LAPORAN TUGAS BESAR
IF2224 TEORI BAHASA FORMAL DAN OTOMATA

MILESTONE 1 - Lexical Analysis

Disusun oleh

Kelompok 9 - TMP

Bertha Soliany Frandi	13523026
Brian Albar Hadian	13523048
Muhammad Izzat Jundy	13523092
Michael Alexander Angkawijaya	13523102

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2025

DAFTAR ISI

BAB I

LANDASAN TEORI

1.1 Analisis Leksikal

Analisis leksikal adalah tahap pertama dalam proses kompilasi. Pada tahap ini program *compiler* akan membaca kode program yang berupa deretan karakter dan mengelompokkannya menjadi unit-unit bermakna yang disebut token. Token dapat berupa keyword, identifier, operator, literal, atau delimiter. Tujuan dari analisis leksikal adalah menyederhanakan proses parsing dengan mengubah teks mentah menjadi struktur yang lebih mudah diproses. Tahap ini juga bertugas untuk mendeteksi kesalahan karakter dan mengabaikan spasi, tab, atau komentar. Proses bekerja dengan membaca karakter satu per satu dari kiri ke kanan lalu mencocokannya dengan pola token yang valid.

1.2 Deterministic Finite Automata

Deterministic Finite Automata (DFA) adalah model komputasi matematis yang digunakan untuk mengenali pola dalam deretan simbol. Dalam konteks analisis leksikal, DFA digunakan untuk mendeskripsikan bagaimana *lexer* berpindah dari satu *state* ke *state* lain berdasarkan karakter input hingga menemukan token yang valid.

DFA terdiri dari Σ (sigma), Q, q_0, F, dan δ (delta). Sigma melambangkan himpunan simbol input, Q adalah himpunan *state*, q_0 adalah *start date*, F adalah himpunan *final state*, dan delta adalah fungsi transisi. Ketika *lexer* membaca karakter, *lexer* akan menggunakan fungsi transisi untuk berpindah antar *state*. Jika input berakhir di salah satu *final state*, makan urutan karakter tersebut dikenali sebagai token yang valid.

1.3 Pascal-S

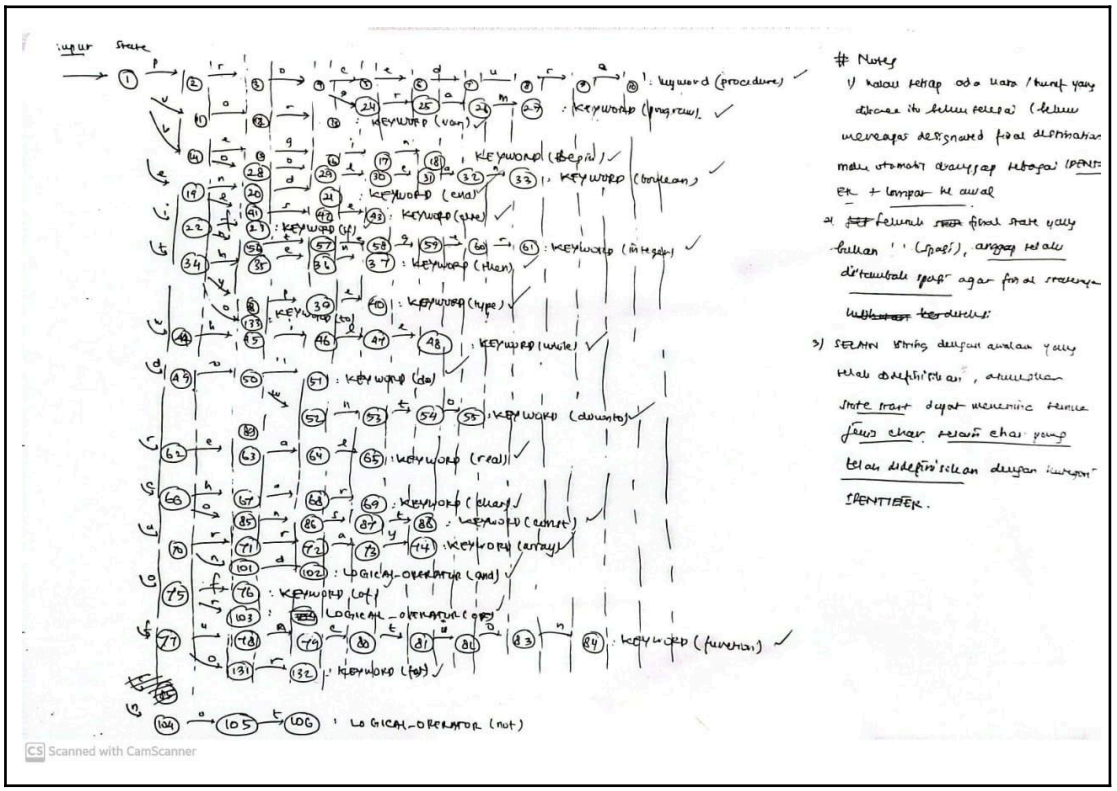
Pascal-S adalah subset dari bahasa pemrograman Pascal. Bahasa ini mempertahankan sebagian besar struktur dasar Pascal seperti begin, end, if, while, dan procedure, namun menghilangkan fitur-fitur kompleks yang tidak esensial. Pascal-S dijadikan sebagai bahasa yang dikompilasi dikarenakan kesederhanaannya tersebut dibandingkan dengan Pascal.

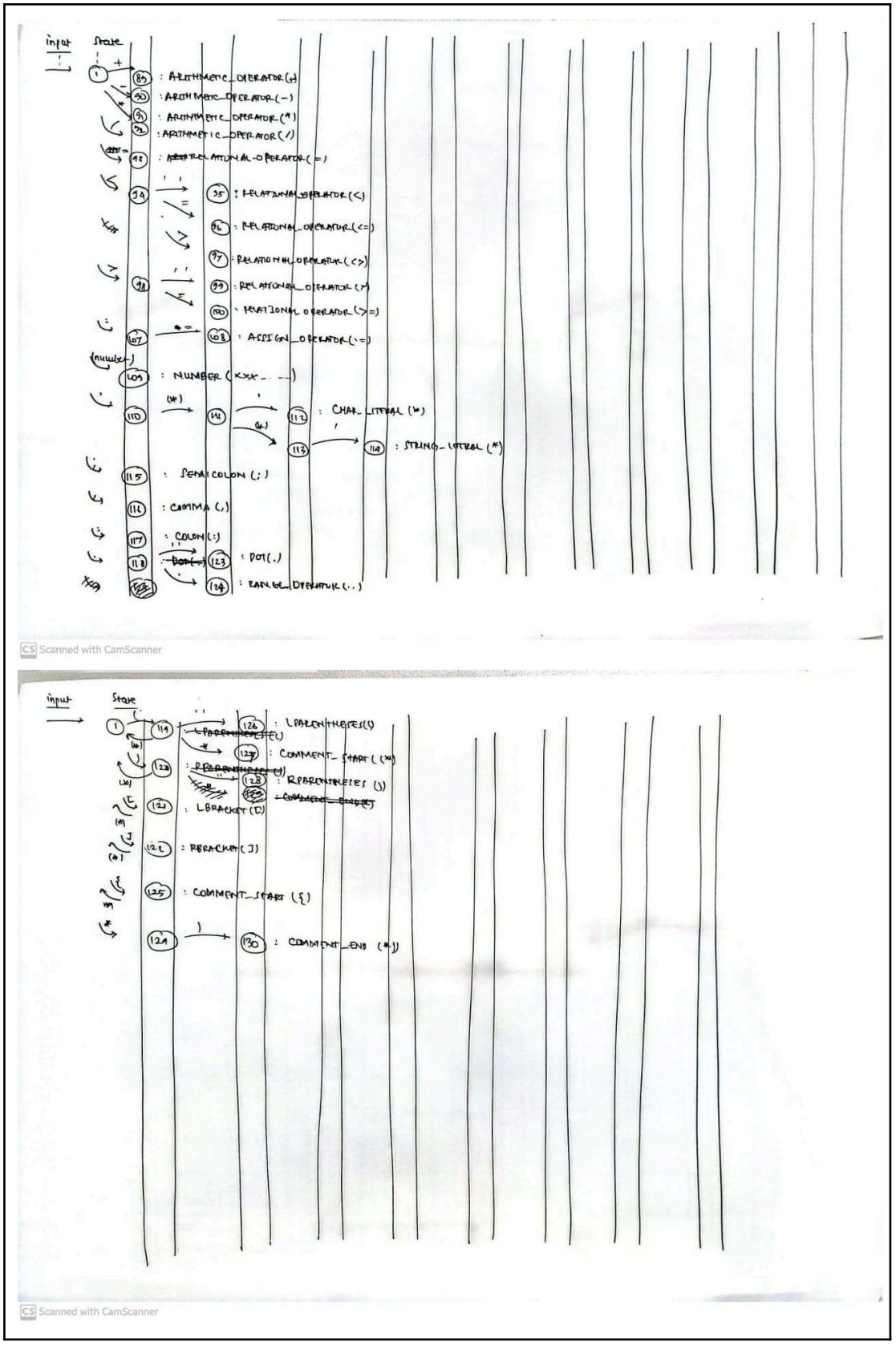
BAB II

PERANCANGAN & IMPLEMENTASI

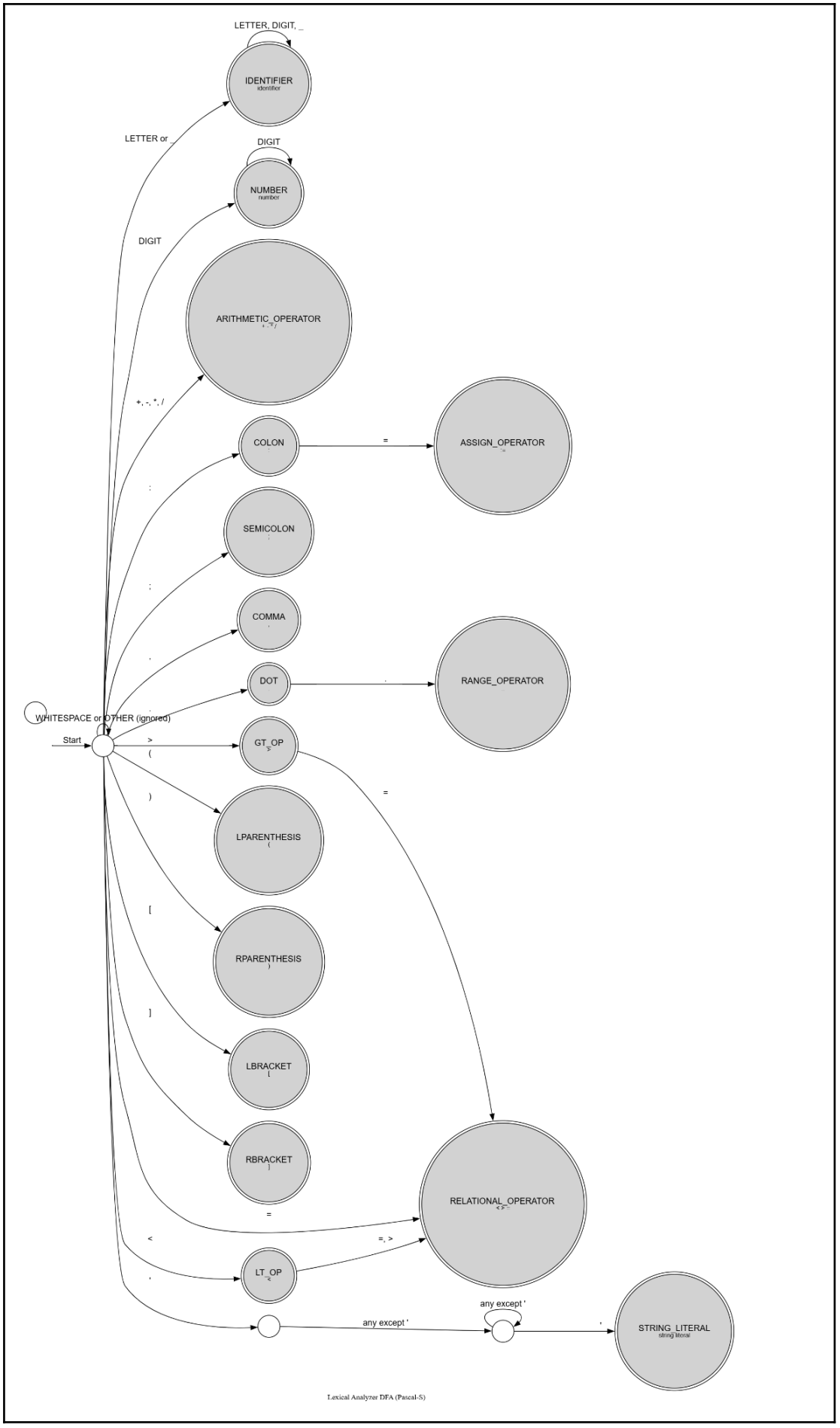
2.1 Diagram DFA

Diagram DFA awalnya dirancang dengan cara tulis tangan dan masih berupa sketsa NFA. DFA dibuat berdasarkan token yang akan dimiliki oleh program. Berikut adalah sketsa awal.





Setelah sketsa, dibuat aturan DFA versi file .txt dan dilakukan konversi ke diagram DFA yang sesuai.



Lexical Analyzer DFA (Pascal-S)

2.2 Implementasi

Implementasi dilakukan dengan menggunakan bahasa Go. Penggunaan bahasa Go dimotivasi dari kecepatannya dalam melakukan kompilasi dibandingkan dengan bahasa lainnya dan penggunaan bahasa ini dapat memberikan kemampuan konkurensi yang cenderung sama cepatnya dengan bahasa C, tetapi memiliki *syntax* penulisan yang bersifat *high-level* sehingga dapat mendorong performa *compiler* dan memberikan kesempatan lebih untuk pengembangan selanjutnya.

Berikut adalah struktur folder dari program. Folder src menyimpan segala program yang akan dibuat dan folder test berisi semua test case. Terdapat pula folder doc yang akan berisi laporan ini dalam bentuk pdf dan diagram DFA. File dfa.txt merupakan file yang berisi aturan DFA.

```
TMP-TUBES-IF2224/
├── src/
│   ├── milestone1/
│   │   ├── dfa.go
│   │   ├── dfa.txt
│   │   ├── lexicalAnalyzer.go
│   │   └── tokenize.go
│   └── main.go
├── test/
│   ├── milestone-1/
│   │   ├── .gitkeep
│   │   ├── test_case1.pas
│   │   ├── test_case2.pas
│   │   ├── test_case3.pas
│   │   ├── test_case4.pas
│   │   ├── test_case5.pas
│   │   └── tester.pas
│   └── output/
│       ├── tokens.txt
│       └── .gitkeep
├── doc/
│   └── Laporan-1-TMP
└── README.md
```

2.2.1 Program

Program memiliki alur utama berupa menerima input, melakukan parsing DFA, melakukan analisis leksikal, dan menampilkan output ke terminal dan menyimpannya di file. Input yang diterima adalah file aturan DFA dan file kode program Pascal-S. Setelah dilihat bahwa input valid, program akan membaca file DFA dan membuat struktur data automata. Barulah program akan melakukan analisis leksikal pada file kode program secara baris per baris dan melakukan tokenization (perubahan dari kode program ke token-token). Setelah itu, hasil dari tokenization akan ditampilkan

pada terminal sekaligus disimpan ke dalam file. Untuk sekarang, implementasi penyimpanan file hanya bisa ke tokens.txt

Berikut adalah penjelasan mengenai fungsi yang terdapat pada program di berbagai file.

Nama Fungsi/Tipe	Penjelasan
func main()	Merupakan titik awal program yang mengatur flow utama program.
type TransitionKey struct	Key untuk melakukan mapping transisi DFA. Menyimpan state dan input yang bertipe string.
type DFA struct	Merupakan struktur utama yang menyimpan state awal automata, state final berupa array of string, dan map untuk transisi antar state
func LexicalAnalyzer	Merupakan fungsi utama yang memproses satu baris. Proses yang dimaksud adalah menghilangkan comment dan whitespace dari baris.
func ProcessToken	Merupakan fungsi yang membaca karakter demi karakter dari start state DFA dan mencari transisi yang valid dari DFA.
func isReadingString	Merupakan helper function untuk mengecek apakah sedang memproses string atau tidak. Ini untuk mencegah berhenti karena spasi.
func removeComments()	Merupakan fungsi untuk menghapus komentar yang ada di kode program Pascal.
func mapCharForDFA	Merupakan fungsi untuk konversi karakter spesial seperti spasi, tab, dan newline, ke format DFA
func Tokenize()	Merupakan fungsi yang mengklasifikasi raw token menjadi kategori yang sesuai.
func isNumber()	Merupakan helper function untuk melakukan validasi apakah token berisi digit atau tidak

BAB III

PENGUJIAN

3.1 Kasus 1

Input	program LoopTest;
	
	var
	  count: integer;
	  done: boolean;
	
	begin
	  count := 0;
	  done := false;
	  while not done do
	  begin
	  count := count + 1;
	  if count >= 5 then
	  done := true;
	  end;
	end.

Output	KEYWORD(program) IDENTIFIER(LoopTest) SEMICOLON(;) KEYWORD(var) IDENTIFIER(count) COLON(:) KEYWORD(integer) SEMICOLON(;) IDENTIFIER(done) COLON(:) KEYWORD(boolean) SEMICOLON(;) KEYWORD(begin) IDENTIFIER(count) ASSIGN_OPERATOR(:=) NUMBER(0) SEMICOLON(;) IDENTIFIER(done) ASSIGN_OPERATOR(:=) KEYWORD(false) SEMICOLON(;) KEYWORD(while) LOGICAL_OPERATOR(not) IDENTIFIER(done) KEYWORD(do) KEYWORD(begin)

	IDENTIFIER(count) ASSIGN_OPERATOR(:=) IDENTIFIER(count) ARITHMETIC_OPERATOR(+) NUMBER(1) SEMICOLON(;) KEYWORD(if) IDENTIFIER(count) RELATIONAL_OPERATOR(>=) NUMBER(5) KEYWORD(then) IDENTIFIER(done) ASSIGN_OPERATOR(:=) KEYWORD(true) SEMICOLON(;) KEYWORD(end) SEMICOLON(;) KEYWORD(end) DOT(.)

3.2 Kasus 2

Input	program Compare;
	
	var
	  x, y: integer;
	
	begin
	  x := 7;
	  y := 10.15;
	  if x < y then
	  x := x * 2
	  else
	  y := y div 2;
	end.

Output	KEYWORD(program) IDENTIFIER(Compare) SEMICOLON(;) KEYWORD(var) IDENTIFIER(x) COMMA(,) IDENTIFIER(y) COLON(:) KEYWORD(integer) SEMICOLON(;) KEYWORD(begin) IDENTIFIER(x)

	ASSIGN_OPERATOR(:=) NUMBER(7) SEMICOLON(;) IDENTIFIER(y) ASSIGN_OPERATOR(:=) NUMBER(10) DOT(.) NUMBER(15) SEMICOLON(;) KEYWORD(if) IDENTIFIER(x) IDENTIFIER(y) KEYWORD(then) IDENTIFIER(x) ASSIGN_OPERATOR(:=) IDENTIFIER(x) ARITHMETIC_OPERATOR(*) NUMBER(2) KEYWORD(else) IDENTIFIER(y) ASSIGN_OPERATOR(:=) IDENTIFIER(y) ARITHMETIC_OPERATOR(div) NUMBER(2) SEMICOLON(;) KEYWORD(end) DOT(.)

3.3 Kasus 3

Input	program Countdown;
	
	var
	  i: integer;
	
	begin
	  for i := 5 downto 1 do
	  writeln('T-minus ', i);
	end.

Output	KEYWORD(program) IDENTIFIER(Countdown) SEMICOLON(;) KEYWORD(var) IDENTIFIER(i) COLON(:) KEYWORD(integer) SEMICOLON(;)

	KEYWORD(begin) KEYWORD(for) IDENTIFIER(i) ASSIGN_OPERATOR(:=) NUMBER(5) KEYWORD(downto) NUMBER(1) KEYWORD(do) IDENTIFIER(writeln) LPARENTHESIS(() STRING_LITERAL('T-minus ') COMMA(,) IDENTIFIER(i) RPARENTHESIS()) SEMICOLON(;) KEYWORD(end) DOT(.)

3.4 Kasus 4

Input	program ArrayDemo;
	
	var
	  scores: array [1..3] of integer;
	  i: integer;
	
	begin
	  for i := 1 to 3 do
	  scores[i] := i * 10;
	  writeln('First = ', scores[1]);
	end.

Output	KEYWORD(program) IDENTIFIER(ArrayDemo) SEMICOLON(;) KEYWORD(var) IDENTIFIER(scores) COLON(:) KEYWORD(array) LBRACKET([) NUMBER(1) RANGE_OPERATOR(..) NUMBER(3) RBRACKET(]) KEYWORD(of) KEYWORD(integer) SEMICOLON(;) IDENTIFIER(i)

	COLON(:) KEYWORD(integer) SEMICOLON(;) KEYWORD(begin) KEYWORD(for) IDENTIFIER(i) ASSIGN_OPERATOR(:=) NUMBER(1) KEYWORD(to) NUMBER(3) KEYWORD(do) IDENTIFIER(scores) LBRACKET([) IDENTIFIER(i) RBRACKET(]) ASSIGN_OPERATOR(:=) IDENTIFIER(i) ARITHMETIC_OPERATOR(*) NUMBER(10) SEMICOLON(;) IDENTIFIER(writeln) LPARENTHESIS(() STRING_LITERAL('First = ') COMMA(,) IDENTIFIER(scores) LBRACKET([) NUMBER(1) RBRACKET(]) RPARENTHESIS()) SEMICOLON(;) KEYWORD(end) DOT(.)

3.5 Kasus 5

Input	program Unterminated;
	
	begin
	  writeln('Hello World);
	end.

Output	KEYWORD(program) IDENTIFIER(Unterminated) SEMICOLON(;) KEYWORD(begin) IDENTIFIER(writeln) LPARENTHESIS(() ERROR('Hello)

	IDENTIFIER(World) RPARENTHESIS()) SEMICOLON(;) KEYWORD(end) DOT(.)

3.6 Kasus 6

Input	program LoopError;
	
	var
	  scores: array [1..10] of integer;
	  i: integer;
	
	begin
	  for i := 1 to 1.5 do
	  scores[i] := i ** 10;
	  writeln('First = ', scores[1]);
	end.

Output	KEYWORD(program) IDENTIFIER(LoopError) SEMICOLON(;) KEYWORD(var) IDENTIFIER(scores) COLON(:) KEYWORD(array) LBRACKET([) NUMBER(1) RANGE_OPERATOR(..) NUMBER(10) RBRACKET(]) KEYWORD(of) KEYWORD(integer) SEMICOLON(;) IDENTIFIER(i) COLON(:) KEYWORD(integer) SEMICOLON(;) KEYWORD(begin) KEYWORD(for) IDENTIFIER(i) ASSIGN_OPERATOR(:=) NUMBER(1) KEYWORD(to) NUMBER(1) DOT(.) NUMBER(5)

	KEYWORD(do) IDENTIFIER(scores) LBRACKET([) IDENTIFIER(i) RBRACKET(]) ASSIGN_OPERATOR(:=) IDENTIFIER(i) ARITHMETIC_OPERATOR(*) ARITHMETIC_OPERATOR(*) NUMBER(10) SEMICOLON(;) IDENTIFIER(writeln) LPARENTHESIS(() STRING_LITERAL('First = ') COMMA(,) IDENTIFIER(scores) LBRACKET([) NUMBER(1) RBRACKET(]) RPARENTHESIS()) SEMICOLON(;) KEYWORD(end) DOT(.)

3.7 Kasus 7

Input	Program WhileError;
	
	var
	  number: integer;
	
	begin
	  while number:<0 do
	  begin
	  writeln('Hello World');
	  number := number + 1;
	  end;
	end.

Output	KEYWORD(Program) IDENTIFIER(WhileError) SEMICOLON(;) KEYWORD(var) IDENTIFIER(number) COLON(:) KEYWORD(integer) SEMICOLON(;) KEYWORD(begin)

	KEYWORD(while) IDENTIFIER(number) COLON(:) RELATIONAL_OPERATOR(<) NUMBER(0) KEYWORD(do) KEYWORD(begin) IDENTIFIER(writeln) LPARENTHESIS(() STRING_LITERAL('Hello World') RPARENTHESIS()) SEMICOLON(;) IDENTIFIER(number) ASSIGN_OPERATOR(:=) IDENTIFIER(number) ARITHMETIC_OPERATOR(+) NUMBER(1) SEMICOLON(;) KEYWORD(end) SEMICOLON(;) KEYWORD(end) DOT(.)

3.8 Kasus 8

Input	program DecimalTest;
	
	var
	  number : integer;
	
	begin
	  number := 20..5;
	end.

Output	KEYWORD(program) IDENTIFIER(DecimalTest) SEMICOLON(;) KEYWORD(var) IDENTIFIER(number) COLON(:) KEYWORD(integer) SEMICOLON(;) KEYWORD(begin) IDENTIFIER(number) ASSIGN_OPERATOR(:=) NUMBER(20) RANGE_OPERATOR(..) NUMBER(5)

	SEMICOLON(;) KEYWORD(end) DOT(.)

BAB IV

KESIMPULAN DAN SARAN

4.1 Kesimpulan

Berdasarkan hasil implementasi yang telah diperoleh, implementasi lexical analyzer yang telah dibuat untuk bahasa pemrograman bahasa Pascal-S telah berhasil dirancang dan dibuat. Pemanfaatan automata DFA sebagai implementasi utama memiliki beberapa kekurangan dan kelebihan, di antaranya adalah rumitnya pemanfaatan state untuk memperoleh suatu kelas kategori tertentu, tetapi memiliki kelebihan bahwa untuk setiap state, dipastikan seluruh masukan memiliki keluaran tertentu.

Milestone 1 mengimplementasikan *lexical analyzer* untuk bahasa Pascal-S menggunakan automata DFA dengan tahapan implementasi dimulai dari perancangan dalam bentuk NFA, yang kemudian diterjemahkan ke dalam bentuk DFA. Program mampu membaca file kode sumber dan menghasilkan daftar token sesuai spesifikasi. Tahapan ini menjadi fondasi untuk milestone selanjutnya dalam pembuatan *compiler*, yakni *syntax analyzer*.

4.2 Saran

1. Melakukan pembagian tugas dengan jelas dan rinci.
2. Tidak menunda pekerjaan.
3. Merancang DFA dengan mempertimbangkan state untuk *lexical error*.

REFERENSI

- https://www.geeksforgeeks.org/compiler-design/introduction-to-compilers/
- https://www.geeksforgeeks.org/compiler-design/introduction-of-lexical-analysis/
- http://pascal.hansotten.com/uploads/pascals/PASCAL-S%20A%20subset%20and%20its%20Implementation%20012.pdf
- https://www.tutorialspoint.com/compiler_design/compiler_design_lexical_analysis.htm
- Aho, et al. (2007). Compilers : Principles, Techniques, & Tools. New York : Pearson Education Inc.

LAMPIRAN

1. Tauran Repository Github
   https://github.com/BrianHadianSTEI23/TMP-Tubes-IF2224

2. Tautan Release Repository Github
   https://github.com/BrianHadianSTEI23/TMP-Tubes-IF2224/tree/v0.1.1

3. Tautan *workspace* Diagram
   graphviz
   miro

4. Pembagian Tugas

Nama	NIM	Tugas
Bertha Soliany Frandi	13523026	Laporan, program, testing
Brian Albar Hadian	13523048	Inisialisasi, program, diagram, testing
Muhammad Izzat Jundy	13523092	Diagram, testing
Michael Alexander Angkawijaya	13523102	Laporan, diagram, testing