

LAPORAN TUGAS BESAR TEORI BAHASA FORMAL DAN AUTOMATA

IF3170 Teori Bahasa Formal dan Automata

Milestone 1 : LEXICAL ANALYSIS



Disusun Oleh:

Indah Novita Tangdililing (13523047)

Muhammad FIthra Rizki (13523049)

Sakti Bimasena (13523053)

Muhammad Timur Kanigara (13523055)

Kefas Kurnia Jonathan (13523113)

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG**

2025

DAFTAR ISI

DAFTAR ISI.....	3
------------------------	----------

1. Landasan Teori

1.1 Konsep Bahasa dan Automata

Analisis leksikal merupakan penerapan dari teori bahasa dan automata, yang menjelaskan bagaimana komputer dapat mengenali pola bahasa melalui seperangkat aturan yang terdefinisi.

Menurut Noam Chomsky, bahasa dibedakan menjadi empat jenis yang disebut Hirarki Chomsky, yaitu:

- Tipe 0 (Unrestricted Grammar) – dikenali oleh Turing Machine dan tidak memiliki batasan aturan produksi.
- Tipe 1 (Context-Sensitive Grammar) – dikenali oleh Linear Bounded Automaton, dengan panjang ruas kiri aturan produksi lebih kecil atau sama dengan ruas kanan.
- Tipe 2 (Context-Free Grammar) – dikenali oleh Push Down Automaton, dengan ruas kiri aturan produksi berupa satu simbol variabel.
- Tipe 3 (Regular Grammar) – dikenali oleh Finite State Automaton (FSA) dan menjadi dasar dari proses analisis leksikal.

Analisis leksikal bekerja dengan bahasa regular (tipe 3) yang dapat dikenali menggunakan Deterministic Finite Automata (DFA).

1.2 Tata Bahasa dan Notasi BNF

Tata bahasa (grammar) adalah sekumpulan aturan produksi yang mendefinisikan bagaimana simbol disusun untuk membentuk kalimat dalam suatu bahasa. Grammar terdiri dari simbol terminal, non-terminal, simbol awal, serta aturan produksi.

Backus–Naur Form (BNF) digunakan untuk menuliskan grammar secara formal. Notasi ini memudahkan pendefinisian struktur sintaks bahasa pemrograman dengan cara yang terstandar dan mudah dipahami. Dengan BNF, struktur bahasa dapat dijelaskan secara sistematis sehingga membantu dalam proses pembuatan parser pada compiler.

1.3 Analisis Leksikal

Analisis leksikal adalah tahap pertama dalam proses kompilasi, dimana program sumber yang berupa kumpulan karakter akan dibaca dan diubah menjadi unit-unit yang disebut token. Token merupakan elemen dasar yang memiliki arti tertentu dalam bahasa pemrograman, seperti kata kunci, *identifier*, *operator*, angka, atau tanda baca.

Analisis leksikal bertugas untuk mengenali pola karakter dan mengubah menjadi token, mengabaikan spasi, *tab*, dan juga komentar. Selain itu, juga menangani kesalahan seperti karakter yang tidak dikenal, dan menyimpan informasi token ke dalam tabel simbol yang ada. Komponen yang melakukan proses analisis ini disebut *lexer* atau *scanner*. *Lexer* membaca input dari kiri ke kanan dan menghasilkan daftar token yang akan digunakan di tahap selanjutnya.

1.4 Deterministic Finite Automata (DFA)

Deterministic Finite Automata (DFA) merupakan model matematis untuk mengenali bahasa regular. DFA memiliki beberapa keadaan (*state*) dan aturan transisi yang menentukan perpindahan antar *state* berdasarkan masukan simbol tertentu. Dalam konteks analisis leksikal di bab sebelumnya, DFA berfungsi mengenali pola karakter yang sesuai dengan jenis token tertentu. Setiap jalur transisi pada DFA menggambarkan pola karakter untuk suatu token. Ketika *lexer* membaca urutan karakter yang sesuai dengan pola tersebut, maka DFA akan mencapai keadaan yang menandakan token dikenali. Dengan demikian, DFA berperan penting dalam membantu proses pengenalan token dalam compiler.

1.5 Token dan Jenis-Jenisnya

Token adalah hasil dari proses analisis leksikal dan merupakan satuan terkecil yang memiliki arti dalam kode sumber. Setiap token memiliki dua bagian utama, yaitu jenis (*type*) dan nilai (*value*).

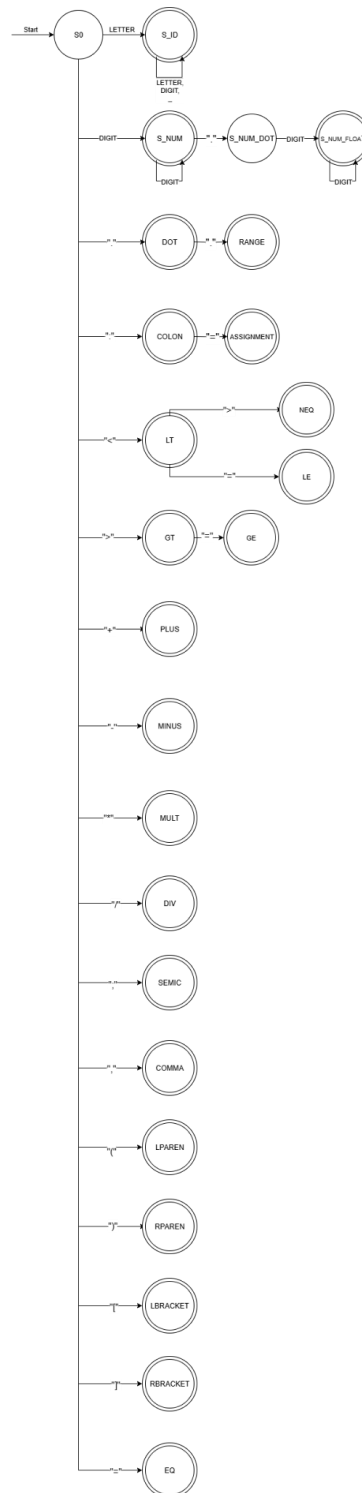
Jenis-jenis token yang digunakan dalam bahasa Pascal-S meliputi:

- Keyword : kata kunci dengan fungsi khusus seperti program, begin, end, if, dan then.
- Identifier : nama variabel, prosedur, atau fungsi yang ditentukan oleh pengguna.
- Operator : simbol yang digunakan untuk operasi aritmetika, logika, atau relasional, misalnya +, -, *, /, :=, <, dan >.
- Literal – nilai konstan seperti angka, karakter, atau string.
- Delimiter : tanda pemisah atau pembatas seperti titik koma, koma, dan tanda kurung.

Token-token ini menjadi masukan bagi parser untuk membangun struktur sintaks program.

2. Perancangan dan Implementasi

2.1 Diagram



Pada diagram DFA yang telah dibuat, diagram ini mendeskripsikan bagaimana setiap karakter dibaca dan diklasifikasikan menjadi jenis token tertentu, berdasarkan aturan transisi antar state yang ada. DFA dimulai dari state awal (S0), yang merepresentasikan kondisi awal sebelum ada karakter dibaca. Berdasarkan jenis karakter pertama yang ditemui, DFA akan berpindah ke *state* lain sesuai kategori input. Setiap *final state* merepresentasikan satu jenis token yang valid.

Dari S0, jika karakter pertama adalah huruf (LETTER), automata akan berpindah ke S_ID. Selama karakter berikutnya merupakan huruf, angka, atau garis bawah, automata tetap berada di S_ID. Ketika menemukan karakter non-valid, maka *lexeme* sebelumnya akan diakui sebagai *identifier* atau *keyword*. Jika karakter pertama digit, automata akan berpindah ke S_NUM. Jika setelah angka ditemukan tanda titik, maka berpindah ke S_NUM_DOT. Jika setelah titik ditemukan digit lagi, automata menuju S_NUM_FLOAT yang merepresentasikan bilangan desimal. Untuk simbol ganda, DFA dibuat membuat transisi berurutan, dan juga sudah mencakup operator tunggal.

2.2 Implementasi

Implementasi Token:

```
from dataclasses import dataclass

@dataclass
class Token:
    type: str
    value: str
    line: int
    column: int

    def __str__(self):
        if self.type == 'STRING_LITERAL':
            return f"{self.type}('{self.value}')"
        else:
            return f"{self.type}({self.value})"
```

Kelas Token menggunakan dataclass untuk menyimpan informasi token yang dihasilkan oleh lexer. Setiap token menyimpan tipe, nilai, dan posisi dalam source code untuk keperluan error reporting yang lebih informatif.

Implementasi DFA Loader:

```

class DFARules:
    def __init__(self, data: Dict[str, Any]):
        self.start_state = data.get('start_state')
        self.final_states = data.get('final_states', {})
        self.transitions = data.get('transitions', {})
        self.keywords = set(k.lower() for k in data.get('keywords',
[]))

    def next_state(self, state: str, ch: str) ->
Tuple[Optional[str], bool]:
        state_map = self.transitions.get(state, {})

        # handle EOF khusus
        if ch == '\0':
            if 'OTHER' in state_map:
                return (state_map['OTHER'], False)
            return (None, False)

        # coba exact match dulu
        if ch in state_map:
            return (state_map[ch], True)

        # coba klasifikasi karakter
        if ch.isalpha() and 'LETTER' in state_map:
            return (state_map['LETTER'], True)
        elif ch.isdigit() and 'DIGIT' in state_map:
            return (state_map['DIGIT'], True)

        # OTHER transition (tanpa konsumsi karakter)
        if 'OTHER' in state_map:
            return (state_map['OTHER'], False)

        return (None, False)

```

Kelas DFARules bertanggung jawab memuat aturan DFA dari file JSON dan menyediakan fungsi transisi. Method `next_state()` mengembalikan tuple berisi state selanjutnya dan boolean yang menunjukkan apakah karakter dikonsumsi atau tidak.

Implementasi Lexer:

```

class Lexer:
    def __init__(self, dfa: DFARules):
        self.dfa = dfa
        self.keywords = dfa.keywords

    def tokenize(self, text: str) -> List[Token]:
        tokens: List[Token] = []
        i = 0
        line = 1
        col = 1

        # tambahkan EOF biar OTHER transitions bisa jalan
        text_with_eof = text + '\0'
        n = len(text_with_eof)

```

```

while i < n - 1: # berhenti sebelum EOF
    ch = text_with_eof[i]

    # skip whitespace
    if ch.isspace():
        if ch == '\n':
            line += 1
            col = 1
        else:
            col += 1
        i += 1
        continue

    # mulai proses DFA
    start_i = i
    start_col = col
    state = self.dfa.start_state
    last_accept_pos = None
    last_accept_state = None

    j = i
    while j < n:
        chj = text_with_eof[j]
        ns, consume = self.dfa.next_state(state, chj)

        if ns is None:
            break

        # update state
        state = ns

        # cek apakah final state
        if self.dfa.is_final(state):
            if consume:
                last_accept_pos = j + 1
            else:
                last_accept_pos = j
            last_accept_state = state

        if consume:
            j += 1
        else:
            # OTHER transition tanpa konsumsi
            j += 1

    if last_accept_pos is not None:
        # pastiin gak masukin EOF di token
        actual_end = min(last_accept_pos, len(text))
        token_info =
self.dfa.get_token_for_final(last_accept_state)
        raw = text[start_i:actual_end]
        tok_type = token_info.get('token')
        tok_value = token_info.get('value')

        # handle string literal khusus
        if tok_type == 'STRING_LITERAL':
            if len(raw) >= 2 and raw[0] == '"' and raw[-1]

```



```

== "'":
    string_content = raw[1:-1].replace("'",
    "'")
    tokens.append(Token('STRING_LITERAL',
string_content, line, start_col))
    else:
        tokens.append(Token('STRING_LITERAL', raw,
line, start_col))
    elif tok_type == 'IDENTIFIER':
        # cek keyword atau identifier biasa
        if raw.lower() in self.keywords:
            tokens.append(Token('KEYWORD', raw, line,
start_col))
        else:
            tokens.append(Token('IDENTIFIER', raw, line,
start_col))
    else:
        value = tok_value if tok_value is not None else
raw
        tokens.append(Token(tok_type, value, line,
start_col))

        # update posisi setelah ambil token
        for c in text[start_i:actual_end]:
            if c == '\n':
                line += 1
                col = 1
            else:
                col += 1
        i = actual_end
        continue

        raise LexerError(f"Unrecognized token starting at line
{line} col {col}: '{ch}'")

return tokens

```

Algoritma lexer menggunakan prinsip longest match dengan backtracking. Setiap karakter diproses melalui DFA, dan lexer akan mengingat posisi terakhir yang mencapai final state untuk menangani situasi dimana tidak ada transisi valid.

Implementasi Compiler:

```

def main():
    parser = argparse.ArgumentParser(description='Pascal-S Lexer
(Milestone 1)')
    parser.add_argument('source', help='Path to Pascal-S source file
(.pas)')
    parser.add_argument('--dfa', default='dfa_rules.json',
help='Path to DFA JSON rules file')
    args = parser.parse_args()

    try:
        dfa = DFARules.from_file(args.dfa)
    except Exception as e:

```

```

        print(f'Gagal load DFA Rules dari {args.dfa}: {e}',
              file=sys.stderr)
        sys.exit(2)

    try:
        with open(args.source, 'r', encoding='utf-8') as f:
            src_text = f.read()
    except Exception as e:
        print(f'Gagal baca file sumber {args.source}: {e}',
              file=sys.stderr)
        sys.exit(2)

    lexer = Lexer(dfa)
    try:
        tokens = lexer.tokenize(src_text)
    except LexerError as le:
        print('Lexer error:', le, file=sys.stderr)
        sys.exit(3)

    for t in tokens:
        print(str(t))

```

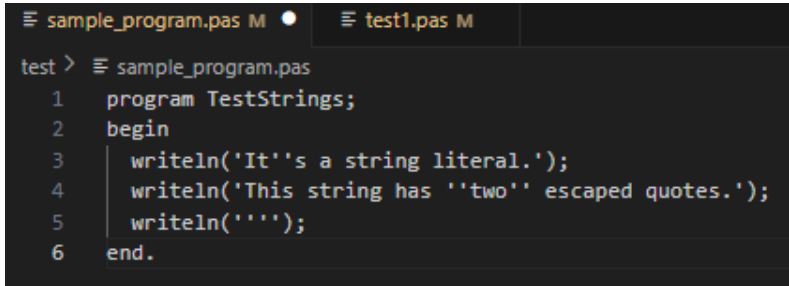
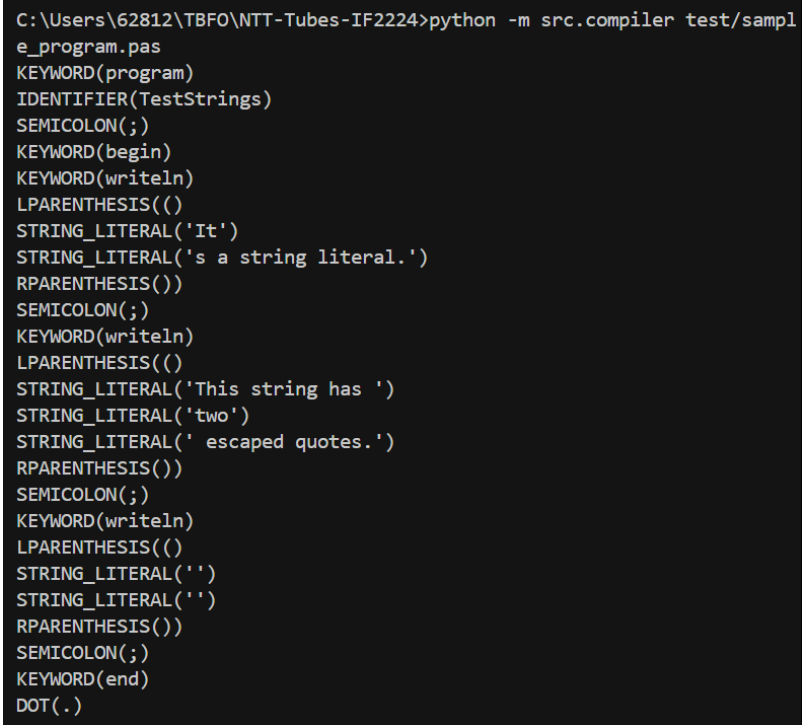
Entry point program menggunakan argparse untuk menangani command line arguments dan menyediakan error handling yang informatif.

Fitur-Fitur di Pengimplementasian:

1. Position Tracking: Lexer melacak nomor baris dan kolom untuk setiap token yang memudahkan debugging.
2. String Literal Handling: Menangani escape sequence untuk single quote (\" menjadi \") dalam string literal.
3. Keyword Recognition: Identifier dicek terhadap daftar keyword Pascal-S untuk klasifikasi yang tepat.
4. EOF Handling: Menggunakan null character (\0) sebagai EOF marker untuk menangani OTHER transitions.
5. Error Recovery: Memberikan pesan error yang informatif dengan posisi yang tepat ketika menemukan karakter yang tidak dikenali.

3. Pengujian

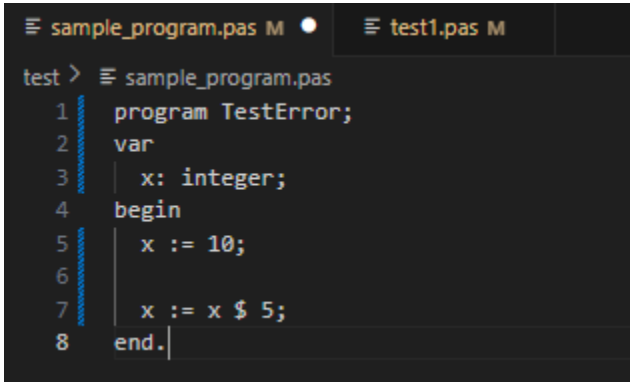
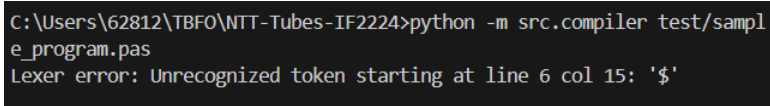
a. Kasus pengujian TestStrings

Input	
Output	
Penjelasan	<p>Pengujian ini bertujuan untuk memvalidasi penanganan <code>STRING_LITERAL</code>, terutama pada kasus escaped quotes (").</p> <p>Output pada screenshot menunjukkan bahwa lexer Anda dirancang untuk memecah string setiap kali ia menemukan escaped quote.</p>

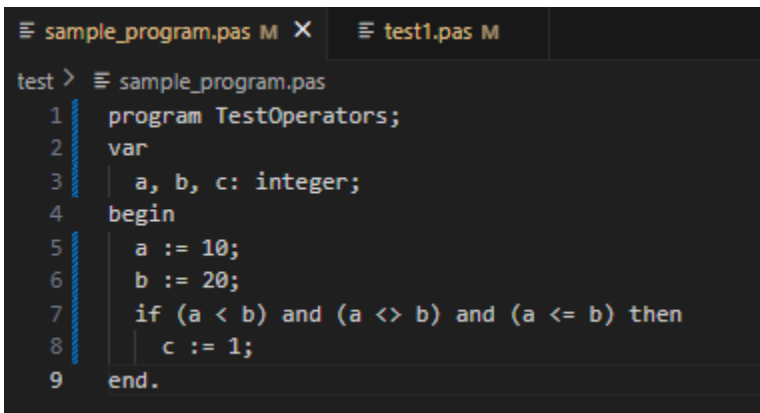
b. Kasus pengujian TestSpacing

Input	<pre> test > milestone1 > ≡ test2.pas 1 PROGRAM TestSpacing; 2 VAR 3 X:INTEGER; 4 BEGIN 5 IF(X=1)THEN 6 X:=2; 7 if (x = 1) then 8 x := 2; 9 END. </pre>
Output	<pre> C:\Users\62812\TBF0\NTT-Tubes-IF2224>python -m src.compiler test/sample_program.pas KEYWORD(PROGRAM) IDENTIFIER(TestSpacing) SEMICOLON(;) KEYWORD(VAR) IDENTIFIER(X) COLON(:) KEYWORD(INTEGER) SEMICOLON(;) KEYWORD(BEGIN) KEYWORD(IF) LPARENTHESIS(() IDENTIFIER(X) RELATIONAL_OPERATOR(=) NUMBER(1) RPARENTHESIS()) KEYWORD(THEN) IDENTIFIER(X) ASSIGN_OPERATOR(:=) NUMBER(2) SEMICOLON(;) KEYWORD(if) LPARENTHESIS(() IDENTIFIER(x) RELATIONAL_OPERATOR(=) NUMBER(1) RPARENTHESIS()) KEYWORD(then) IDENTIFIER(x) ASSIGN_OPERATOR(:=) NUMBER(2) SEMICOLON(;) KEYWORD(END) DOT(.) </pre>
Penjelasan	<p>Baris IF(X=1)THEN tidak memiliki spasi di antara token-tokennya. Output menunjukkan bahwa lexer berhasil memecahnya dengan benar menjadi 7 token: KEYWORD(IF), LPARENTHESIS(() , IDENTIFIER(X), RELATIONAL_OPERATOR(=), NUMBER(1), RPARENTHESIS()), dan KEYWORD(THEN). Input menggunakan keywords dalam huruf besar (PROGRAM, VAR, BEGIN, IF, THEN, END). Output menunjukkan bahwa semua token tersebut dikenali dengan benar sebagai KEYWORD.</p>

c. Kasus pengujian TestError

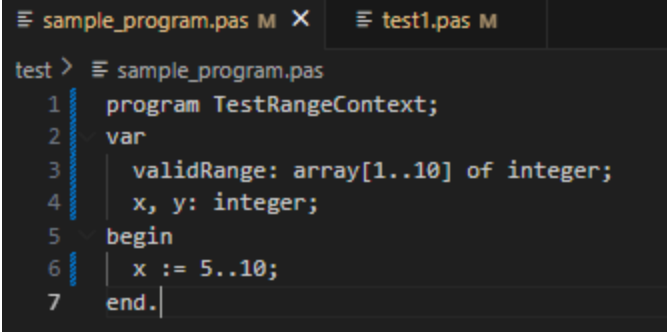
Input	
Output	
Penjelasan	Output menunjukkan bahwa program tidak crash. Sebaliknya, lexer berhasil mengidentifikasi \$ sebagai token yang tidak dikenal dan berhenti, lalu melempar LexerError. Exception ini kemudian ditangkap oleh driver compiler.py, yang mencetak pesan error informatif ke terminal.

d. Kasus pengujian TestOperators

Input	
-------	--

Output	<pre> C:\Users\62812\TBFO\NTT-Tubes-IF2224>python -m src.compiler.test/sample_program.pas KEYWORD(program) IDENTIFIER(TestOperators) SEMICOLON(;) KEYWORD(var) IDENTIFIER(a) COMMA(,) IDENTIFIER(b) COMMA(,) IDENTIFIER(c) COLON(:) KEYWORD(integer) SEMICOLON(;) KEYWORD(begin) IDENTIFIER(a) ASSIGN_OPERATOR(:=) NUMBER(10) SEMICOLON(;) IDENTIFIER(b) ASSIGN_OPERATOR(:=) NUMBER(20) SEMICOLON(;) KEYWORD(if) LPARENTHESIS((IDENTIFIER(a) RELATIONAL_OPERATOR(<) IDENTIFIER(b) RPARENTHESIS()) KEYWORD(and) LPARENTHESIS((IDENTIFIER(a) RELATIONAL_OPERATOR(<>) IDENTIFIER(b) RPARENTHESIS()) KEYWORD(and) LPARENTHESIS((IDENTIFIER(a) RELATIONAL_OPERATOR(<=) IDENTIFIER(b) RPARENTHESIS()) KEYWORD(then) IDENTIFIER(c) ASSIGN_OPERATOR(:=) NUMBER(1) SEMICOLON(;) KEYWORD(end) DOT(.) </pre>
Penjelasan	<p>Lexer mampu mengenali ASSIGN_OPERATOR(:=), RELATIONAL_OPERATOR(<>), dan RELATIONAL_OPERATOR(<=) sebagai token tunggal yang utuh. Ini membuktikan bahwa implementasi driver DFA di lexer.py, khususnya penggunaan last_accept_pos, telah bekerja dengan benar</p>

e. Kasus pengujian TestRangeContext

Input	 <pre> ≡ sample_program.pas M X ≡ test1.pas M test > ≡ sample_program.pas 1 program TestRangeContext; 2 var 3 validRange: array[1..10] of integer; 4 x, y: integer; 5 begin 6 x := 5..10; 7 end. </pre>
Output	<pre> C:\Users\62812\TBFO\WTT-Tubes-IF2224> C:\Users\62812\TBFO\WTT-Tubes-IF2224>python -m src.compiler test/sample_program.pas KEYWORD(program) IDENTIFIER(TestRangeContext) SEMICOLON(;) KEYWORD(var) IDENTIFIER(validRange) COLON(:) KEYWORD(array) LBRACKET([) NUMBER(1) RANGE_OPERATOR(..) NUMBER(10) RBRACKET(]) KEYWORD(of) KEYWORD(integer) SEMICOLON(;) IDENTIFIER(x) COMMA(,) IDENTIFIER(y) COLON(:) KEYWORD(integer) SEMICOLON(;) KEYWORD(begin) IDENTIFIER(x) ASSIGN_OPERATOR(:=) NUMBER(5) RANGE_OPERATOR(..) NUMBER(10) SEMICOLON(;) KEYWORD(end) DOT(.) </pre>
Penjelasan	<p>Input pengujian berisi dua skenario: array[1..10] (di dalam brackets) dan x := 5..10; (di luar brackets).</p>

4. Kesimpulan dan Saran

4.1 Kesimpulan

Sebuah *Lexical Analyzer* (lexer) untuk bahasa Pascal-S telah berhasil diimplementasikan dengan menggunakan model komputasi *Deterministic Finite Automaton* (DFA). Pendekatan yang digunakan adalah dengan membaca aturan transisi DFA dari sebuah file eksternal (dfa_rules.json), yang kemudian disimulasikan oleh sebuah *driver* (lexer.py) untuk mengubah kode sumber .pas menjadi daftar token. Hasil pengujian menunjukkan bahwa *lexer* yang dikembangkan mampu secara efektif mengidentifikasi token yang telah dispesifikasikan. Penerapan DFA terbukti berhasil menangani *Longest Match* dengan baik, seperti yang ditunjukkan pada keberhasilan "TestOperators" dalam mengenali operator multi-karakter. Selain itu, *lexer* juga berhasil membedakan KEYWORD dari IDENTIFIER dan mampu memisahkan token dengan benar meskipun tidak ada *whitespace* di antaranya, seperti yang divalidasi dalam "TestSpacing". Implementasi ini juga didukung oleh fitur-fitur seperti pelacakan posisi baris dan kolom, penanganan *escape sequence* untuk *string literal*, penanganan *End-of-File* (EOF), dan mekanisme *error*.

4.2 Saran

Terdapat beberapa saran yang diajukan, di antaranya manajemen waktu dan pembagian kerja yang lebih efektif.

5. Lampiran

Link Release Repository Github :

Link *workspace* diagram :

<https://drive.google.com/file/d/12FUhx4rAR8moG4VKbKYNQLdQqeyBdEyp/view>

Pembagian Tugas

NIM	Nama	Pembagian Tugas	Presentasi Kontribusi
13523047	Indah Novita Tangdililing	<ul style="list-style-type: none">● Membuat DFA Rules● Implementasi dfa_load.py● Melakukan testing● Membuat laporan	20%
13523049	Muhammad Fithra Rizki	<ul style="list-style-type: none">● Implementasi lexer.py● Membuat DFA Rules● Melakukan testing● Membuat laporan● Membuat state diagram DFA	20%

13523053	Sakti Bimasena	<ul style="list-style-type: none"> ● Implementasi lexer.py ● Membuat DFA Rules ● Melakukan testing ● Membuat laporan ● Membuat state diagram DFA 	20%
13523055	Muhammad Timur Kanigara	<ul style="list-style-type: none"> ● Membuat DFA Rules ● Implementasi token.py ● Melakukan testing ● Membuat laporan 	20%
13523113	Kefas Kurnia Jonathan	<ul style="list-style-type: none"> ● Membuat DFA Rules ● Implementasi compiler.py ● Melakukan testing ● Membuat laporan 	20%

6. 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.

Tim Dosen IF2224 TBFO Teknik Informatika ITB. Diakses pada 15 Oktober 2025.