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

Disusun oleh :

Refki Alfarizi	13523002
Muhammad Aditya Rahmadeni	13523028
Aryo Bama Wiratama	13523088
Fityatul Haq Rosyidi	13523116

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

2025

Daftar Isi

Bab I Landasan Teori

1. Deterministic Finite Automata (DFA)

Deterministic Finite Automata (DFA) adalah model matematis dalam teori komputasi yang digunakan untuk mengenali bahasa reguler. DFA didefinisikan secara formal sebagai *quintuple* $M = (Q, \Sigma, \delta, q_0, F)$, di mana:

- Q merupakan himpunan state (keadaan) yang terbatas, mewakili kondisi mesin pada saat tertentu.
- Σ adalah alfabet input, yaitu himpunan simbol yang dapat diproses.
- $\delta : Q \times \Sigma \to Q$ adalah fungsi transisi yang menentukan state selanjutnya secara unik berdasarkan state saat ini dan simbol input.
- $q_0 \in Q$ adalah state awal, titik mulai pemrosesan.
- $F \subseteq Q$ adalah himpunan state akhir, yang menandakan penerimaan input jika proses berakhir di salah satunya.

Sifat deterministik DFA memastikan bahwa untuk setiap pasangan state dan input, hanya ada satu transisi, sehingga menghindari ambiguitas dan memungkinkan simulasi efisien dengan kompleksitas waktu linier $O(n)$, di mana n adalah panjang string input. DFA dapat dibangun dari Non-deterministic Finite Automata (NFA) melalui proses subset *construction*, yang mengonversi transisi non-deterministik menjadi deterministik. Aplikasi DFA meliputi pengenalan pola dalam *string*, seperti dalam mesin pencari atau validator input, karena kemampuannya memproses data secara *sequential* tanpa *backtracking*.

2. Compiler

Compiler adalah perangkat lunak yang menerjemahkan kode sumber dari bahasa pemrograman tingkat tinggi ke bahasa dengan tingkatan yang lebih rendah sehingga dapat dieksekusi. Proses kompilasi biasanya terdiri dari fase-fase berikut: analisis *front-end* (leksikal, sintaksis, semantik) dan *back-end* (optimasi, generasi kode). Front-end fokus pada pemahaman struktur kode sumber, sementara *back-end* menangani transformasi ke representasi target.

Menurut prinsip desain *compiler*, seperti yang diuraikan dalam buku berjudul *Compilers: Principles, Techniques, and Tools* karya Aho, Sethi, dan Ullman, *compiler* harus menjamin kebenaran semantik, efisiensi eksekusi, dan penanganan kesalahan.

3. Lexical Analyzer

Lexical analyzer, juga dikenal sebagai *lexer* atau *scanner*, adalah komponen awal dalam proses kompilasi yang bertanggung jawab untuk memecah kode sumber menjadi token-token, yaitu unit sintaksis terkecil seperti kata kunci, *identifier*, operator, atau literal. Lexer memproses input

sebagai aliran karakter, mengabaikan elemen non-esensial seperti spasi putih atau komentar, dan menghasilkan *stream* token yang masing-masing memiliki tipe dan nilai.

Dalam teori, *lexer* dapat diimplementasikan menggunakan automata seperti DFA untuk mengenali pola reguler, di mana fungsi transisi DFA diimplementasikan sebagai pemrosesan karakter demi karakter. Keterkaitannya dengan *compiler* terletak pada peran sebagai tahap pertama, yaitu menyediakan input yang sudah teridentifikasi bagi *parser* sintaksis. Lexer juga menangani ambiguitas, seperti membedakan operator dari identifier, melalui aturan *longest match* atau *priority*. Kesalahan leksikal, seperti simbol tidak valid, ditangani dengan mekanisme *error recovery* untuk melanjutkan pemrosesan.

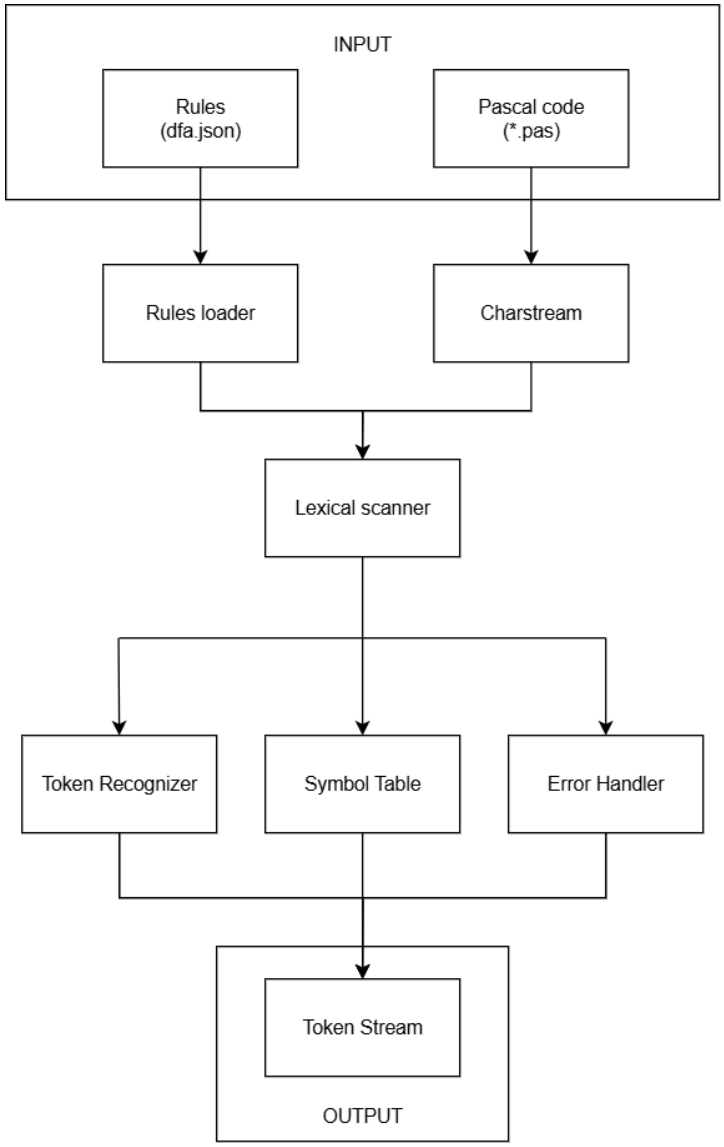
4. Pascal-S

Pascal-S, merupakan varian subset dari bahasa pemrograman Pascal yang disederhanakan untuk tujuan pendidikan pembuatan sistem kompilasi. Dibuat oleh Niklaus Wirth pada 1970, Pascal menekankan sintaksis yang jelas, tipe data statis, dan pencegahan kesalahan saat kompilasi, terinspirasi dari ALGOL. Pascal-S menyederhanakan fitur-fitur yang dimiliki Pascal dengan mempertahankan elemen utama seperti struktur blok, deklarasi variabel eksplisit, serta dukungan untuk prosedur dan fungsi guna mendukung modularitas. Bahasa ini menggunakan tipe data statis untuk keamanan tipe dan mendukung kontrol alur terstruktur serta operator dasar.

Karena sintaksisnya yang reguler, Pascal-S cocok untuk analisis leksikal menggunakan Deterministic Finite Automata (DFA) sehingga memungkinkan pengenalan token yang efisien dalam *pipeline compiler*.

Bab II Perancangan dan Implementasi

1. Arsitektur Program



Gambar 2.1. Diagram arsitektur program

Sistem lexer ini dirancang dengan arsitektur modular yang terdiri dari beberapa komponen utama yang saling berinteraksi. Arsitektur ini memberikan fleksibilitas tinggi karena rules dapat diubah tanpa mengubah kode program serta memudahkan maintenance dan ekstensibilitas sistem. Berikut adalah penjelasan setiap komponen dalam arsitektur:

a. **CharStream**

   CharStream adalah komponen yang bertanggung jawab untuk manajemen pembacaan karakter dari source code. Komponen ini menyimpan seluruh isi source code dalam

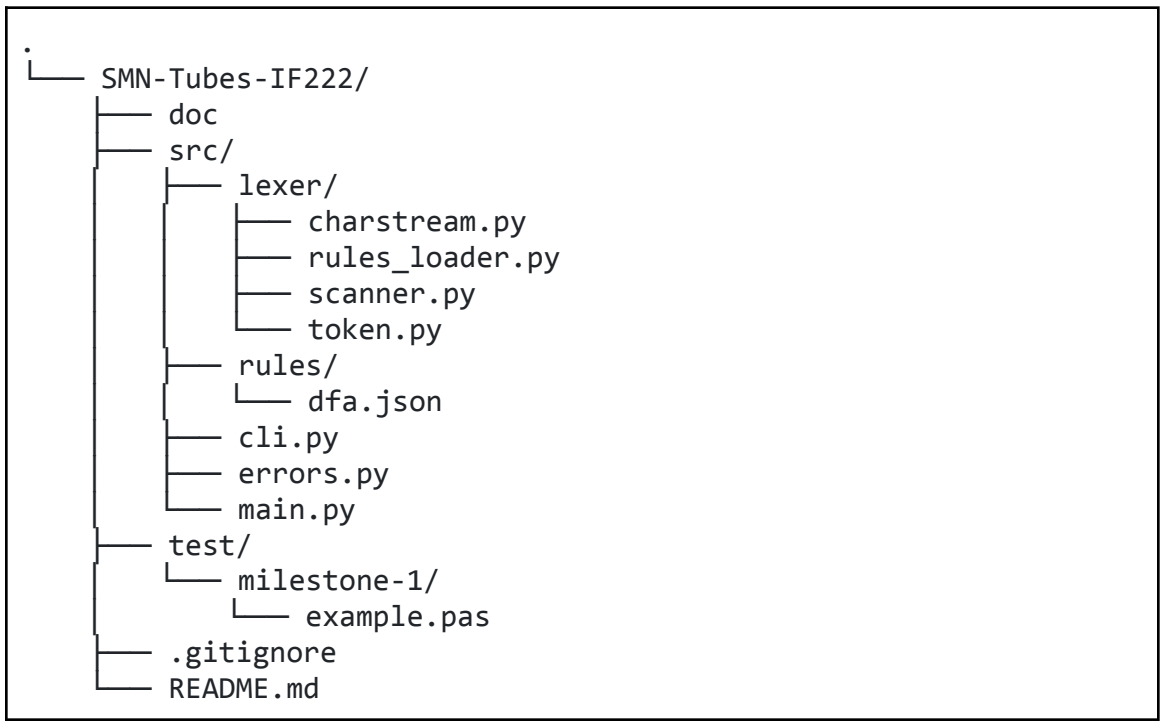
bentuk string dan menyediakan pointer untuk melacak posisi pembacaan saat ini. CharStream dilengkapi dengan method next() untuk membaca dan menggeser pointer dan peek() untuk melihat karakter tanpa menggeser pointer

b. **Rules loader**
   Rules loader membaca file dfa.json dan mengubahnya menjadi struktur data yang dapat digunakan oleh lexer. Komponen ini melakukan parsing JSON, validasi format aturan, dan membangun tabel-tabel internal seperti keyword table (menggunakan hash map untuk lookup cepat), dan regex pattern.
c. **Scanner**
   Scanner adalah inti dari sistem yang mengkoordinasikan seluruh proses tokenisasi. Scanner mengimplementasikan algoritma scanning dengan loop utama yang terus membaca karakter dari CharStream hingga mencapai end-of-file. Untuk setiap karakter, scanner menentukan jenis token yang mungkin berdasarkan karakter awal, kemudian melakukan state transition sesuai DFA hingga mencapai accept state atau error state. Scanner juga menangani whitespace yang perlu diabaikan.
d. **Token recognizer**
   Token Recognizer merupakan salah satu bagian scanner. Komponen ini bertugas untuk melakukan validasi dan klasifikasi token yang telah diidentifikasi oleh state machine. Untuk identifier, recognizer memeriksa apakah identifier tersebut adalah keyword dengan melakukan lookup ke keyword table. Untuk number, recognizer memvalidasi format numerik (integer atau float). Recognizer juga menangani string literals dengan memproses escape sequences dan memvalidasi bahwa string ditutup dengan benar.
e. **Symbol table**
   Symbol Table berfungsi untuk menyimpan hasil ekspansi character classes yang didefinisikan dalam file konfigurasi DFA. Struktur data yang digunakan adalah hash table dengan kompleksitas lookup O(1), sehingga proses pengecekan karakter menjadi sangat efisien. Setiap entry dalam symbol table berisi nama character class sebagai key dan set karakter hasil ekspansi sebagai value. Hasil ekspansi ini digunakan oleh scanner untuk melakukan pattern matching dengan cepat, tanpa perlu melakukan parsing ulang pada setiap karakter yang dibaca.
f. **Error handler**
   Error Handler bertanggung jawab untuk menangani dan melaporkan kesalahan yang terjadi selama proses lexing. Handler mencatat lokasi error dengan presisi (line dan column) dan memberikan pesan error.

Implementasi lexer akan menggunakan bahasa Python karena kemudahan sintaks yang memungkinkan fokus pada logika algoritma, kelengkapan struktur data bawaan (dictionary untuk Symbol Table, set untuk character classes, list untuk token stream), dan library standard yang mendukung seperti json untuk parsing konfigurasi. Python juga memungkinkan development yang cepat tanpa proses kompilasi dan bersifat cross-platform.

2. Struktur repository

Berikut struktur repository untuk mengimplementasikan arsitektur di atas



Repository ini diorganisir secara modular dengan pemisahan yang jelas antara komponen lexer, konfigurasi, dan testing. Berikut adalah penjelasan detail setiap direktori dan file:

a. **src/lexer**
   Package utama lexer yang berisi charstream.py untuk manajemen pembacaan karakter, rules_loader.py untuk loading DFA configuration, scanner.py sebagai core engine, dan token.py untuk definisi token.
b. **src/rules**
   Berisi file dfa.json yang mendefinisikan character classes, token patterns, dan aturan DFA. Pemisahan konfigurasi ini memungkinkan perubahan rules tanpa modifikasi kode.
c. **src/cli.py**
   Modul pendukung untuk command line interface
d. **src/error.py**
   Berisi definisi error yang bisa terjadi
e. **src/main.py**
   File yang berfungsi sebagai entry point program
f. **test/**
   Berisi kumpulan source code dalam bahasa pascal yang berfungsi untuk menguji program
g. **doc/**
   Berisi dokumentasi berupa laporan tentang program

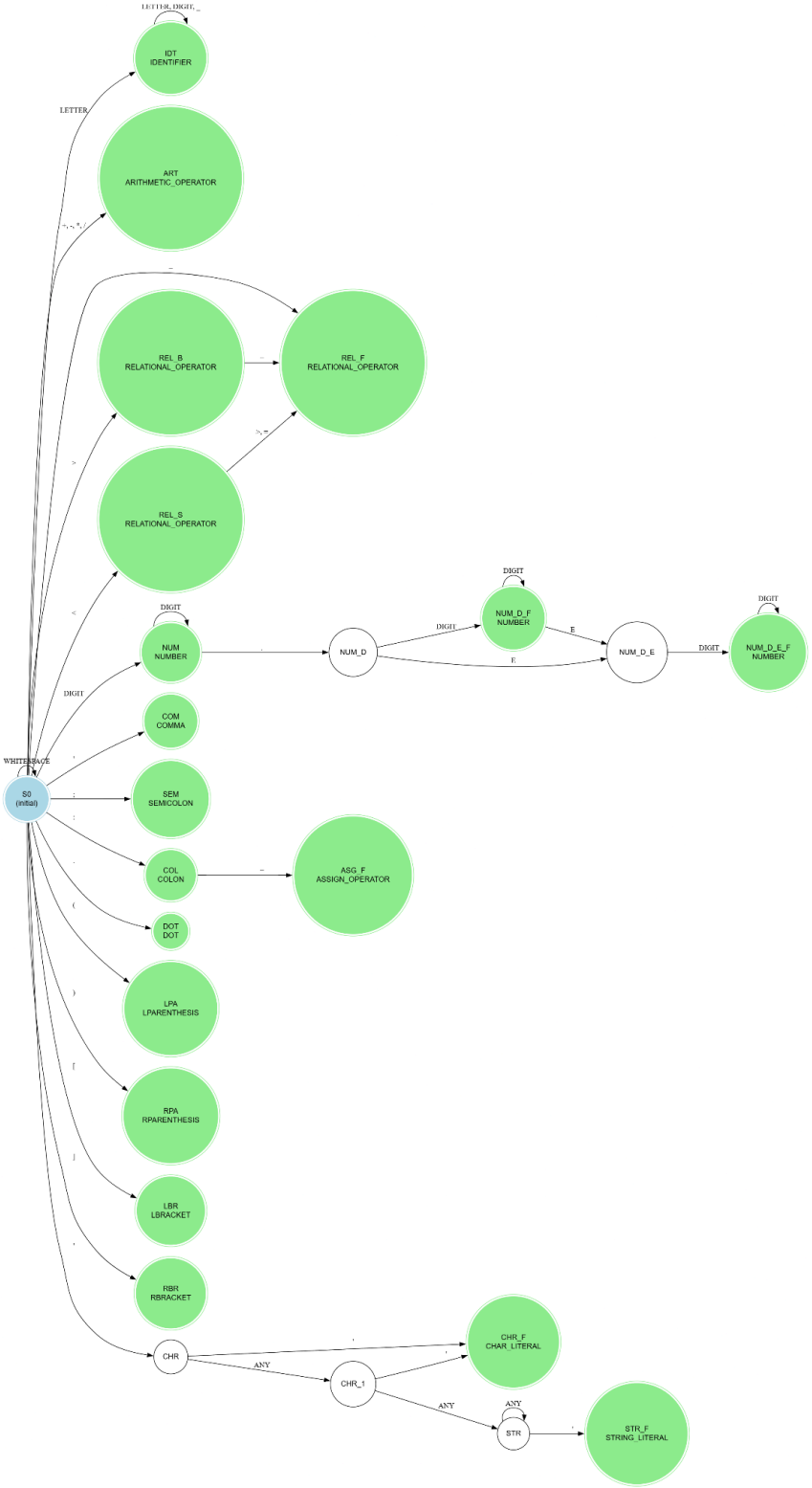
3. Spesifikasi Token

Lexer yang dirancang mampu mengenali berbagai jenis token yang didefinisikan berdasarkan spesifikasi bahasa Pascal-S. Berikut adalah spesifikasi dari setiap jenis token yang didukung oleh sistem.

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

4. DFA



Gambar 2.2 Diagram DFA

Berdasarkan desain diagram DFA diatas, aturan DFA akan diimplementasikan dalam file `.json`. Implementasinya sebagai berikut:

dfa.json

```
{
    "initial_state": "S0",
    "final_states": {
        "IDT": "IDENTIFIER",
        "ART": "ARITHMETIC_OPERATOR",
        "REL_F": "RELATIONAL_OPERATOR",
        "REL_B": "RELATIONAL_OPERATOR",
        "REL_S": "RELATIONAL_OPERATOR",
        "ASG_F": "ASSIGN_OPERATOR",
        "NUM": "NUMBER",
        "NUM_D_F": "NUMBER",
        "NUM_D_E_F": "NUMBER",
        "NUM_D_E_NEG_F": "NUMBER",
        "CHR_F": "CHAR_LITERAL",
        "STR_F": "STRING_LITERAL",
        "COM": "COMMA",
        "SEM": "SEMICOLON",
        "COL": "COLON",
        "DOT": "DOT",
        "LPA": "LPARENTHESIS",
        "RPA": "RPARENTHESIS",
        "LBR": "LBRACKET",
        "RBR": "RBRACKET"
    },
    "transition": {
        "S0": {
            "LETTER": "IDT",
            "DIGIT": "NUM",
            "+": "ART",
            "*": "ART",
            "-": "ART",
            "/": "ART",
            "=": "REL_F",
            "<": "REL_S",
            ">": "REL_B",
            ";": "SEM",
            ",": "COM",
            ".": "DOT",
            "(": "LPA",
            ")": "RPA",
            "[": "LBR",
            "]": "RBR",
            "'": "CHR",
            ":": "COL",
            "WHITESPACE": "S0"
        },
```

```
"IDT": {
    "LETTER": "IDT",
    "DIGIT": "IDT",
    "_": "IDT"
},

"NUM": {
    "DIGIT": "NUM",
    ".": "NUM_D"
},

"NUM_D": {
    "E": "NUM_D_E",
    "DIGIT": "NUM_D_F"
},

"NUM_D_F": {
    "E": "NUM_D_E",
    "DIGIT": "NUM_D_F"
},

"NUM_D_E": {
    "DIGIT": "NUM_D_E_F"
},

"NUM_D_E_F": {
    "DIGIT": "NUM_D_E_F"
},

"REL_S": {
    ">": "REL_F",
    "=": "REL_F"
},

"REL_B": {
    "=": "REL_F"
},

"COL": {
    "=": "ASG_F"
},

"CHR": {
    "ANY": "CHR_1",
    "'": "CHR_F"
},

"CHR_1": {
    "ANY": "STR",
    "'": "CHR_F"
```

```
        },

        "STR": {
            "ANY": "STR",
            "'": "STR_F"
        }
    },

    "char_classes": {
        "LETTER": "a-zA-Z",
        "DIGIT": "0-9",
        "WHITESPACE": "^[ \\t\\n\\r]$",
        "ANY": "^.$"
    },

    "lookup": {
        "program": "KEYWORD",
        "var": "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",
        "function": "KEYWORD",
        "const": "KEYWORD",
        "type": "KEYWORD",
        "procedure": "KEYWORD",
        "begin": "KEYWORD",
        "true": "KEYWORD",
        "false": "KEYWORD",
        "mod": "ARITHMETIC_OPERATOR",
        "div": "ARITHMETIC_OPERATOR",
        "and": "LOGICAL_OPERATOR",
        "or": "LOGICAL_OPERATOR",
        "not": "LOGICAL_OPERATOR"
    }
}
```

Transisi dari state ke state yang lain dapat diabstraksikan menjadi beberapa jenis agar rule mudah dibaca, yaitu LETTER, DIGIT, WHITESPACE, dan ANY.

- LETTER disini merujuk pada huruf alfabet dari a hingga z, kapital maupun non-kapital.
- DIGIT merujuk pada angka dari 0 hingga 9.
- WHITESPACE merujuk pada karakter spesial yang menandakan TAB, NEW LINE, CARRIER RETURN
- ANY ini merujuk pada semua bit mulai dari printable ASCII hingga UNICODE yang merepresentasikan huruf non-alfabet.

Setiap state akhir pada DFA ini merujuk pada satu token yang berbeda, kecuali untuk state akhir IDT. State IDT ini akan digunakan untuk mencari tahu semua token yang berupa huruf, seperti KEYWORD, ARITHMETIC_OPERATOR, LOGICAL_OPERATOR dan IDENTIFIER itu sendiri. Jika suatu input string mencapai state akhir IDT maka string tersebut akan diproses melalui lookup map untuk menentukan token yang cocok dengan string tersebut. Jika tidak ada, maka stringt tersebut akan dianggap IDENTIFIER.

Dengan pendekatan ini, proses klasifikasi token menjadi lebih efisien dan terstruktur, karena DFA hanya perlu mengenali pola karakter, sedangkan pemetaan semantik token dilakukan pada tahap lookup.

Bab III Pengujian

1. example.pas

Kasus uji ini mencakup kasus yang dicantumkan pada dokumen *Spesifikasi Milestone 1* sebagai salah satu contoh hasil yang diharapkan.

Input

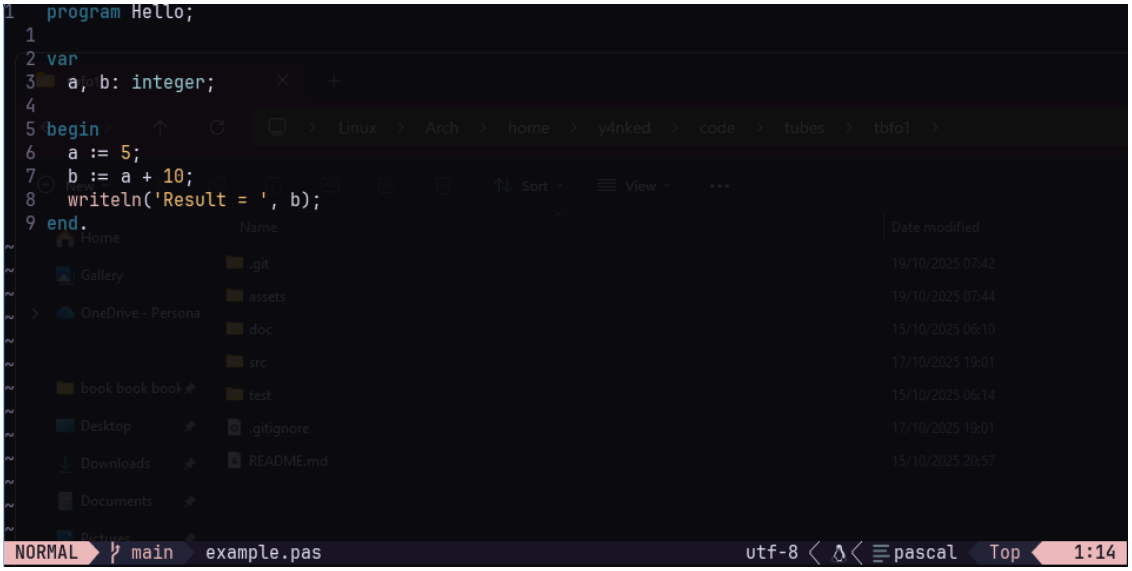
```
program Hello;

var
  a, b: integer;

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

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(.)
```

Bukti Input (Screenshot)



Bukti Output (Screenshot)

```
y4nked  tbfo1  main  python -m src.main test/milestone-1/example.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(.)
```

2. calc.pas

Kasus uji ini mencakup seluruh operator aritmatika. Selain itu, kasus ini juga menguji kemampuan/keterbatasan model DFA dalam mengenali token yang beririsan (tanda "-" yang diikuti angka).

Input

```
program Calc;
var
  a, b, c: integer;
begin
  a := 7 + 3 * 2 -4 / 2 - 3 + (-1);
  b := a div 3;
  c := a mod 4;
  writeln('a = ', a);
```

```
  writeln('b = ', b);
  writeln('c = ', c);
end.
```

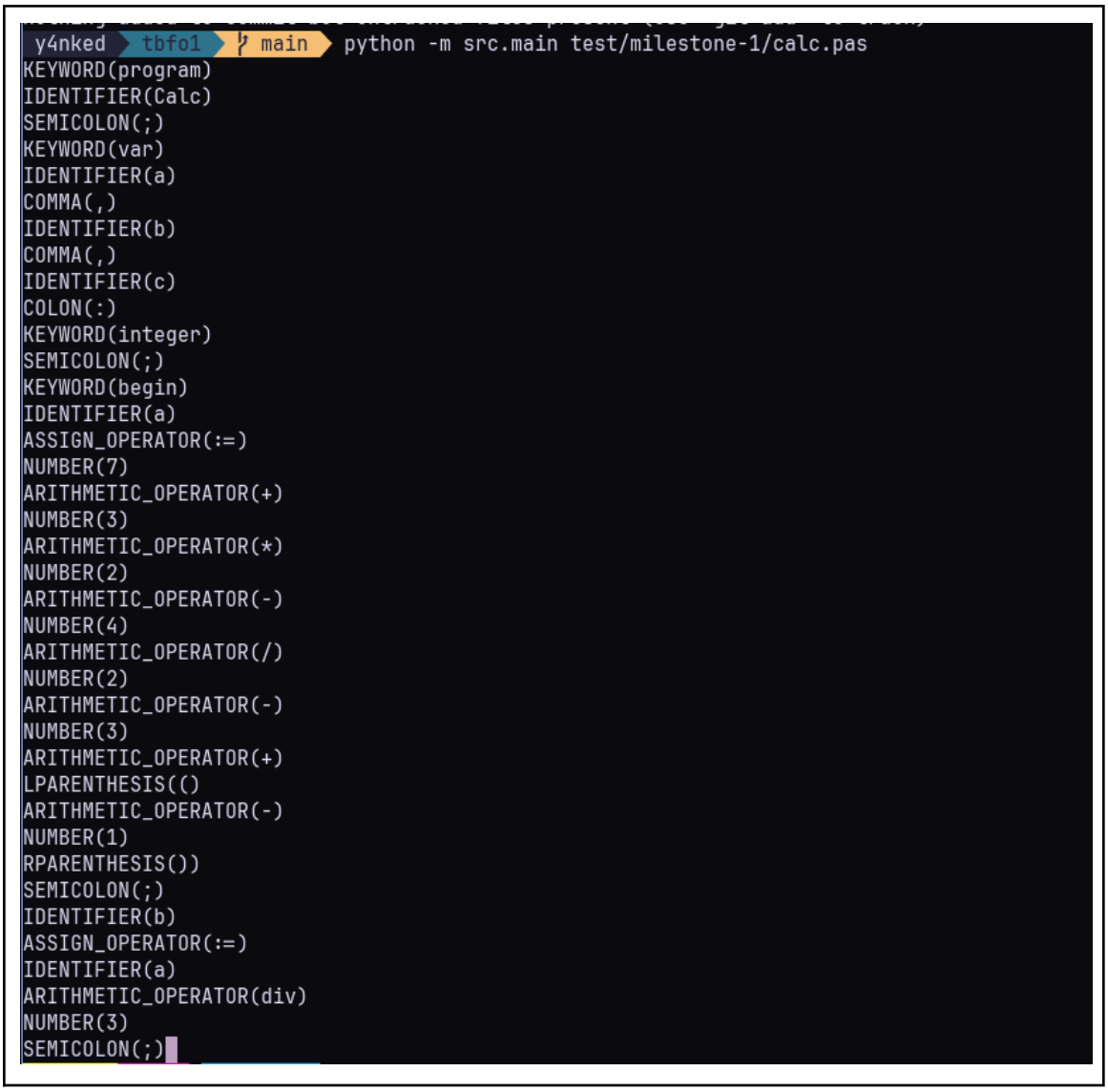
Output

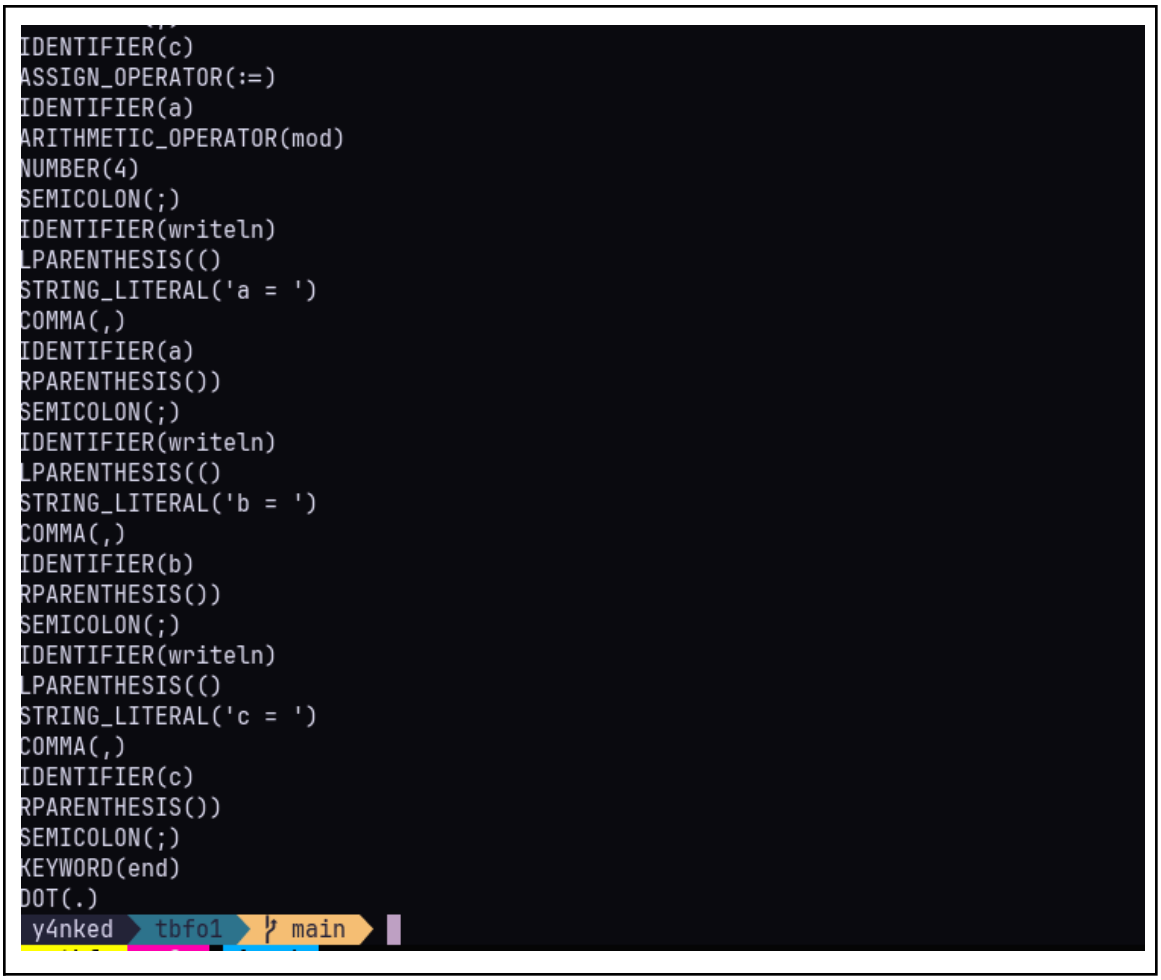
```
KEYWORD(program)
IDENTIFIER(Calc)
SEMICOLON(;)
KEYWORD(var)
IDENTIFIER(a)
COMMA(,)
IDENTIFIER(b)
COMMA(,)
IDENTIFIER(c)
COLON(:)
KEYWORD(integer)
SEMICOLON(;)
KEYWORD(begin)
IDENTIFIER(a)
ASSIGN_OPERATOR(:=)
NUMBER(7)
ARITHMETIC_OPERATOR(+)
NUMBER(3)
ARITHMETIC_OPERATOR(*)
NUMBER(2)
ARITHMETIC_OPERATOR(-)
NUMBER(4)
ARITHMETIC_OPERATOR(/)
NUMBER(2)
ARITHMETIC_OPERATOR(-)
NUMBER(3)
ARITHMETIC_OPERATOR(+)
LPARENTHESIS(()
ARITHMETIC_OPERATOR(-)
NUMBER(1)
RPARENTHESIS())
SEMICOLON(;)
IDENTIFIER(b)
ASSIGN_OPERATOR(:=)
IDENTIFIER(a)
ARITHMETIC_OPERATOR(div)
NUMBER(3)
SEMICOLON(;)
IDENTIFIER(c)
ASSIGN_OPERATOR(:=)
IDENTIFIER(a)
ARITHMETIC_OPERATOR(mod)
NUMBER(4)
SEMICOLON(;)
IDENTIFIER(writeln)
```

```
LPARENTHESIS(()
STRING_LITERAL('a = ')
COMMA(,)
IDENTIFIER(a)
RPARENTHESIS())
SEMICOLON(;)
IDENTIFIER(writeln)
LPARENTHESIS(()
STRING_LITERAL('b = ')
COMMA(,)
IDENTIFIER(b)
RPARENTHESIS())
SEMICOLON(;)
IDENTIFIER(writeln)
LPARENTHESIS(()
STRING_LITERAL('c = ')
COMMA(,)
IDENTIFIER(c)
RPARENTHESIS())
SEMICOLON(;)
KEYWORD(end)
DOT(.)
```

Bukti Input (Screenshot)

Bukti Output (Screenshot baris awal dan akhir)


y4nked tbfo1 main python -m src.main test/milestone-1/calc.pas
KEYWORD(program)
IDENTIFIER(Calc)
SEMICOLON(;)
KEYWORD(var)
IDENTIFIER(a)
COMMA(,)
IDENTIFIER(b)
COMMA(,)
IDENTIFIER(c)
COLON(:)
KEYWORD(integer)
SEMICOLON(;)
KEYWORD(begin)
IDENTIFIER(a)
ASSIGN_OPERATOR(:=)
NUMBER(7)
ARITHMETIC_OPERATOR(+)
NUMBER(3)
ARITHMETIC_OPERATOR(*)
NUMBER(2)
ARITHMETIC_OPERATOR(-)
NUMBER(4)
ARITHMETIC_OPERATOR(/)
NUMBER(2)
ARITHMETIC_OPERATOR(-)
NUMBER(3)
ARITHMETIC_OPERATOR(+)
LPARENTHESIS(()
ARITHMETIC_OPERATOR(-)
NUMBER(1)
RPARENTHESIS())
SEMICOLON(;)
IDENTIFIER(b)
ASSIGN_OPERATOR(:=)
IDENTIFIER(a)
ARITHMETIC_OPERATOR(div)
NUMBER(3)
SEMICOLON(;)



3. float.pas

Kasus uji ini mencakup analisis leksikal pada kode pascal yang didalamnya terdapat token bilangan real (*floating point number*). Kasus uji ini bertujuan menguji kemampuan *lexer* dalam memindai token bilangan real

Input

```
 program Float;
var
  a, b, c, res: Real;
begin
  a := 5.3452;
  b := 2.2213;
  c := 3.4321;

  res := (a + b) * c;
```

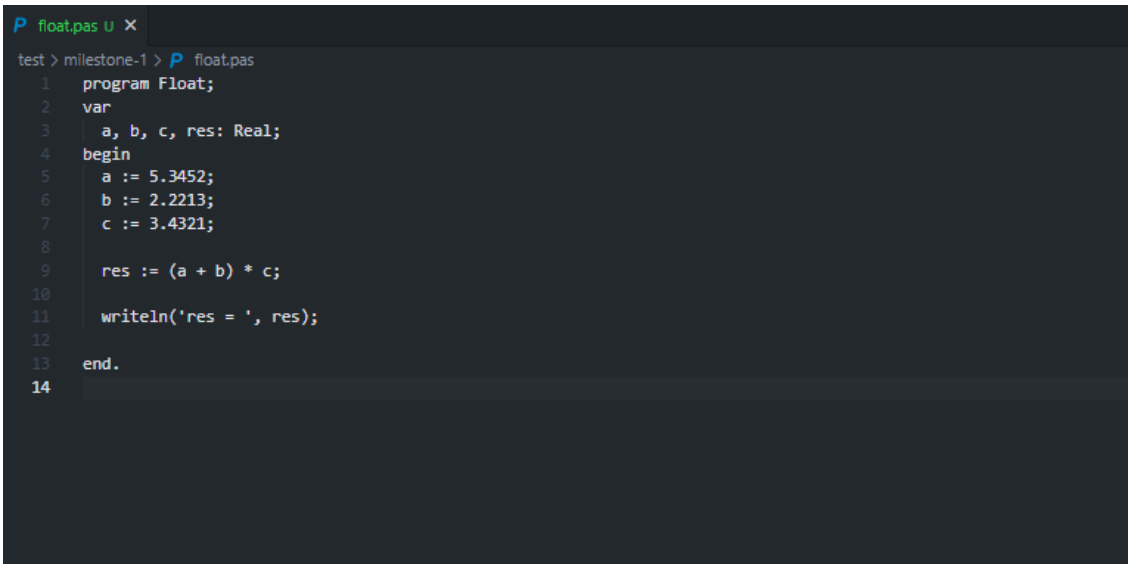
```
  writeln('res = ', res:0:4);

end.
```

Output

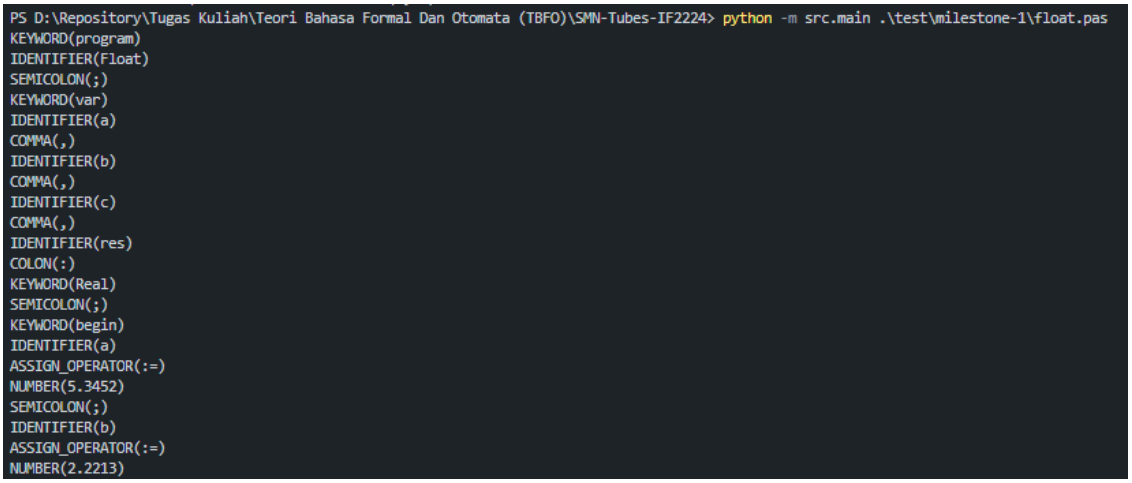
```
KEYWORD(program)
IDENTIFIER(Float)
SEMICOLON(;)
KEYWORD(var)
IDENTIFIER(a)
COMMA(,)
IDENTIFIER(b)
COMMA(,)
IDENTIFIER(c)
COMMA(,)
IDENTIFIER(res)
COLON(:)
KEYWORD(Real)
SEMICOLON(;)
KEYWORD(begin)
IDENTIFIER(a)
ASSIGN_OPERATOR(:=)
NUMBER(5.3452)
SEMICOLON(;)
IDENTIFIER(b)
ASSIGN_OPERATOR(:=)
NUMBER(2.2213)
SEMICOLON(;)
IDENTIFIER(c)
ASSIGN_OPERATOR(:=)
NUMBER(3.4321)
SEMICOLON(;)
IDENTIFIER(res)
ASSIGN_OPERATOR(:=)
LPARENTHESIS(()
IDENTIFIER(a)
ARITHMETIC_OPERATOR(+)
IDENTIFIER(b)
RPARENTHESIS())
ARITHMETIC_OPERATOR(*)
IDENTIFIER(c)
SEMICOLON(;)
IDENTIFIER(writeln)
LPARENTHESIS(()
STRING_LITERAL('res = ')
COMMA(,)
IDENTIFIER(res)
RPARENTHESIS())
SEMICOLON(;)
```

```
KEYWORD(end)
DOT(.)
```

Bukti Input (Screenshot)



Bukti Output (Screenshot)



4. illegal.pas

Kasus uji ini mencakup analisis leksikal pada program Pascal yang mengandung karakter ilegal seperti @, #, dll. Tujuan dari kasus ini adalah menguji kemampuan *error handling* dari *lexer*

Input

`program Illegal;`
`var`
`  x$: integer;`
`begin`
`  x := 10 @ 2;`
`end.`

Output
`LexError at 3:5 - Unexpected character '$' at (3, 4)`
Bukti Input (Screenshot)

illegal.pas U

test > milestone-1 > illegal.pas

```
1  program Illegal;
2  var
3    x$: integer;
4  begin
5    x := 10 @ 2;
6  end.
```

Bukti Output (Screenshot)

5. forreal.pas

Pada kasus uji ini, kami menguji *lexer* dengan kode pascal yang mengandung identifier yang menyerupai keyword, seperti forreal, todo, dan sebagainya. Pengujian dengan kasus ini bertujuan menguji kemampuan *lexer* dalam mengidentifikasi apakah suatu identifier adalah benar identifier walaupun terdapat keyword di subset string nya

Input

```
program Forreal;
var
  offf: integer;
  fofofafa: integer;
```

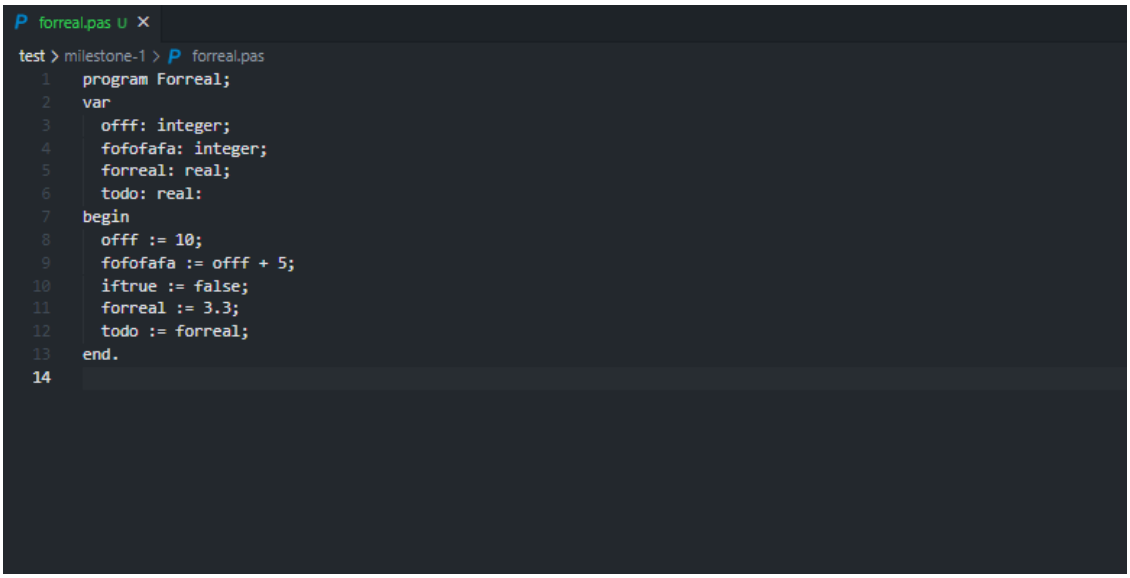
```
  forreal: real;
  todo: real:
begin
  offf := 10;
  fofofafa := offf + 5;
  iftrue := false;
  forreal := 3.3;
  todo := forreal;
end.
```

Output

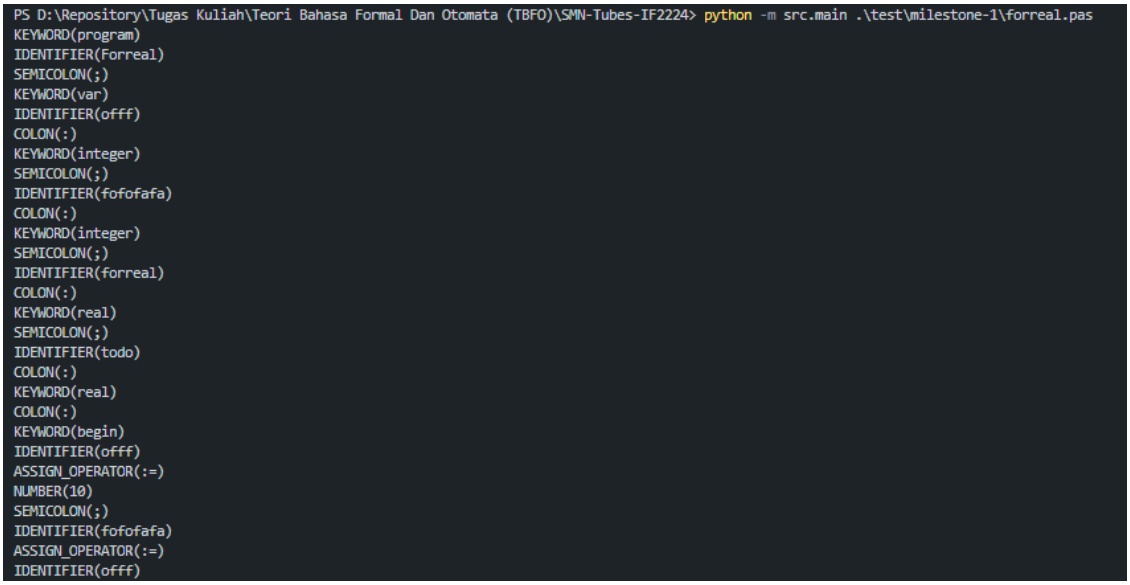
```
KEYWORD(program)
IDENTIFIER(Fofofafa)
SEMICOLON(;)
KEYWORD(var)
IDENTIFIER(offf)
COLON(:)
KEYWORD(integer)
SEMICOLON(;)
IDENTIFIER(fofofafa)
COLON(:)
KEYWORD(integer)
SEMICOLON(;)
IDENTIFIER(forreal)
COLON(:)
KEYWORD(real)
SEMICOLON(;)
IDENTIFIER(todo)
COLON(:)
KEYWORD(real)
COLON(:)
KEYWORD(begin)
IDENTIFIER(offf)
ASSIGN_OPERATOR(:=)
NUMBER(10)
SEMICOLON(;)
IDENTIFIER(fofofafa)
ASSIGN_OPERATOR(:=)
IDENTIFIER(offf)
ARITHMETIC_OPERATOR(+)
NUMBER(5)
SEMICOLON(;)
IDENTIFIER(iftrue)
ASSIGN_OPERATOR(:=)
KEYWORD(false)
SEMICOLON(;)
IDENTIFIER(forreal)
ASSIGN_OPERATOR(:=)
NUMBER(3.3)
SEMICOLON(;)
```

```
IDENTIFIER(todo)
ASSIGN_OPERATOR(:=)
IDENTIFIER(forreal)
SEMICOLON(;)
KEYWORD(end)
DOT(.)
```

Bukti Input (Screenshot)



Bukti Output (Screenshot)



```
ASSIGN_OPERATOR(:=)
IDENTIFIER(offf)
ARITHMETIC_OPERATOR(+)
NUMBER(5)
SEMICOLON(;)
IDENTIFIER(iftrue)
ASSIGN_OPERATOR(:=)
KEYWORD(false)
SEMICOLON(;)
IDENTIFIER(forreal)
ASSIGN_OPERATOR(:=)
NUMBER(3.3)
SEMICOLON(;)
IDENTIFIER(todo)
ASSIGN_OPERATOR(:=)
IDENTIFIER(forreal)
SEMICOLON(;)
KEYWORD(end)
DOT(.)
PS D:\Repository\Tugas Kuliah\Teori Bahasa Formal Dan Otomata (TBFO)\SMN-Tubes-IF2224>
```

6. all.pas

Kasus uji ini mencakup **setidaknya** seluruh nilai token yang terdapat pada tabel daftar token yang tercantum di bagian Spesifikasi Token dan Spesifikasi Milestone 1. Kasus ini bertujuan untuk menguji kemampuan *lexer* dalam mengidentifikasi tipe token yang sudah didefinisikan.

Input

```
program test_full_tokens;

const
  maxVal = 100;
  piVal  = 3.14;
  greet  = 'tbfo';

type
  index = integer;
  realArray = array[1..5] of real;
  letter = char;
  flag = boolean;

var
  x, y, z, sum, avg, count: integer;
  c: char;
  s: string;
  arr: realArray;
  done: boolean;

procedure showMessage(msg: string);
begin
  if not done then
    writeln('Program ', msg, ' seru sekali')
  else
    writeln('Selesai');
```

```
end;

function add(a, b: integer): integer;
begin
  add := a + b;
end;

begin
  x := 22;
  y := 3;
  z := 2018;
  sum := x + y * (z div 10) - (x mod y);
  avg := (x + y + z) / 3.0;

  done := false;

  if (sum >= maxVal) and not done then
    done := true
  else if (sum < maxVal) or (x <> y) then
    done := false;

  if (x <= y) then
    done := false;

  while (x > 0) do
  begin
    x := x - 1;
  end;

  arr[1] := 1.0;
  arr[2] := 2.5555;
  arr[3] := 3.14;
  arr[4] := 4.0;
  arr[5] := 5.0;

  c := 'a';
  c := 'b';
  c := 'c';
  s := 'seru sekali';

  for count := 1 to 5 do
    sum := sum + count;

  for count := 5 downto 1 do
    avg := avg + arr[count];

  showMessage(greet);
  x := add(10, 20);

end.
```

Output

```
KEYWORD(program)
IDENTIFIER(test_full_tokens)
SEMICOLON(;)
KEYWORD(const)
IDENTIFIER(maxVal)
RELATIONAL_OPERATOR(=)
NUMBER(100)
SEMICOLON(;)
IDENTIFIER(piVal)
RELATIONAL_OPERATOR(=)
NUMBER(3.14)
SEMICOLON(;)
IDENTIFIER(greet)
RELATIONAL_OPERATOR(=)
STRING_LITERAL('tbfo')
SEMICOLON(;)
KEYWORD(type)
IDENTIFIER(index)
RELATIONAL_OPERATOR(=)
KEYWORD(integer)
SEMICOLON(;)
IDENTIFIER(realArray)
RELATIONAL_OPERATOR(=)
KEYWORD(array)
LBRACKET([)
NUMBER(1.)
DOT(.)
NUMBER(5)
RBRACKET(])
KEYWORD(of)
KEYWORD(real)
SEMICOLON(;)
IDENTIFIER(letter)
RELATIONAL_OPERATOR(=)
KEYWORD(char)
SEMICOLON(;)
IDENTIFIER(flag)
RELATIONAL_OPERATOR(=)
KEYWORD(boolean)
SEMICOLON(;)
KEYWORD(var)
IDENTIFIER(x)
COMMA(,)
IDENTIFIER(y)
COMMA(,)
IDENTIFIER(z)
COMMA(,)
IDENTIFIER(sum)
COMMA(,)
```

```
IDENTIFIER(avg)
COMMA(,)
IDENTIFIER(count)
COLON(:)
KEYWORD(integer)
SEMICOLON(;)
IDENTIFIER(c)
COLON(:)
KEYWORD(char)
SEMICOLON(;)
IDENTIFIER(s)
COLON(:)
IDENTIFIER(string)
SEMICOLON(;)
IDENTIFIER(arr)
COLON(:)
IDENTIFIER(realArray)
SEMICOLON(;)
IDENTIFIER(done)
COLON(:)
KEYWORD(boolean)
SEMICOLON(;)
KEYWORD(procedure)
IDENTIFIER(showMessage)
LPARENTHESIS(()
IDENTIFIER(msg)
COLON(:)
IDENTIFIER(string)
RPARENTHESIS())
SEMICOLON(;)
KEYWORD(begin)
KEYWORD(if)
LOGICAL_OPERATOR(not)
IDENTIFIER(done)
KEYWORD(then)
IDENTIFIER(writeln)
LPARENTHESIS(()
STRING_LITERAL('Program ')
COMMA(,)
IDENTIFIER(msg)
COMMA(,)
STRING_LITERAL(' seru sekali')
RPARENTHESIS())
KEYWORD(else)
IDENTIFIER(writeln)
LPARENTHESIS(()
STRING_LITERAL('Selesai')
RPARENTHESIS())
SEMICOLON(;)
KEYWORD(end)
SEMICOLON(;)
```

```
KEYWORD(function)
IDENTIFIER(add)
LPARENTHESIS(()
IDENTIFIER(a)
COMMA(,)
IDENTIFIER(b)
COLON(:)
KEYWORD(integer)
RPARENTHESIS())
COLON(:)
KEYWORD(integer)
SEMICOLON(;)
KEYWORD(begin)
IDENTIFIER(add)
ASSIGN_OPERATOR(:=)
IDENTIFIER(a)
ARITHMETIC_OPERATOR(+)
IDENTIFIER(b)
SEMICOLON(;)
KEYWORD(end)
SEMICOLON(;)
KEYWORD(begin)
IDENTIFIER(x)
ASSIGN_OPERATOR(:=)
NUMBER(22)
SEMICOLON(;)
IDENTIFIER(y)
ASSIGN_OPERATOR(:=)
NUMBER(3)
SEMICOLON(;)
IDENTIFIER(z)
ASSIGN_OPERATOR(:=)
NUMBER(2018)
SEMICOLON(;)
IDENTIFIER(sum)
ASSIGN_OPERATOR(:=)
IDENTIFIER(x)
ARITHMETIC_OPERATOR(+)
IDENTIFIER(y)
ARITHMETIC_OPERATOR(*)
LPARENTHESIS(()
IDENTIFIER(z)
ARITHMETIC_OPERATOR(div)
NUMBER(10)
RPARENTHESIS())
ARITHMETIC_OPERATOR(-)
LPARENTHESIS(()
IDENTIFIER(x)
ARITHMETIC_OPERATOR(mod)
IDENTIFIER(y)
RPARENTHESIS())
```

```
SEMICOLON(;)
IDENTIFIER(avg)
ASSIGN_OPERATOR(:=)
LPARENTHESIS(()
IDENTIFIER(x)
ARITHMETIC_OPERATOR(+)
IDENTIFIER(y)
ARITHMETIC_OPERATOR(+)
IDENTIFIER(z)
RPARENTHESIS())
ARITHMETIC_OPERATOR(/)
NUMBER(3.0)
SEMICOLON(;)
IDENTIFIER(done)
ASSIGN_OPERATOR(:=)
KEYWORD(false)
SEMICOLON(;)
KEYWORD(if)
LPARENTHESIS(()
IDENTIFIER(sum)
RELATIONAL_OPERATOR(>=)
IDENTIFIER(maxVal)
RPARENTHESIS())
LOGICAL_OPERATOR(and)
LOGICAL_OPERATOR(not)
IDENTIFIER(done)
KEYWORD(then)
IDENTIFIER(done)
ASSIGN_OPERATOR(:=)
KEYWORD(true)
KEYWORD(else)
KEYWORD(if)
LPARENTHESIS(()
IDENTIFIER(sum)
RELATIONAL_OPERATOR(<)
IDENTIFIER(maxVal)
RPARENTHESIS())
LOGICAL_OPERATOR(or)
LPARENTHESIS(()
IDENTIFIER(x)
RELATIONAL_OPERATOR(<>)
IDENTIFIER(y)
RPARENTHESIS())
KEYWORD(then)
IDENTIFIER(done)
ASSIGN_OPERATOR(:=)
KEYWORD(false)
SEMICOLON(;)
KEYWORD(if)
LPARENTHESIS(()
IDENTIFIER(x)
```

```
RELATIONAL_OPERATOR(<=)
IDENTIFIER(y)
RPARENTHESIS())
KEYWORD(then)
IDENTIFIER(done)
ASSIGN_OPERATOR(:=)
KEYWORD(false)
SEMICOLON(;)
KEYWORD(while)
LPARENTHESIS(()
IDENTIFIER(x)
RELATIONAL_OPERATOR(>)
NUMBER(0)
RPARENTHESIS())
KEYWORD(do)
KEYWORD(begin)
IDENTIFIER(x)
ASSIGN_OPERATOR(:=)
IDENTIFIER(x)
ARITHMETIC_OPERATOR(-)
NUMBER(1)
SEMICOLON(;)
KEYWORD(end)
SEMICOLON(;)
IDENTIFIER(arr)
LBRACKET([)
NUMBER(1)
RBRACKET(])
ASSIGN_OPERATOR(:=)
NUMBER(1.0)
SEMICOLON(;)
IDENTIFIER(arr)
LBRACKET([)
NUMBER(2)
RBRACKET(])
ASSIGN_OPERATOR(:=)
NUMBER(2.5555)
SEMICOLON(;)
IDENTIFIER(arr)
LBRACKET([)
NUMBER(3)
RBRACKET(])
ASSIGN_OPERATOR(:=)
NUMBER(3.14)
SEMICOLON(;)
IDENTIFIER(arr)
LBRACKET([)
NUMBER(4)
RBRACKET(])
ASSIGN_OPERATOR(:=)
NUMBER(4.0)
```

```
SEMICOLON(;)
IDENTIFIER(arr)
LBRACKET([)
NUMBER(5)
RBRACKET(])
ASSIGN_OPERATOR(:=)
NUMBER(5.0)
SEMICOLON(;)
IDENTIFIER(c)
ASSIGN_OPERATOR(:=)
CHAR_LITERAL('a')
SEMICOLON(;)
IDENTIFIER(c)
ASSIGN_OPERATOR(:=)
CHAR_LITERAL('b')
SEMICOLON(;)
IDENTIFIER(c)
ASSIGN_OPERATOR(:=)
CHAR_LITERAL('c')
SEMICOLON(;)
IDENTIFIER(s)
ASSIGN_OPERATOR(:=)
STRING_LITERAL('seru sekali')
SEMICOLON(;)
KEYWORD(for)
IDENTIFIER(count)
ASSIGN_OPERATOR(:=)
NUMBER(1)
KEYWORD(to)
NUMBER(5)
KEYWORD(do)
IDENTIFIER(sum)
ASSIGN_OPERATOR(:=)
IDENTIFIER(sum)
ARITHMETIC_OPERATOR(+)
IDENTIFIER(count)
SEMICOLON(;)
KEYWORD(for)
IDENTIFIER(count)
ASSIGN_OPERATOR(:=)
NUMBER(5)
KEYWORD(downto)
NUMBER(1)
KEYWORD(do)
IDENTIFIER(avg)
ASSIGN_OPERATOR(:=)
IDENTIFIER(avg)
ARITHMETIC_OPERATOR(+)
IDENTIFIER(arr)
LBRACKET([)
IDENTIFIER(count)
```

```
RBRACKET(])
SEMICOLON(;)
IDENTIFIER(showMessage)
LPARENTHESIS(()
IDENTIFIER(greet)
RPARENTHESIS())
SEMICOLON(;)
IDENTIFIER(x)
ASSIGN_OPERATOR(:=)
IDENTIFIER(add)
LPARENTHESIS(()
NUMBER(10)
COMMA(,)
NUMBER(20)
RPARENTHESIS())
SEMICOLON(;)
KEYWORD(end)
DOT(.)
```

Bukti Input (Screenshot awal dan akhir halaman)

```
23    x := x - 1;
22  end;
21
20  arr[1] := 1.0;
19  arr[2] := 2.5555;
18  arr[3] := 3.14;
17  arr[4] := 4.0;
16  arr[5] := 5.0;
15
14  c := 'a';
13  c := 'b';
12  c := 'c';
11  s := 'seru sekali';
10
 9  for count := 1 to 5 do
 8    sum := sum + count;
 7
 6  for count := 5 downto 1 do
 5    avg := avg + arr[count];
 4
 3  showMessage(greet);
 2  x := add(10, 20);
 1
76  end.
NORMAL  main  all.pas                                  utf-8  pascal  Bot  76:4
```

Bukti Output (Screenshot baris awal dan akhir)

```
y4nked  tbfo1  main ±+  python -m src.main test/milestone-1/all.pas
KEYWORD(program)
IDENTIFIER(test_full_tokens)
SEMICOLON(;)
KEYWORD(const)
IDENTIFIER(maxVal)
RELATIONAL_OPERATOR(=)
NUMBER(100)
SEMICOLON(;)
IDENTIFIER(piVal)
RELATIONAL_OPERATOR(=)
NUMBER(3.14)
SEMICOLON(;)
IDENTIFIER(greet)
RELATIONAL_OPERATOR(=)
STRING_LITERAL('tbfo')
SEMICOLON(;)
KEYWORD(type)
IDENTIFIER(index)
RELATIONAL_OPERATOR(=)
KEYWORD(integer)
SEMICOLON(;)
IDENTIFIER(realArray)
RELATIONAL_OPERATOR(=)
KEYWORD(array)
LBRACKET([)
```


RBRACKET(])
SEMICOLON(;)
IDENTIFIER(showMessage)
LPARENTHESIS(()
IDENTIFIER(greet)
RPARENTHESIS())
SEMICOLON(;)
IDENTIFIER(x)
ASSIGN_OPERATOR(:=)
IDENTIFIER(add)
LPARENTHESIS(()
NUMBER(10)
COMMA(,)
NUMBER(20)
RPARENTHESIS())
SEMICOLON(;)
KEYWORD(end)
DOT(.)
y4nked
tbfo1
main ±+

Kesimpulan dan Saran

1. Kesimpulan

Pada *milestone* pertama tugas besar ini, *lexical analyzer* untuk bahasa Pascal-S telah berhasil dikembangkan menggunakan pendekatan *Deterministic Finite Automata* (DFA). *Lexer* yang dibuat mampu memproses kode sumber Pascal-S dan menghasilkan daftar token yang sesuai dengan spesifikasi. Pengujian menunjukkan bahwa program dapat menangani berbagai kasus input serta memberikan pesan *error* yang informatif tanpa menyebabkan *crash*.

Namun, terdapat beberapa kasus *lexer* tidak mampu menghasilkan token yang diharapkan karena keterbatasan konteks/memori pada pendekatan DFA. Contohnya adalah operator "-" dengan angka negatif yang tidak bisa dibedakan karena memerlukan konteks keberadaan token lain.

2. Saran

Untuk *milestone* selanjutnya, disarankan untuk mengembangkan parser sintaksis yang dapat memanfaatkan *output* token dari *lexical analyzer* ini dengan menambahkan mekanisme konteks untuk menangani ambiguitas seperti membedakan operator "-" dari angka negatif melalui integrasi dengan aturan tata bahasa (*grammar*) Pascal-S.

Lampiran

1. Pranala Repositori Github

Pranala Github : Kurondt/SMN-Tubes-IF2224: Tugas Besar 1 IF2224

2. Pranala Workspace Diagram

Pranala Workspace : Graphviz Diagram DFA

3. Pembagian Tugas

NIM	Nama	Tugas	Persentase
13523002	Refki Alfarizi	- Inisiasi projek dan implementasi fungsionalitas dasar - *Testing*	25%
13523028	Muhammad Aditya Rahmadeni	- Merancang aturan DFA - Implementasi *Rule Loader* - Membuat diagram DFA	25%
13523088	Aryo Bama Wiratama	- Implementasi scanner/lexer - Merancang arsitektur program	25%
13523116	Fityatul Haq Rosyidi	- Implementasi algoritma *character stream* - *Testing*	25%

Referensi

[1] Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compilers: Principles, Techniques*, and Tools. Pearson Education. Diakses pada 15 Oktober 2025 dari https://repository.unikom.ac.id/48769/1/Compilers%20-%20Principles,%20Techniques,%20and%20Tools%20(2006).pdf

[2] Wirth, N. (1976). *PASCAL-S: A Subset and its Implementation*. ETH Zürich. Diakses pada 15 Oktober 2025 dari http://pascal.hansotten.com/uploads/pascals/PASCAL-S%20A%20subset%20and%20its%20Implementation%20012.pdf

[3] TutorialsPoint. (n.d.). *Compiler Design - Lexical Analysis*. Diakses pada 15 Oktober 2025 dari https://www.tutorialspoint.com/compiler_design/compiler_design_lexical_analysis.htm

[4] Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation (3rd ed.)*. Pearson Education. Diakses pada 15 Oktober 2025 dari https://cdn-edunex.itb.ac.id/29161-Formal-Language-Theory-and-Automata/1629640939613_Introduction-to-Automata-Theory,-Languages,-and-Computation-Edition-3.pdf