LAPORAN MILESTONE 1
IF2224 TEORI BAHASA FORMAL DAN OTOMATA
LEXICAL ANALYSIS

Kelompok 6 DFAntastic (DFC)

Mayla Yaffa Ludmilla 13523050
Anella Utari Gunadi 13523078
Muhammad Edo Raduputu Aprima 13523096
Athian Nugraha Muarajuang 13523106

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

DAFTAR ISI

I. DESKRIPSI TUGAS

Pada milestone ini, kami diminta untuk membuat **tahapan pertama dari compiler**, yaitu lexer atau **lexical analyzer**. Lexer berfungsi melakukan analisis leksikal (lexical analysis) dengan menggunakan Deterministic Finite Automata (**DFA**) untuk mengenali pola karakter dalam source code.

Tahapan ini bertujuan untuk mengubah source code Pascal-S dari kumpulan karakter mentah menjadi unit-unit bermakna yang disebut *token*. Setiap token memiliki jenis (type) dan nilai (value), serta akan menjadi masukan bagi tahap berikutnya dalam proses kompilasi.

Masukan	Kode Pascal-S (.pas)
Keluaran	List Token (misalnya VAR(X), ASSIGN(=), OPERATOR(+))
Otomata	Deterministic finite automata (DFA)

Program harus menggunakan DFA, dengan **file aturan DFA disimpan dalam format .txt atau .json** dan dibaca oleh program. Proses scanning dilakukan huruf demi huruf pada source code Pascal-S.

Input Command (contoh yang digunakan menggunakan Python) Format: python [Compiler] [Kode Pascal]
python compiler.py program.pas
Contoh file aturan DFA (harus dengan format [State Awal] [Input] [State Selanjutnya] dan ada pendefinisian start state dan final state)
DEFINE THE START STATE AND FINAL STATE Start_state = State_0 Final_state = State_x, State_y, State_z State_0 a State_1

State_1 b State_1

.

.

.

Masukan (input)

```
program Hello;


var
  a, b: integer;


begin
  a := 5;
  b := a + 10;
  writeln('Result = ', b);
end.
```

Proses (State DFA belum tentu benar)

p => char => State 1

r => char => State 1

.

m => char => State 1

=> char => State 2 => Gotten: KEYWORD(program)

.

H => char => State 3

.

; => char => State 0 => Gotten: IDENTIFIER(Hello) & SEMICOLON

.

.

.

Keluaran (output)

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(.)
```

Selain program, kami juga diminta untuk membuat **diagram transisi DFA** yang sesuai dengan aturan pada file tersebut. Diagram dapat dibuat menggunakan alat apa pun (misalnya draw.io) dan diserahkan dalam format **.pdf** beserta **tautan workspace**-nya.

II. LANDASAN TEORI

2.1. Finite Automata

Finite Automata merupakan mesin abstrak yang digunakan untuk mengenali pola pada suatu urutan masukan (*input sequence*). Konsep ini menjadi dasar penting dalam memahami *regular language* atau bahasa regular dalam ilmu komputer.

Otomata bekerja dengan memproses setiap simbol masukan secara bertahap, berpindah dari satu state (keadaan) ke state lainnya melalui transisi yang telah ditentukan Setelah seluruh masukan selesai diproses, sistem akan memeriksa posisi akhirnya.Jika mesin berhenti pada accepting state (keadaan penerima), maka masukan tersebut dianggap diterima. Sebaliknya, jika berhenti di state lain, masukan ditolak.

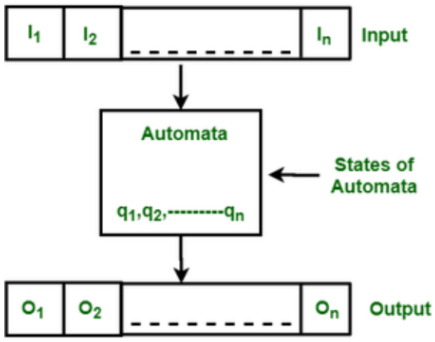
Secara umum, terdapat dua jenis finite automata:

1. **Deterministic Finite Automata (DFA)**: setiap masukan dan *state* hanya memiliki satu transisi yang mungkin
2. **Non-Deterministic Automata (NFA)**: sebuah *state* dapat memiliki lebih dari satu kemungkinan transisi untuk simbol yang sama.

Meskipun berbeda dalam cara kerja, DFA dan NFA memiliki kemampuan pengenalan bahasa yang sama, yaitu sama-sama dapat mengenali himpunan bahasa regular.

Ciri-ciri Finite Automata

1. Input: kumpulan simbol atau karakter yang diberikan ke mesin.
2. Output: hasil akhir berupa keputusan “diterima” atau “ditolak” terhadap pola masukan.
3. State (keadaan): kondisi atau konfigurasi mesin pada suatu waktu tertentu.
4. State Relation (relasi antar-keadaan): aturan transisi yang menentukan perpindahan dari satu *state* ke *state* lain berdasarkan simbol masukan.
5. Output Relation (relasi keluaran): keputusan akhir berdasarkan *state* tempat mesin berhenti setelah semua masukan diproses.



Gambar 2.1.1 FItur pada Finite Automata
Sumber: https://www.geeksforgeeks.org/theory-of-computation/introduction-of-finite-automata/

Definisi Formal Finite Automata

Sebuah *finite automaton* dapat didefinisikan sebagai 5-tuple berikut:

$$M = (Q, \Sigma, q0, F, \delta)$$

dengan:

- Q : himpunan hingga dari *state* (keadaan)
- Σ : himpunan simbol masukan (*input alphabet*)
- $q0$: *start state* atau *state* awal
- F : himpunan *final state* atau *accepting state*
- δ : fungsi transisi yang menentukan perpindahan antar *state* berdasarkan simbol masukan

2.2. Deterministic Finite Automata (DFA)

Sebuah Deterministic Finite Automata (DFA) direpresentasikan dalam bentuk himpunan lima unsur (5-tuple) sebagai berikut:

$$M = (Q, \Sigma, q_0, F, \delta)$$

Pada DFA, setiap simbol masukan (input symbol) hanya memiliki satu transisi yang pasti menuju satu *state* tertentu. Dengan kata lain, untuk setiap pasangan (*state*, *input*), hanya terdapat satu kemungkinan arah perpindahan.

Berbeda dengan NFA, DFA tidak memperbolehkan transisi kosong (ε-transition). Artinya, setiap *state* harus memiliki definisi transisi yang lengkap untuk seluruh simbol dalam alfabet masukan (*input alphabet*).

Secara formal, DFA terdiri atas lima komponen berikut:

- Q : Himpunan hingga dari semua *state* yang mungkin.
- Σ : Himpunan simbol masukan yang dapat dibaca oleh mesin.
- q_0 : *State* awal tempat mesin memulai proses.
- F : Himpunan *final state* atau *accepting state*, yaitu keadaan di mana input diterima.
- δ : Fungsi transisi yang didefinisikan sebagai

$$\delta : Q \times \Sigma \rightarrow Q$$

Fungsi ini menentukan perpindahan dari satu *state* ke *state* lain berdasarkan simbol masukan.

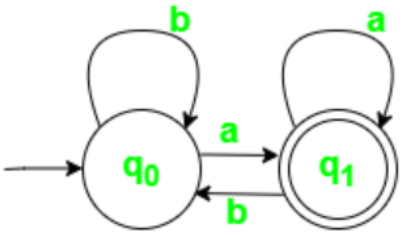
Contoh DFA

Misalkan dibuat DFA yang menerima semua string yang berakhir dengan huruf ‘a’.

Diketahui:

- Σ = {a, b}
- Q = {q_0, q_1}
- F = {q_1}

Diagram Transisi



Gambar 2.2.1 Diagram Transisi DFA
Sumber: https://www.geeksforgeeks.org/theory-of-computation/introduction-of-finite-automata/

Tabel Transisi:

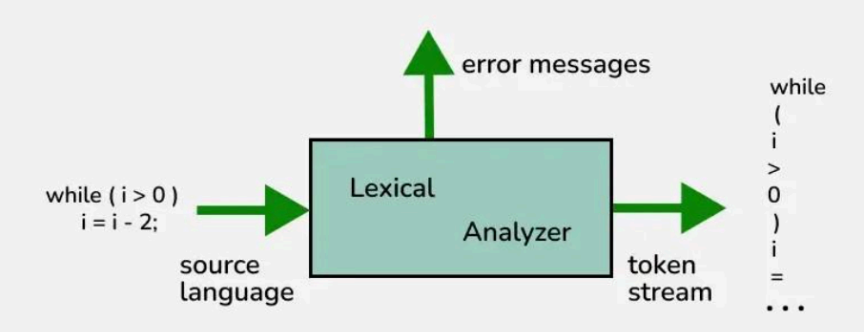
State	Input a	Input b
q_0	q_1	q_0
q_1	q_1	q_0

Penjelasan:

- Mesin memulai dari *state* awal **q**0.
- Jika membaca simbol **'a'**, maka berpindah ke *state* **q**1.
- Jika membaca simbol **'b'**, maka tetap atau kembali ke *state* **q**0.
- Setelah seluruh masukan dibaca, bila mesin berakhir di *state* **q**1, maka string diterima karena berakhir dengan huruf **'a'**.

2.3. Lexical Analysis

Lexical analysis adalah tahap pertama dalam proses compiler. Pada tahap ini, program sumber dibaca karakter demi karakter dari kiri ke kanan dan dikonversi menjadi token-token bermakna.



Gambar 2.3.1 Lexical Analysis
Sumber: https://www.geeksforgeeks.org/compiler-design/introduction-of-lexical-analysis/

Apa Itu Token?

Token adalah rangkaian karakter yang dikenali sebagai satu satuan dalam tata bahasa pemrograman. Misalnya kata while, x, >=, angka 123, dan simbol ; dianggap token. Token

adalah representasi abstrak; sedangkan rangkaian karakter yang membentuk token itu disebut **lexeme**.

Kategori Token

Beberapa kategori token umum dalam bahasa pemrograman antara lain:

- Keyword (kata kunci): kata khusus yang sudah didefinisikan dalam bahasa (misalnya if, else, for)
- Identifier: nama variabel, fungsi, atau elemen buatan pengguna
- Konstanta / Literal: nilai tetap seperti angka, karakter, string
- Operator: simbol operasi seperti +, -, *, /, >=
- Simbol khusus / tanda baca: seperti ;, ,, {, }, (,)

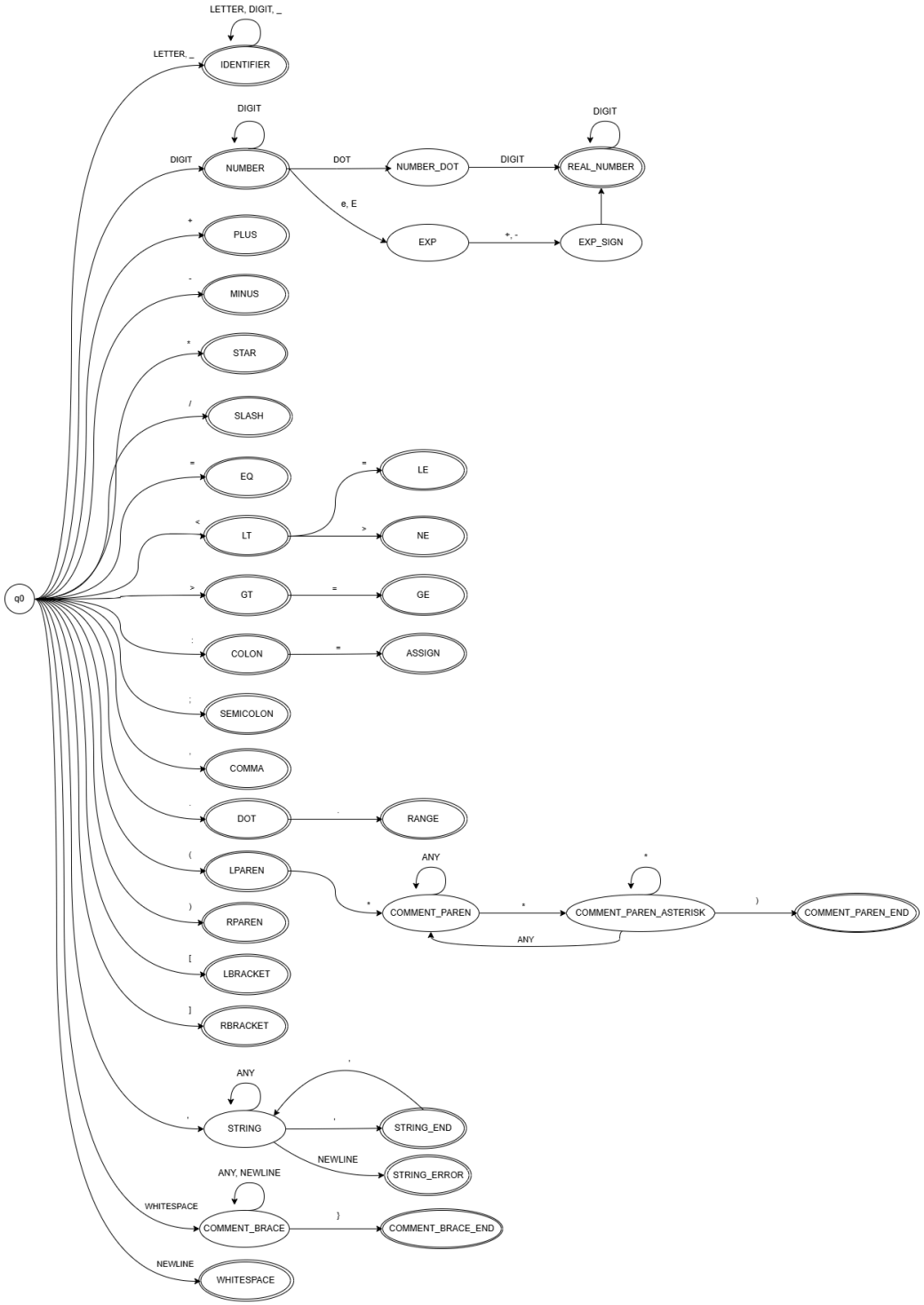
Cara Kerja *Lexical Analysis*

- Token-token dapat dinyatakan dengan ekspresi reguler (regular expressions).
- Scanner (lexer) menggunakan DFA (Deterministic Finite Automaton) untuk mengenali token-token ini, sebab DFA ideal untuk mengenali bahasa regular.
- Setiap *final state* pada DFA akan memetakan ke jenis token tertentu, sehingga ketika DFA berakhir pada status final, *lexeme* yang terbentuk dapat diklasifikasikan sebagai token yang sesuai.

Selain itu, lexer juga bertugas mendeteksi kesalahan leksikal. Ketika ada karakter atau urutan karakter yang tidak cocok dengan pola token yang sah. Pada saat itu, lexer biasanya akan melaporkan posisi baris dan kolom dari kesalahan tersebut.

III. PERANCANGAN & IMPLEMENTASI

3.1. Perancangan



DFA dimulai dari state awal q0 dan dirancang untuk mengenali berbagai jenis token dalam bahasa Pascal seperti identifier, angka, operator, dan lain lain. Token-token tersebut menjadi final state dan karakter yang dibaca menentukan arah transisi dari satu state ke state lainnya. DFA ini tidak dapat mengenali keywords secara langsung. Jika mencapai final state berupa identifier, lexer akan mengecek kembali apakah teks tersebut termasuk keyword dengan membandingkannya ke daftar keywords yang ada.

3.2.Implementasi

3.2.1. Struktur Kode Program

```
root
├── doc
│   └── .gitkeep
├── src
│   ├── app.py
│   ├── dfa_rules.json
│   ├── dfa.py
│   ├── lexer.py
│   ├── main.py
│   ├── pascal_token.py
│   └── utils.py
├── test
│   └── milestone-1
│   ├── input
│   └── output
└── README.md
```

3.2.2. Implementasi Kode Program

- app.py

  Fungsi utama yang mengorkestrasi seluruh alur kerja program. Fungsi ini memvalidasi argumen command-line, membaca file input, memuat aturan DFA dari file JSON, membuat instance dari kelas Lexer, menjalankan proses tokenisasi, dan mencetak hasilnya ke konsol dengan format yang sesuai.

```
def app():
    # Langkah 1: Validasi input dari command line
    if len(sys.argv) != 2:
        print("Usage: python app.py <source_file.pas>")
        sys.exit(1)

    source_path = sys.argv[1]
    dfa_path = "src/dfa_rules.json"

    # Langkah 2: Baca konten file source code
    source = read_source_code(source_path)
```

```
    # Langkah 3: Muat aturan DFA
    dfa_rules = load_dfa_rules(dfa_path)

    # Langkah 4: Buat objek Lexer dan jalankan tokenisasi
    lexer = Lexer(source, dfa_rules)
    tokens = lexer.tokenize()

    # Langkah 5: Cetak setiap token dalam daftar hasil
    for token in tokens:
        print(token)
```

- dfa.py

  Kelas DFA melakukan simulasi langkah demi langkah pada mesin Deterministic Finite Automaton (DFA) berdasarkan aturan-aturan yang didefinisikan dalam dfa_rules.json. Metode get_char_category() digunakan untuk mengelompokkan setiap karakter input ke dalam kategori yang dikenali oleh DFA. Metode simulate_dfa_step() digunakan untuk menjalankan langkah transisi DFA berdasarkan state saat ini dan karakter yang sedang dibaca.

```
class DFA:
    @staticmethod
    def get_char_category(char: str) -> str:
        """
        Mengklasifikasikan sebuah karakter ke dalam kategori yang
dikenali oleh DFA.

        Fungsi ini bertindak sebagai pemetaan dari karakter spesifik
('a', 'b', '7', ';')
        ke kategori umum yang ada di file dfa_rules.json.

        Args:
            char (str): Karakter tunggal yang akan diklasifikasikan.

        Returns:
            str: Nama kategori yang sesuai (misal: 'LETTER', 'DIGIT',
'WHITESPACE').
        """
        if char.isalpha():
            return "LETTER"
        elif char.isdigit():
            return "DIGIT"
        elif char == '\n' or char == '\r':
            return "NEWLINE"
        elif char.isspace():
            return "WHITESPACE"
        elif char == '_':
            return "UNDERSCORE"
        else:
            return "UNKNOWN"

    @staticmethod
    def simulate_dfa_step(current_state: str, char: str, dfa_rules: dict)
-> str:
        """
```

```
        Melakukan satu langkah transisi DFA berdasarkan karakter input.

        Args:
            current_state (str): State saat ini.
            char (str): Karakter input tunggal yang dibaca.
            dfa_rules (dict): Aturan DFA yang berisi 'transitions',
'initial_state', dan 'final_states'.

        Returns:
            str: State berikutnya setelah membaca karakter tersebut.
                Jika tidak ada transisi yang valid, mengembalikan None.
        """
        transitions = dfa_rules.get("transitions", {})
        current_transitions = transitions.get(current_state, {})

        if char in current_transitions:
            return current_transitions[char]

        category = DFA.get_char_category(char)

        if category in current_transitions:
            return current_transitions[category]
        elif "ANY" in current_transitions:
            return current_transitions["ANY"]
        else:
            return None
```

- dfa_rules.json

  Aturan-aturan DFA yang digunakan dalam program ini.

```
{
  "initial_state": "q0",
  "final_states": {
    "IDENTIFIER": { "token": "IDENTIFIER" },
    "NUMBER": { "token": "NUMBER" },
    "REAL_NUMBER": { "token": "NUMBER" },
    "PLUS": { "token": "ARITHMETIC_OPERATOR" },
    "MINUS": { "token": "ARITHMETIC_OPERATOR" },
    "STAR": { "token": "ARITHMETIC_OPERATOR" },
    "SLASH": { "token": "ARITHMETIC_OPERATOR" },
    "EQ": { "token": "RELATIONAL_OPERATOR" },
    "LT": { "token": "RELATIONAL_OPERATOR" },
    "GT": { "token": "RELATIONAL_OPERATOR" },
    "NE": { "token": "RELATIONAL_OPERATOR" },
    "LE": { "token": "RELATIONAL_OPERATOR" },
    "GE": { "token": "RELATIONAL_OPERATOR" },
    "ASSIGN": { "token": "ASSIGN_OPERATOR" },
    "COLON": { "token": "COLON" },
    "SEMICOLON": { "token": "SEMICOLON" },
    "COMMA": { "token": "COMMA" },
    "DOT": { "token": "DOT" },
    "RANGE": { "token": "RANGE_OPERATOR" },
    "LPAREN": { "token": "LPARENTHESIS" },
    "RPAREN": { "token": "RPARENTHESIS" },
    "LBRACKET": { "token": "LBRACKET" },
    "RBRACKET": { "token": "RBRACKET" },
    "STRING_END": { "token": "STRING_LITERAL" },
```

```
    "STRING_ERROR": { "token": "ERROR", "is_error": true, "error_type":
"UNTERMINATED_STRING" },
    "COMMENT_BRACE_END": { "token": "COMMENT", "ignore": true },
    "COMMENT_PAREN_END": { "token": "COMMENT", "ignore": true },
    "WHITESPACE": { "token": "WHITESPACE", "ignore": true }
  },
  "transitions": {
    "q0": {
      "LETTER": "IDENTIFIER",
      "_": "IDENTIFIER",
      "DIGIT": "NUMBER",
      "+": "PLUS",
      "-": "MINUS",
      "*": "STAR",
      "/": "SLASH",
      "=": "EQ",
      "<": "LT",
      ">": "GT",
      ":": "COLON",
      ";": "SEMICOLON",
      ",": "COMMA",
      ".": "DOT",
      "(": "LPAREN",
      ")": "RPAREN",
      "[": "LBRACKET",
      "]": "RBRACKET",
      "'": "STRING",
      "{": "COMMENT_BRACE",
      "WHITESPACE": "WHITESPACE",
      "NEWLINE": "WHITESPACE"
    },
    "IDENTIFIER": {
      "LETTER": "IDENTIFIER",
      "DIGIT": "IDENTIFIER",
      "UNDERSCORE": "IDENTIFIER"
    },
    "NUMBER": {
      "DIGIT": "NUMBER",
      ".": "NUMBER_DOT",
      "e": "EXP",
      "E": "EXP"
    },
    "REAL_NUMBER": {
      "DIGIT": "REAL_NUMBER",
      "e": "EXP",
      "E": "EXP"
    },
    "LT": {
      "=": "LE",
      ">": "NE"
    },
    "GT": {
      "=": "GE"
    },
    "COLON": {
      "=": "ASSIGN"
    },
    "DOT": {
      ".": "RANGE"
    },
    "NUMBER_DOT": {
      "DIGIT": "REAL_NUMBER"
```

```
    },
    "EXP": {
      "+": "EXP_SIGN",
      "-": "EXP_SIGN",
      "DIGIT": "REAL_NUMBER"
    },
    "EXP_SIGN": {
      "DIGIT": "REAL_NUMBER"
    },
    "LPAREN": {
      "*": "COMMENT_PAREN"
    },
    "COMMENT_BRACE": {
      "ANY": "COMMENT_BRACE",
      "NEWLINE": "COMMENT_BRACE",
      "}": "COMMENT_BRACE_END"
    },
    "COMMENT_PAREN": {
      "*": "COMMENT_PAREN_ASTERISK",
      "ANY": "COMMENT_PAREN"
    },
    "COMMENT_PAREN_ASTERISK": {
      ")": "COMMENT_PAREN_END",
      "*": "COMMENT_PAREN_ASTERISK",
      "ANY": "COMMENT_PAREN"
    },
    "STRING_END": {
      "'": "STRING"
    },
    "STRING": {
      "ANY": "STRING",
      "'": "STRING_END",
      "NEWLINE": "STRING_ERROR"
    }
  },
  "KEYWORDS": [
    "program",
    "var",
    "begin",
    "end",
    "if",
    "then",
    "else",
    "while",
    "do",
    "for",
    "to",
    "downto",
    "integer",
    "real",
    "boolean",
    "char",
    "array",
    "of",
    "procedure",
    "function",
    "const",
    "type",
    "true",
    "false"
  ],
  "WORD_ARITHMETIC": ["div", "mod"],
```

```
  "WORD_LOGICAL": ["and", "or", "not"]
}
```

- lexer.py

  Kelas lexer mengubah string kode sumber mentah menjadi daftar objek Token berdasarkan aturan yang didefinisikan dalam file DFA. Lexer membaca input karakter demi karakter lalu menentukan jenis token berdasarkan hasil simulasi DFA. Metode _get_next_token Menganalisis kode sumber dari posisi saat ini untuk menemukan satu token berikutnya menggunakan aturan 'longest match'. Metode _finalilze_token() kemudian membentuk objek token sesuai jenisnya. Token yang ditandai dengan flag ignore seperti komentar dan whitespace akan diabaikan. Metode _handle_error(), _advance_pos() dan _set_post_to() digunakan untuk menangani karakter ilegal dan memperbarui posisi dari kolom pembacaan. Metode tokenize() adalah fungsi utama yang menjalankan keseluruhan proses tokenisasi.

```
class Lexer:
    """
    Lexer: mengubah string kode sumber mentah menjadi daftar objek Token
    berdasarkan aturan yang didefinisikan dalam file DFA.
    """
    def __init__(self, source_code: str, dfa_rules: dict):
        """
        Inisialisasi lexer.

        Args:
            source_code (str): Seluruh kode sumber PASCAL-S sebagai satu
string.
            dfa_rules (dict): Aturan DFA yang sudah di-load dari file
JSON.
        """
        self.source_code = source_code
        self.dfa_rules = dfa_rules
        self.current_pos = 0
        self.current_line = 1
        self.current_col = 1
        self.fatal_error = False

        self.keywords = set(dfa_rules.get("KEYWORDS", []))
        self.word_arithmetic = set(dfa_rules.get("WORD_ARITHMETIC", []))
        self.word_logical = set(dfa_rules.get("WORD_LOGICAL", []))

    def _get_next_token(self) -> Token | None:
        """
        Menganalisis kode sumber dari posisi saat ini untuk menemukan
satu token berikutnya
        menggunakan aturan 'longest match'.
        """
        if self.current_pos >= len(self.source_code):
            return None
```

```
        start_pos = self.current_pos
        start_line = self.current_line
        start_col = self.current_col

        last_final_state = None
        last_final_pos = -1

        current_state = self.dfa_rules["initial_state"]
        pos_tracker = self.current_pos

        while pos_tracker < len(self.source_code):
            char = self.source_code[pos_tracker]

            next_state = DFA.simulate_dfa_step(current_state, char,
self.dfa_rules)

            if next_state is None:
                break

            current_state = next_state
            pos_tracker += 1

            if current_state in self.dfa_rules["final_states"]:
                last_final_state = current_state
                last_final_pos = pos_tracker

        if last_final_state and
self.dfa_rules["final_states"][last_final_state].get("is_error", False):
            error_type =
self.dfa_rules["final_states"][last_final_state].get("error_type",
"UNKNOWN")
            if error_type == "UNTERMINATED_STRING":
                print(f"Error: Unterminated string literal at Line
{start_line}:{start_col}")
            self.fatal_error = True
            self._advance_pos()
            return None

        if pos_tracker >= len(self.source_code) and current_state ==
"STRING":
            print(f"Error: Unterminated string literal at Line
{start_line}:{start_col}")
            self.fatal_error = True
            return None

        if last_final_state is None:
            if not self.source_code[start_pos].isspace():
                self._handle_error(self.source_code[start_pos],
start_line, start_col)

            self._advance_pos()
            return None

        lexeme = self.source_code[start_pos:last_final_pos]

        self._set_pos_to(last_final_pos)

        return self._finalize_token(lexeme, last_final_state, start_line,
start_col)

    def _finalize_token(self, lexeme: str, final_state: str, line: int,
```

```
col: int) -> Token | None:
        """
        Membuat objek Token, melakukan lookup keyword, dan mengecek flag
'ignore'.
        """
        token_info = self.dfa_rules["final_states"][final_state]
        token_type = token_info["token"]

        if token_type == "IDENTIFIER":
            lexeme_lower = lexeme.lower()
            if lexeme_lower in self.keywords:
                token_type = "KEYWORD"
            elif lexeme_lower in self.word_arithmetic:
                token_type = "ARITHMETIC_OPERATOR"
            elif lexeme_lower in self.word_logical:
                token_type = "LOGICAL_OPERATOR"

        if token_type == "STRING_LITERAL":
            string_content = lexeme[1:-1]
            string_content = string_content.replace("\'\'", "\'")
            if len(string_content) <= 1:
                token_type = "CHAR_LITERAL"
            lexeme = f"\'{string_content}\'"

        if token_info.get("ignore", False):
            return None

        return Token(token_type=token_type, value=lexeme, line=line,
column=col)

    def _handle_error(self, char: str, line: int, col: int):
        """
        Menangani error karakter tidak dikenal.
        """
        print(f"Error: Invalid character '{char}' at Line {line}:{col}")

    def _advance_pos(self):
        """Helper untuk memajukan lexer 1 karakter dan update
line/col."""
        if self.current_pos >= len(self.source_code):
            return

        char = self.source_code[self.current_pos]

        if char == '\r':
            self.current_line += 1
            self.current_col = 1
            self.current_pos += 1
            if self.current_pos < len(self.source_code) and
self.source_code[self.current_pos] == '\n':
                self.current_pos += 1
        elif char == '\n':
            self.current_line += 1
            self.current_col = 1
            self.current_pos += 1
        else:
            self.current_col += 1
            self.current_pos += 1

    def _set_pos_to(self, new_pos: int):
        """Helper untuk menggerakkan lexer ke posisi baru (setelah token)
dan update line/col."""
```

```
        while self.current_pos < new_pos:
            self._advance_pos()

    def tokenize(self) -> list[Token]:
        """
        Fungsi publik utama untuk menjalankan keseluruhan proses
tokenisasi.
        """
        tokens = []
        while True:
            token = self._get_next_token()

            if self.fatal_error:
                break

            if token is None:
                if self.current_pos >= len(self.source_code):
                    break
                continue

            tokens.append(token)

        return tokens
```

- main.py

Kode ini memanggil fungsi app() untuk memastikan bahwa fungsi app() hanya akan dieksekusi ketika file ini dijalankan sebagai script utama.

```
if __name__ == "__main__":
    """
    Konstruksi ini memastikan bahwa fungsi app() hanya akan dieksekusi
    ketika file ini dijalankan sebagai script utama. Ini adalah praktik
    standar dalam Python. Program dijalankan dengan perintah:
    `python -m src.main path/to/your/program.pas`
    """
    app()
```

- token.py

Kelas token adalah kelas yang merepresentasikan token leksikal.

```
@dataclass
class Token:
    """
    Sebuah kelas untuk merepresentasikan token leksikal.

    Setiap token memiliki jenis (misalnya, 'IDENTIFIER'), nilai
(misalnya, 'x'),
```

```
    dan nomor baris tempat token itu ditemukan dalam kode sumber.
    Ini membantu untuk pelacakan dan pelaporan kesalahan (error
reporting) nanti.
    """
    token_type: str
    value: str
    line: int
    column: int

    def __repr__(self) -> str:
        """
        Menyediakan representasi string yang jelas untuk token.

        Ini sangat berguna untuk proses debugging dan untuk mencetak
daftar token
        yang berhasil di-scan sesuai format output yang diminta.
        Contoh: "KEYWORD(program)" atau "NUMBER(123)".
        """
        return f"{self.token_type}({self.value})"
```

- utils.py

  File ini berisi fungsi-fungsi bantu untuk membaca file konfigurasi dan kode sumber. Fungsi load_dfa_rules(filepath) digunakan untuk membuka dan memuat isi file dfa_rules.json ke dalam bentuk dictionary Python yang berisi definisi transisi, state awal, dan final state DFA. Fungsi read_source_code(path) digunakan untuk membaca isi file kode sumber Pascal dan mengembalikannya sebagai string agar dapat diproses lebih lanjut oleh lexer.

```
import json
import sys

def load_dfa_rules(filepath: str) -> dict:
    """
    Membaca file aturan DFA dalam format JSON dan mengubahnya menjadi
dictionary.

    Fungsi ini menangani pembukaan file dan parsing JSON. Jika terjadi
error,
    seperti file tidak ditemukan atau format JSON tidak valid, fungsi
akan
    mencetak pesan error dan menghentikan program secara aman.

    Args:
        filepath (str): Path menuju file dfa_rules.json.

    Returns:
        dict: Dictionary yang berisi aturan transisi, state awal, dan
final states.
              Mengembalikan None jika terjadi error.
    """
    try:
```

```
        with open(filepath, 'r', encoding='utf-8') as file:
            dfa_rules = json.load(file)
            return dfa_rules
    except FileNotFoundError:
        print(f"Error: File '{filepath}' tidak ditemukan.")
        sys.exit(1)
    except json.JSONDecodeError as e:
        print(f"Error: File '{filepath}' bukan file JSON valid: {e}")
        sys.exit(1)
    except Exception as e:
        print(f"Terjadi kesalahan saat membaca file DFA '{filepath}':
{e}")
        sys.exit(1)

def read_source_code(path):
    """
    Membaca file Pascal-S input dan mengembalikan string kontennya.
    """
    with open(path, 'r', encoding='utf-8') as f:
        return f.read()
```

IV. Pengujian

4.1. Pengujian 1

Tujuan pengujian: Untuk memvalidasi bahwa lexer dapat mengenali struktur program PASCAL-S yang paling minimal dan valid sehingga pengujian berfokus pada token-token fundamental, termasuk keyword (program, var, begin, end, integer), IDENTIFIER, NUMBER, ASSIGN_OPERATOR (:=), dan delimiter (:, ;, .).

File: 1-basic.pas

Input	Output
program Basic; var x: integer; begin x := 10; end.	KEYWORD(program) IDENTIFIER(Basic) SEMICOLON(;) KEYWORD(var) IDENTIFIER(x) COLON(:) KEYWORD(integer) SEMICOLON(;) KEYWORD(begin) IDENTIFIER(x) ASSIGN_OPERATOR(:=) NUMBER(10) SEMICOLON(;) KEYWORD(end) DOT(.)

4.2. Pengujian 2

Tujuan pengujian: Untuk memvalidasi pengenalan semua kategori operator yang ada di spesifikasi, mencakup operator aritmatika simbolik (+, -, *, /), aritmatika kata (div, mod), operator relasional (>, <=, <>, =), dan operator logika (and, or, not).

File: 2-all-operators.pas

Input	Output
program AllOperators; var a, b, c: integer; d, e: boolean; begin	KEYWORD(program) IDENTIFIER(AllOperators) SEMICOLON(;) KEYWORD(var) IDENTIFIER(a)

{ Arithmetic } a := 10 + 5 - 2 * 3; b := 100 div 10 mod 3; c := a / b; { Relational } d := (a > b) and (b <= c); e := (a <> c) or (c = 10); { Logical } d := not e; end.	COMMA(,) IDENTIFIER(b) COMMA(,) IDENTIFIER(c) COLON(:) KEYWORD(integer) SEMICOLON(;) IDENTIFIER(d) COMMA(,) IDENTIFIER(e) COLON(:) KEYWORD(boolean) SEMICOLON(;) KEYWORD(begin) IDENTIFIER(a) ASSIGN_OPERATOR(:=) NUMBER(10) ARITHMETIC_OPERATOR(+) NUMBER(5) ARITHMETIC_OPERATOR(-) NUMBER(2) ARITHMETIC_OPERATOR(*) NUMBER(3) SEMICOLON(;) IDENTIFIER(b) ASSIGN_OPERATOR(:=) NUMBER(100) ARITHMETIC_OPERATOR(div) NUMBER(10) ARITHMETIC_OPERATOR(mod) NUMBER(3) SEMICOLON(;) IDENTIFIER(c) ASSIGN_OPERATOR(:=) IDENTIFIER(a) ARITHMETIC_OPERATOR(/) IDENTIFIER(b) SEMICOLON(;) IDENTIFIER(d) ASSIGN_OPERATOR(:=) LPARENTHESIS(() IDENTIFIER(a) RELATIONAL_OPERATOR(>) IDENTIFIER(b) RPARENTHESIS()) LOGICAL_OPERATOR(and)

	LPARENTHESIS(() IDENTIFIER(b) RELATIONAL_OPERATOR(<=) IDENTIFIER(c) RPARENTHESIS()) SEMICOLON(;) IDENTIFIER(e) ASSIGN_OPERATOR(:=) LPARENTHESIS(() IDENTIFIER(a) RELATIONAL_OPERATOR(<>) IDENTIFIER(c) RPARENTHESIS()) LOGICAL_OPERATOR(or) LPARENTHESIS(() IDENTIFIER(c) RELATIONAL_OPERATOR(=) NUMBER(10) RPARENTHESIS()) SEMICOLON(;) IDENTIFIER(d) ASSIGN_OPERATOR(:=) LOGICAL_OPERATOR(not) IDENTIFIER(e) SEMICOLON(;) KEYWORD(end) DOT(.)

4.3. Pengujian 3

Tujuan pengujian: Untuk menguji pengenalan keyword yang terkait dengan definisi tipe data, konstanta, dan struktur program, seperti const, type, array, real, boolean, char, procedure, dan function.

File: 3-all-types.pas

Input	Output
program AllTypes; const PI = 3.14; type MyArray = array [1..10] of real; var	KEYWORD(program) IDENTIFIER(AllTypes) SEMICOLON(;) KEYWORD(const) IDENTIFIER(PI) RELATIONAL_OPERATOR(=)

x: integer; y: real; z: boolean; w: char; arr: MyArray; procedure TestProc; begin { do nothing } end; function TestFunc: integer; begin TestFunc := 1; end; begin y := PI; z := true; w := 'a'; end.	NUMBER(3.14) SEMICOLON(;) KEYWORD(type) IDENTIFIER(MyArray) RELATIONAL_OPERATOR(=) KEYWORD(array) LBRACKET([) NUMBER(1) RANGE_OPERATOR(..) NUMBER(10) RBRACKET(]) KEYWORD(of) KEYWORD(real) SEMICOLON(;) KEYWORD(var) IDENTIFIER(x) COLON(:) KEYWORD(integer) SEMICOLON(;) IDENTIFIER(y) COLON(:) KEYWORD(real) SEMICOLON(;) IDENTIFIER(z) COLON(:) KEYWORD(boolean) SEMICOLON(;) IDENTIFIER(w) COLON(:) KEYWORD(char) SEMICOLON(;) IDENTIFIER(arr) COLON(:) IDENTIFIER(MyArray) SEMICOLON(;) KEYWORD(procedure) IDENTIFIER(TestProc) SEMICOLON(;) KEYWORD(begin) KEYWORD(end) SEMICOLON(;) KEYWORD(function) IDENTIFIER(TestFunc) COLON(:) KEYWORD(integer) SEMICOLON(;)

	KEYWORD(begin) IDENTIFIER(TestFunc) ASSIGN_OPERATOR(:=) NUMBER(1) SEMICOLON(;) KEYWORD(end) SEMICOLON(;) KEYWORD(begin) IDENTIFIER(y) ASSIGN_OPERATOR(:=) IDENTIFIER(PI) SEMICOLON(;) IDENTIFIER(z) ASSIGN_OPERATOR(:=) KEYWORD(true) SEMICOLON(;) IDENTIFIER(w) ASSIGN_OPERATOR(:=) CHAR_LITERAL('a') SEMICOLON(;) KEYWORD(end) DOT(.)

4.4. Pengujian 4

Tujuan pengujian: Untuk memvalidasi kemampuan DFA dalam menangani berbagai format NUMBER, mencakup bilangan real, notasi ilmiah (menggunakan E+ atau E-), serta menangani kasus bilangan negatif.

File: 4-numbers.pas

Input	Output
program TestNumbers; var r_num: real; i_num: integer; begin r_num := 123.456; r_num := 0.5; i_num := 5; { Scientific Notation } r_num := 1.23E+10;	KEYWORD(program) IDENTIFIER(TestNumbers) SEMICOLON(;) KEYWORD(var) IDENTIFIER(r_num) COLON(:) KEYWORD(real) SEMICOLON(;) IDENTIFIER(i_num) COLON(:) KEYWORD(integer)

r_num := 5E-2; { Negative numbers (as Op + Num) } i_num := -10; { Longest match test: Number vs Dot } i_num := 10.; { -> NUMBER(10) DOT(.) } end.	SEMICOLON(;) KEYWORD(begin) IDENTIFIER(r_num) ASSIGN_OPERATOR(:=) NUMBER(123.456) SEMICOLON(;) IDENTIFIER(r_num) ASSIGN_OPERATOR(:=) NUMBER(0.5) SEMICOLON(;) IDENTIFIER(i_num) ASSIGN_OPERATOR(:=) NUMBER(5) SEMICOLON(;) IDENTIFIER(r_num) ASSIGN_OPERATOR(:=) NUMBER(1.23E+10) SEMICOLON(;) IDENTIFIER(r_num) ASSIGN_OPERATOR(:=) NUMBER(5E-2) SEMICOLON(;) IDENTIFIER(i_num) ASSIGN_OPERATOR(:=) ARITHMETIC_OPERATOR(-) NUMBER(10) SEMICOLON(;) IDENTIFIER(i_num) ASSIGN_OPERATOR(:=) NUMBER(10) DOT(.) SEMICOLON(;) KEYWORD(end) DOT(.)

4.5. Pengujian 5

Tujuan pengujian: Untuk memvalidasi pengenalan STRING_LITERAL dan CHAR_LITERAL. Pengujian ini mencakup kasus khusus seperti string kosong dan penggunaan *escaped quote* dalam string.

File: 5-string-char.pas

Input	Output

program TestStrings; var s: array [1..20] of char; c: char; begin s := 'Hello TBFO!'; c := 'X'; { Empty string should be CHAR_LITERAL } c := ''; { Double quote inside string } s := 'Ini adalah "quote" di dalam.'; writeln(s, c); end.	KEYWORD(program) IDENTIFIER(TestStrings) SEMICOLON(;) KEYWORD(var) IDENTIFIER(s) COLON(:) KEYWORD(array) LBRACKET([) NUMBER(1) RANGE_OPERATOR(..) NUMBER(20) RBRACKET(]) KEYWORD(of) KEYWORD(char) SEMICOLON(;) IDENTIFIER(c) COLON(:) KEYWORD(char) SEMICOLON(;) KEYWORD(begin) IDENTIFIER(s) ASSIGN_OPERATOR(:=) STRING_LITERAL('Hello TBFO!') SEMICOLON(;) IDENTIFIER(c) ASSIGN_OPERATOR(:=) CHAR_LITERAL('X') SEMICOLON(;) IDENTIFIER(c) ASSIGN_OPERATOR(:=) CHAR_LITERAL('') SEMICOLON(;) IDENTIFIER(s) ASSIGN_OPERATOR(:=) STRING_LITERAL('Ini adalah 'quote' di dalam.') SEMICOLON(;) IDENTIFIER(writeln) LPARENTHESIS(() IDENTIFIER(s) COMMA(,) IDENTIFIER(c) RPARENTHESIS()) SEMICOLON(;) KEYWORD(end) DOT(.)

4.6. Pengujian 6

Tujuan pengujian: Untuk memverifikasi bahwa lexer mengabaikan semua blok komentar yang menggunakan format kurung kurawal {...} maupun format kurung asterisk (*...*).

File: 6-comments.pas

Input	Output
program Basic; { Ini komentar brace } var x: integer; (* Ini komentar asterisk *) { Komentar multi-baris } begin x := 10; (* x di-assign 10 *) end.	KEYWORD(program) IDENTIFIER(Basic) SEMICOLON(;) KEYWORD(var) IDENTIFIER(x) COLON(:) KEYWORD(integer) SEMICOLON(;) KEYWORD(begin) IDENTIFIER(x) ASSIGN_OPERATOR(:=) NUMBER(10) SEMICOLON(;) KEYWORD(end) DOT(.)

4.7. Pengujian 7

Tujuan pengujian: Untuk menguji secara spesifik implementasi prinsip *longest match* pada lexer, di mana lexer harus memilih token terpanjang yang mungkin dari serangkaian karakter yang ambigu.

File: 7-longest-match.pas

Input	Output
program LongMatch; var i: integer; begin i := 10; { Test 1: Keyword vs Identifier } programTest := 1; { ->	KEYWORD(program) IDENTIFIER(LongMatch) SEMICOLON(;) KEYWORD(var) IDENTIFIER(i) COLON(:) KEYWORD(integer) SEMICOLON(;)

IDENTIFIER(programTest) } { Test 2: Operators } if i <> 0 then { -> RELATIONAL_OPERATOR(<>) } if i >= 1 then { -> RELATIONAL_OPERATOR(>=) } i := 5; { -> ASSIGN_OPERATOR(:=) } { Test 3: Range vs Dot } i := 1..10; { -> NUMBER(1) RANGE_OPERATOR(..) NUMBER(10) } end.	KEYWORD(begin) IDENTIFIER(i) ASSIGN_OPERATOR(:=) NUMBER(10) SEMICOLON(;) IDENTIFIER(programTest) ASSIGN_OPERATOR(:=) NUMBER(1) SEMICOLON(;) KEYWORD(if) IDENTIFIER(i) RELATIONAL_OPERATOR(<>) NUMBER(0) KEYWORD(then) KEYWORD(if) IDENTIFIER(i) RELATIONAL_OPERATOR(>=) NUMBER(1) KEYWORD(then) IDENTIFIER(i) ASSIGN_OPERATOR(:=) NUMBER(5) SEMICOLON(;) IDENTIFIER(i) ASSIGN_OPERATOR(:=) NUMBER(1) RANGE_OPERATOR(..) NUMBER(10) SEMICOLON(;) KEYWORD(end) DOT(.)

4.8. Pengujian 8

Tujuan pengujian: Untuk mendemonstrasikan kemampuan lexer dalam menangani *error handling* ketika menemukan karakter yang tidak dikenali atau tidak valid dalam *source code* PASCAL-S.

File: 8-errors.pas

Input	Output
program TestErrors; var	Error: Invalid character '$' at Line 6:12 Error: Invalid character '@' at Line 7:9

x: integer; begin { Karakter tidak valid } x := 10 $ 5; { $ bukan token valid } x := @x; { @ bukan token valid } x := ~0; { ~ bukan token valid } { String tidak ditutup } writeln('Hello... end.	Error: Invalid character '~' at Line 8:9 Error: Unterminated string literal at Line 11:12 KEYWORD(program) IDENTIFIER(TestErrors) SEMICOLON(;) KEYWORD(var) IDENTIFIER(x) COLON(:) KEYWORD(integer) SEMICOLON(;) KEYWORD(begin) IDENTIFIER(x) ASSIGN_OPERATOR(:=) NUMBER(10) NUMBER(5) SEMICOLON(;) IDENTIFIER(x) ASSIGN_OPERATOR(:=) IDENTIFIER(x) SEMICOLON(;) IDENTIFIER(x) ASSIGN_OPERATOR(:=) NUMBER(0) SEMICOLON(;) IDENTIFIER(writeln) LPARENTHESIS(()

4.9. Pengujian 9

Tujuan pengujian: Untuk memastikan lexer dapat menangani berbagai jenis *whitespace* (spasi, tab, dan *newline*) dengan benar, yaitu dengan mengabaikannya dan tidak menghasilkan token.

File: 9-whitespace.pas

Input	Output
program Whitespace; var x: integer; y: integer;	KEYWORD(program) IDENTIFIER(Whitespace) SEMICOLON(;) KEYWORD(var) IDENTIFIER(x) COLON(:)

begin x := 10; y := 20; writeln(x, y); end.	KEYWORD(integer) SEMICOLON(;) IDENTIFIER(y) COLON(:) KEYWORD(integer) SEMICOLON(;) KEYWORD(begin) IDENTIFIER(x) ASSIGN_OPERATOR(:=) NUMBER(10) SEMICOLON(;) IDENTIFIER(y) ASSIGN_OPERATOR(:=) NUMBER(20) SEMICOLON(;) IDENTIFIER(writeln) LPARENTHESIS(() IDENTIFIER(x) COMMA(,) IDENTIFIER(y) RPARENTHESIS()) SEMICOLON(;) KEYWORD(end) DOT(.)

V. KESIMPULAN DAN SARAN

5.1. Kesimpulan

Implementasi lexer berbasis DFA yang dibuat pada milestone ini berhasil mengubah kode sumber Pascal-S menjadi daftar token sesuai spesifikasi. Pembacaan karakter dan pengidentifikasian token dilakukan oleh program lexer menggunakan prinsip *longest match.* Hasil pengujian menunjukkan bahwa lexer mampu mengenali identifier, keyword, number, dan token lainnya, menangani *error* atau kesalahan leksikal dalam kode sumber, dan mengabaikan whitespace dan komentar.

5.2.Saran

Hasil tokenisasi yang dihasilkan lexer pada tahap ini dapat diintegrasikan langsung dengan parser pada tahap analisis sintaksis berikutnya. Integrasi ini memungkinkan komunikasi yang lebih efisien antara komponen lexer dan parser, sehingga keseluruhan proses kompilasi menjadi lebih terorganisir dan mudah untuk dikembangkan lebih lanjut.

LAMPIRAN

- Tautan *repository* GitHub : https://github.com/anellautari/DFC-Tubes-IF2224
- Tautan *workspace* diagram : Workspace Diagram DFA

PEMBAGIAN TUGAS

NAMA	NIM	TUGAS
Mayla Yaffa Ludmilla	13523050	Diagram DFA, laporan implementasi dan kesimpulan.
Anella Utari Gunadi	13523078	Laporan deskripsi tugas dan landasan teori, implementasi input dan DFA.
Muhammad Edo Raduputu Aprima	13523096	Implementasi utils, test case dan pengujian.
Athian Nugraha Muarajuang	13523106	Aturan DFA, implementasi lexer, dan output.

REFERENSI

Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation. 3rd ed*, 2007. Tersedia: https://cdn-edunex.itb.ac.id/29161-Formal-Language-Theory-and-Automata/1629640939613_Introduction-to-Automata-Theory,-Languages,-and-Computation-Edition-3.pdf [Diakses: 11-Oktober-2025].

Wirth, Niklaus. "PASCAL-S: A Subset and its implementation", Tersedia: http://pascal.hansotten.com/uploads/pascals/PASCAL-S%20A%20subset%20and%20its%20Implementation%20012.pdf

tutorialspoint.com. *Compiler Design*. Tersedia: https://www.tutorialspoint.com/compiler_design/index.htm [Diakses 11-Oktober-2025]

TutorialsPoint. *DFA for Tokens,* 2018. Tersedia: https://youtu.be/2ZYsKiEb5w4?si=6eablbvt4hg-8EE8 [Diakses 12-Oktober-2025]

GeeksForGeeks. *Introduction of Finite Automata,* 2025. Tersedia: https://www.geeksforgeeks.org/theory-of-computation/introduction-of-finite-automata/ [Diakses 13-Oktober-2025]

GeeksForGeeks. *Introduction of Lexical Analysis,* 2025. Tersedia: https://www.geeksforgeeks.org/compiler-design/introduction-of-lexical-analysis/ [Diakses 13-Oktober-2025]