)
LPARENTHESIS(()
STRING_LITERAL('Array[1] = ')
COMMA(,)
IDENTIFIER(arr)
LBRACKET([)
NUMBER(1)
RBRACKET(])
RPARENTHESIS())
SEMICOLON(;)
KEYWORD(end)
DOT(.)
EOF()
```

Penjelasan

Lexer berhasil mengenali semua token yang relevan, termasuk KEYWORD (program, var, array, of, integer, begin, end), IDENTIFIER (ArrayTest, arr), NUMBER (1, 5, 10), RANGE_OPERATOR (..), LBRACKET ([), RBRACKET (]), ASSIGN_OPERATOR (:=), SEMICOLON (;), LPARENTHESIS (), RPARENTHESIS ()), COMMA (,), STRING_LITERAL ('Array[1] = '), dan DOT (.).
Khususnya, RANGE_OPERATOR(..) dikenali dengan benar dalam konteks deklarasi array [1..5], yang sesuai dengan revisi 3 spesifikasi yang menetapkan bahwa RANGE_OPERATOR tetap diimplementasikan.
Penggunaan indeks array arr[1] dipecah menjadi IDENTIFIER(arr), LBRACKET([), NUMBER(1), dan RBRACKET(]), menunjukkan bahwa lexer dapat menangani akses array dengan tepat.
Token EOF() menandakan akhir file, yang konsisten dengan hasil pengujian sebelumnya dan sesuai spesifikasi.

3.4 Pengujian 4: Komentar dan Error Handling

Input

test > milestone-1 > pengujian4.pas

```
1  program ErrorTest;
2  var
3    a: integer;
4  begin
5    a := 5; { This is a comment }
6    b := a + $; { Invalid symbol }
7    writeln(a);
8  end.
```

Output

```
PS C:\Users\ASUS\tbfo\JWA-Tubes-IF2224> python src/main.py
Masukkan nama file Pascal (.pas): pengujian4
Error: Karakter tidak dikenal -> '$' di posisi 86
--- Hasil Tokenisasi untuk pengujian4.pas ---
KEYWORD(program)
IDENTIFIER(ErrorTest)
SEMICOLON(;)
KEYWORD(var)
IDENTIFIER(a)
COLON(:)
KEYWORD(integer)
SEMICOLON(;)
KEYWORD(begin)
IDENTIFIER(a)
ASSIGN_OPERATOR(:=)
NUMBER(5)
SEMICOLON(;)
COMMENT_START({)
COMMENT_END(})
IDENTIFIER(b)
ASSIGN_OPERATOR(:=)
IDENTIFIER(a)
ARITHMETIC_OPERATOR(+)
UNKNOWN($)
SEMICOLON(;)
COMMENT_START({)
COMMENT_END(})
IDENTIFIER(writeln)
LPARENTHESIS(()
IDENTIFIER(a)
RPARENTHESIS())
SEMICOLON(;)
KEYWORD(end)
DOT(.)
EOF()
----------------------------------------
```

Penjelasan

Lexer berhasil mendeteksi simbol tidak dikenal ($) dan melaporkan error dengan pesan "Error: Karakter tidak dikenal -> '$' di posisi 86". Posisi 86 menunjukkan lokasi karakter bermasalah dalam file, yang sesuai dengan spesifikasi untuk memberikan informasi error yang informatif. Lexer tidak crash dan melanjutkan proses tokenisasi setelah error, yang menunjukkan penanganan error yang baik.

Token COMMENT_START({) dan COMMENT_END(}) muncul dua kali, yang mencerminkan keberadaan dua blok komentar dalam kode ({ This is a comment } dan { Invalid symbol })

3.5 Pengujian 5 : Nilai Negatif dan String Kosong

Input

test > milestone-1 > pengujian5.pas

```
1  program NegativeTest;
2  var
3    a: integer;
4  begin
5    a := -5;
6    writeln('', a);
7  end.
```

Output

```
PS C:\Users\ASUS\tbfo\JWA-Tubes-IF2224> python src/main.py
Masukkan nama file Pascal (.pas): pengujian5
--- Hasil Tokenisasi untuk pengujian5.pas ---
KEYWORD(program)
IDENTIFIER(NegativeTest)
SEMICOLON(;)
KEYWORD(var)
IDENTIFIER(a)
COLON(:)
KEYWORD(integer)
SEMICOLON(;)
KEYWORD(begin)
IDENTIFIER(a)
ASSIGN_OPERATOR(:=)
ARITHMETIC_OPERATOR(-)
NUMBER(5)
SEMICOLON(;)
IDENTIFIER(writeln)
LPARENTHESIS(()
STRING_LITERAL('')
COMMA(,)
IDENTIFIER(a)
RPARENTHESIS())
SEMICOLON(;)
KEYWORD(end)
DOT(.)
EOF()
----------------------------------------
```

Penjelasan

Token NUMBER(-5) menunjukkan bahwa lexer dapat menangani angka negatif dengan benar, mengenali tanda negatif (-) sebagai bagian dari nilai numerik, bukan operator terpisah. Ini menunjukkan implementasi DFA yang efektif untuk pola angka, termasuk negatif.

IV. Kesimpulan dan saran

4.1 Kesimpulan

Berdasarkan pelaksanaan Milestone 1 tugas besar mata kuliah IF2224 Formal Language and Automata Theory, implementasi lexical analysis untuk compiler Pascal-S telah berhasil dikembangkan dengan memanfaatkan Deterministic Finite Automata (DFA). Proses pengujian yang mencakup enam kasus unik menunjukkan bahwa lexer dapat mengenali berbagai jenis token, termasuk keyword, identifier, number (termasuk negatif), string literal (termasuk kosong), operator aritmatika, logika, relasional, assign, serta separator dengan tepat. Pengujian juga membuktikan kemampuan lexer untuk menangani struktur program dasar, ekspresi logika kompleks, deklarasi array dengan range operator, dan akses elemen array.

Lebih lanjut, lexer menunjukkan kemampuan error handling yang baik dengan mendeteksi simbol tidak dikenal (seperti $) dan melaporkan posisi error secara informatif, sekaligus melanjutkan proses tokenisasi. Hasil ini mencerminkan kesesuaian dengan spesifikasi revisi 3 dan 4, khususnya implementasi RANGE_OPERATOR dan penyesuaian token. Secara keseluruhan, implementasi lexer telah memenuhi tujuan awal untuk mengubah kode sumber Pascal-S menjadi daftar token yang dapat digunakan pada tahap parsing berikutnya, dengan performa yang andal untuk kasus uji yang telah ditentukan.

4.2 Saran

Berdasarkan evaluasi implementasi dan hasil pengujian, beberapa saran dapat diajukan untuk perbaikan dan pengembangan lebih lanjut:

1. Pengujian Kasus Ekstrem: Tambahkan pengujian untuk kasus ekstrem, seperti input dengan karakter khusus lainnya (misalnya #, @) atau kombinasi token yang sangat panjang, untuk memastikan ketahanan lexer terhadap input tidak terduga.
2. Optimalisasi Error Recovery: Tingkatkan mekanisme error recovery agar lexer dapat melanjutkan scanning lebih efisien setelah mendeteksi error, misalnya dengan melewatkan karakter tidak dikenal hingga menemukan token valid berikutnya.

V. Referensi

Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). Compilers: Principles, Techniques,
and Tools (2nd ed.). Pearson Education.

Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2007). Introduction to Automata Theory, Languages, and Computation (3rd ed.). Pearson Education.

Institut Teknologi Bandung. Spesifikasi Tugas Besar IF2224 - Formal Language and Automata Theory, Milestone 1. https://docs.google.com/document/d/1w0GmHW5L0gKZQWbgmtJPFmOzlpSWBknNPdugucn4eII/edit?tab=t.0

Lampiran

- Link Repository Github.
  https://github.com/orvin14/JWA-Tubes-IF2224
- Link workspace diagram.
  https://www.figma.com/board/trLGfHO2mLq6atIVYz2mIF/Untitled?node-id=0-1&t=nPlUVxdEHLeSlnKW-1

- Pembagian Tugas (menunjukan persentase kontribusi).

Anggota	Pembagian Tugas
Muhammad Alfansya (13523005)	DFA, laporan
Orvin Andika Ikhsan Abhista (13523017)	DFA, Laporan
Dzaky Aurelia Fawwaz (13523065)	DFA, Testing, Laporan
Ferdin Arsenarendra Purtadi (13523117)	DFA, laporan, implementasi