Laporan Tugas Besar
IF2224 Teori Bahasa Formal dan Otomata
Milestone 2 - Syntax 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. Context Free Grammar (CFG)

Context Free Grammar (CFG) adalah sebuah *formal grammar* yang *production rule*-nya dapat diterapkan pada sebuah suatu simbol non-terminal tanpa mempedulikan konteks (simbol-simbol lain) di sekitarnya.

Setiap aturan produksi dalam CFG memiliki bentuk $A \rightarrow \alpha$. Dengan A sebagai simbol non-terminal dan α sebagai *string* yang terdiri dari simbol terminal (unit leksikal terkecil/final) dan/atau non-terminal.

Sifat "*context free*" memiliki arti simbol non-terminal di sisi kiri aturan selalu dapat digantikan oleh α di sisi kanan, terlepas simbol lain yang mengelilingi A.

Secara formal, CFG didefinisikan sebagai *4-tuple* $G = (V, T, P, S)$, dengan komponen sebagai berikut:

- V merupakan himpunan terbatas dari simbol non-terminal.
- T merupakan himpunan terbatas dari simbol terminal (terpisah dari V).
- P merupakan himpunan terbatas dari aturan produksi yang memiliki bentuk $A \rightarrow \alpha$.
- $S \in V$ merupakan simbol awal (*start symbol*), yaitu simbol non-terminal yang merepresentasikan keseluruhan *sentence*.

Tata bahasa formal pada dasarnya merupakan kumpulan aturan produksi yang mendeskripsikan seluruh *string* yang dapat dihasilkan dalam suatu bahasa formal. Bahasa yang dihasilkan oleh CFG disebut *Context-Free Language* (CFL). Simbol non-terminal hanya digunakan selama proses derivasi untuk membentuk suatu struktur, simbol tersebut tidak muncul pada *string* akhir.

CFG pertama kali dikembangkan oleh Noam Chomsky untuk memodelkan struktur kalimat bahasa alami. Dalam ilmu komputer, notasi yang paling umum untuk merepresentasikan CFG adalah *Backus Naur Form* (BNF)

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. Syntax Analyzer

Syntax Analyzer atau *parser* biasanya merupakan tahap kedua pada *front-end compiler* setelah *lexical analyzer* (*lexer*). *Parser* menerima kumpulan token yang telah diidentifikasi oleh *lexer*.

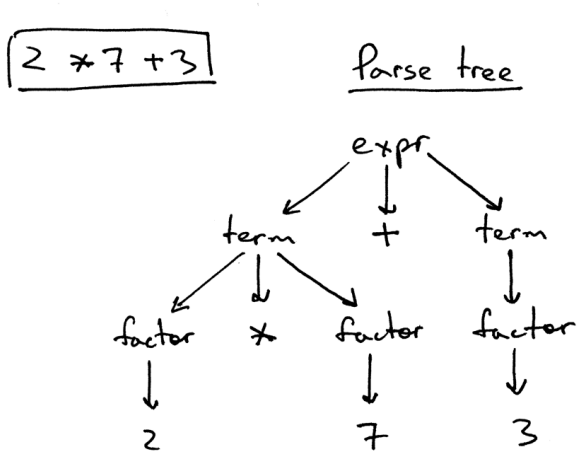
Tujuan utama *parser* adalah memverifikasi bahwa urutan token membentuk suatu struktur gramatikal yang valid sesuai CFG yang ditentukan. *Parser* memeriksa apakah token-token tersebut tersusun menjadi "kalimat" yang valid, seperti pernyataan, ekspresi, atau deklarasi. Selama proses ini, *parser* membangun representasi data hierarkis dari kode sumber (biasanya dalam bentuk *parse tree*). Apabila ditemukan urutan token yang tidak sesuai dengan tata bahasa, *parser* akan mendeteksi dan menyatakan *syntax error*.

4. Parse Tree

Parse Tree adalah representasi proses derivasi suatu string (kumpulan token) dari simbol awal pada CFG yang berbentuk pohon sebagai hasil dari *parser*.

Struktur *parse tree* memiliki ciri sebagai berikut:

- Node akar (*root*), yaitu simbil awal tata bahasa.
- Node internal, yaitu simbol non-terminal.
- Node daun (*leaf*), yaitu simbol terminal.



Gambar Contoh Parse Tree
(Sumber: https://ruslanspivak.com/lsbasi-part7)

Setiap node internal beserta anak-anaknya merepresentasikan satu penerapan aturan produksi. *Parse tree* berguna untuk menunjukkan bagaimana token-token hasil *lexical analysis* membentuk struktur program yang sesuai dengan aturan grammar.

5. Recursive Descent Parser

Recursive descent adalah salah satu strategi *parsing top-down* yang mengimplementasikan *parser* dengan sekumpulan prosedur rekursif yang setiap prosedurnya mengimplementasikan satu simbol non-terminal dari *Context-Free Grammar* (CFG). Struktur program yang dihasilkan secara langsung mencerminkan struktur tata bahasa yang dikenali.

Strategi *recursive descent parser* bekerja dengan cara "menuruni" hierarki aturan produksi dari simbol awal hingga simbol terminal dan melakukan pemanggilan rekursif untuk memproses setiap non-terminal. Setiap prosedur bertugas mencocokkan salah satu aturan produksi yang sesuai dengan non-terminal yang diwakilinya, dengan mengonsumsi token dari aliran masukan secara berurutan.

Dengan strategi ini, aturan produksi yang digunakan haruslah tidak memiliki sifat *left recursion* dan *non-determinism*.

6. 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.

Struktur gramatikal Pascal-S yang terdefinisi dengan rapi menjadikannya sangat cocok untuk dianalisis menggunakan teknik *parsing top-down* seperti *recursive descent*.

Bab II Perancangan dan Implementasi

1. Arsitektur Program

Sistem parser ini dirancang dengan arsitektur modular yang terdiri dari beberapa komponen utama yang saling berinteraksi. Berikut adalah penjelasan setiap komponen dalam arsitektur:

1. **ParseTree**
   Berisi Implementasi struktur data parsetree yang akan digunakan pada algoritma recursive descent parsing.

2. **Parser**
   Ini adalah komponen utama pada milestone 2 ini. Berisi kelas Parser yang memiliki atribut list of token yang diperoleh dari lexical analysis sebelumnya. Kelas ini berisi method-method helper seperti peek() untuk mendapat token yang diproses saat ini, dan match(token) untuk mencocokkan token saat ini dengan token input, dll. Selain itu, kelas ini juga berisi semua grammar yang di hardcode untuk melakukan parsing recursive descent.

2. Struktur repository

Berikut struktur repository untuk mengimplementasikan arsitektur di atas

```
.
└── SMN-Tubes-IF222/
    ├── doc
    ├── src/
    │   ├── lexer/
    │   │   ├── charstream.py
    │   │   ├── rules_loader.py
    │   │   ├── scanner.py
    │   │   └── token.py
    │   ├── parser/
    │   │   ├── parsetree.py
    │   │   └── parser.py
    │   ├── rules/
    │   │   └── dfa.json
    │   ├── cli.py
    │   ├── errors.py
    │   └── main.py
    ├── test/
    │   └── milestone-1/
```

```
│       └── example.pas
├── .gitignore
└── README.md
```

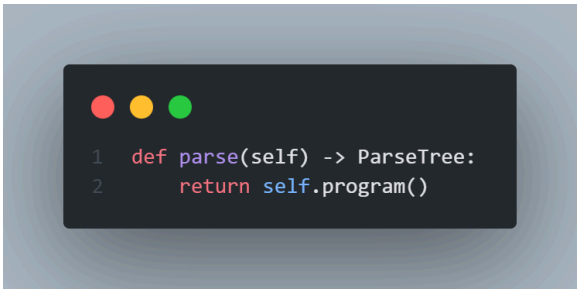
Repository ini diorganisir secara modular dengan pemisahan yang jelas antara komponen lexer, konfigurasi, dan testing. Berikut adalah penjelasan detail setiap direktori dan file:

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/parser**
   Package utama parser yang berisi parsetree.py berisi kelas parsetree dan parser.py yang berisi implementasi grammar dan algoritma recursive descent .

c. **src/rules**
   Berisi file dfa.json yang mendefinisikan character classes, token patterns, dan aturan DFA. Pemisahan konfigurasi ini memungkinkan perubahan rules tanpa modifikasi kode.
d. **src/cli.py**
   Modul pendukung untuk command line interface
e. **src/error.py**
   Berisi definisi error yang bisa terjadi
f. **src/main.py**
   File yang berfungsi sebagai entry point program
g. **test/**
   Berisi kumpulan source code dalam bahasa pascal yang berfungsi untuk menguji program
h. **doc/**
   Berisi dokumentasi berupa laporan tentang program

3. Algoritma Recursive Descent

Algoritma recursive diimplementasikan pada kelas Parser, lebih tepatnya pada method-method yang berisi grammar. Terdapat total 35 grammar yang dihardcode pada kelas ini. Root dari parsetree yang akan dibentuk adalah suatu node bernama <program>, node ini akan dipanggil pada suatu Entrypoint yaitu method parser(), dan method inilah yang akan menghasilkan parsetree keseluruhan dari program yang diinput.



```
def parse(self) -> ParseTree:
    return self.program()
```

Gambar : Entrypoint parser

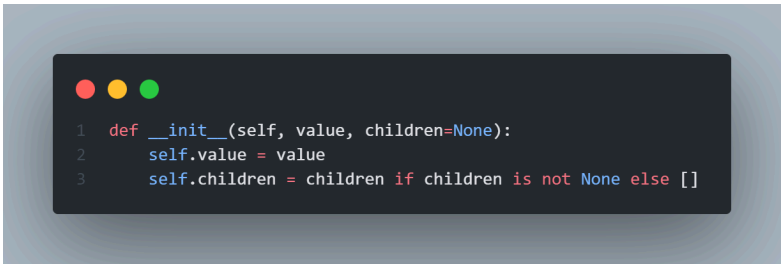
```
# program -> program_header + declaration_part + compound_statement + DOT
def program(self) -> ParseTree :
    node = ParseTree("<program>")

    node.add_child(self.program_header())
    node.add_child(self.declaration_part())
    node.add_child(self.compound_statement())
    node.add_child(self.consume("DOT"))
```

Gambar : grammar utama <program>

Dapat dilihat pada Grammar program diatas memanggil 3 grammar lain (variable non-terminal) dan 1 variabel terminal “DOT” . awalnya dibentuk node dengan label <program>, kemudian untuk setiap grammar/variabel non-terminal di RHS, fungsinya langsung dipanggil dan dijadikan children untuk node <program>. Untuk setiap variabel terminal di RHS, akan digunakan method consume yang akan membuat node misalnya bernama “DOT” dan memastikan token saat ini memang “DOT”, lalu jika iya, node tersebut dijadikan anak node <program>.

4. Parse Tree

Terdapat kelas ParseTree yang merepresentasikan struktur data parsetree. Kelas ini nantinya digunakan untuk algoritma recursive descent. Kelas ini memiliki dua atribut yaitu value dan children



Gambar : atribut kelas ParseTree

Value adalah nama dari node parsetree tersebut, sedangkan children adalah list yang berisi node-node anak.

Beberapa method penting pada kelas ini antara lain yaitu add_child(ParaseTree) yang berfungsi menambahkan node baru ke list children, dan pretty_print() untuk menampilkan ParseTree yang dihasilkan algoritma recursive dengan format yang lebih elegan

5. Aturan Grammar

Terdapat beberapa penyesuaian grammar/aturan produksi dengan yang diberikan pada spesifikasi untuk memenuhi kebutuhan saat pengembangan *parser*.

Berikut merupakan daftar grammar/aturan produksi beserta *node*, deskripsi, dan contoh yang dihasilkan.

No	Aturan Produksi	Nama Node yang Dihasilkan	Deskripsi	Contoh
1	program → program-header + declaration-part + compound-statement + DOT	<program>	Node root yang merepresentasikan keseluruhan program Pascal-S	program Hello; … mulai … selesai .
2	program_header -> KEYWORD(program) + IDENTIFIER + SEMICOLON	<program_header>	Bagian kepala program yang berisi nama program	program Hello;
3	declaration_part -> (const-declaration)* + (type-declaration)* + (var-declaration)* +	<declaration_part>	Bagian deklarasi yang dapat berisi const, type, var, procedure, function (urutan *fixed*)	konstanta MAX = 100; tipe T = integer; variabel a: integer; prosedur p;

	(subprogram-declaration)*			
4	const_declaration -> KEYWORD("konstanta") (IDENTIFIER RELATIONAL_OPERATOR("=") (NUMBER | STRING_LITERAL | CHAR_LITERAL) SEMICOLON)+	<const_declaration>	Deklarasi konstanta	konstanta MAX = 100; PI = 3.14;
5	type_declaration -> KEYWORD("tipe") (IDENTIFIER RELATIONAL_OPERATOR("=") type_definition SEMICOLON)+	<type_declaration>	Deklarasi tipe data baru	tipe Range = 1..10;
6	var_declaration -> KEYWORD("variabel") (identifier_list COLON type SEMICOLON)+	<var_declaration>	Deklarasi variabel	variabel a, b: integer;
7	identifier_list -> IDENTIFIER (COMMA IDENTIFIER)*	<identifier_list>	Daftar identifier yang dipisahkan koma	a, b, c
8	type_definition -> type | record_type	<type_definition>	Tipe = primitive/array/rekaman	integer / rekaman ... selesai
9	type -> array_type | KEYWORD(integer |char|boolean|Real)	<type>	Tipe dasar atau array.	integer
10	array_type -> KEYWORD("larik") LBRACKET range RBRACKET KEYWORD("dari") type	<array_type>	Tipe larik.	larik[1..10] dari integer

11	record_type -> KEYWORD("rekaman") (identifier_list COLON type SEMICOLON)+ KEYWORD("selesai")	<record_type>	Tipe rekaman (record).	rekaman a: integer; selesai
12	range -> expression RANGE_OPERATOR("..") expression	<range>	Rentang / subrange.	1..10
13	subprogram_declaration -> procedure_declaration | function_declaration	<subprogram_declaration>	Prosedur atau fungsi.	-
14	procedure_declaration -> KEYWORD("prosedur") IDENTIFIER (formal_parameter_list)? SEMICOLON block SEMICOLON	<procedure_declaration>	Deklarasi prosedur.	prosedur p(); mulai ... selesai ;
15	function_declaration -> KEYWORD("fungsi") IDENTIFIER (formal_parameter_list)? COLON type SEMICOLON block SEMICOLON	<function_declaration>	Deklarasi fungsi.	fungsi f(): integer; mulai ... selesai ;
16	formal_parameter_list -> LPARENTHESIS parameter_group (SEMICOLON parameter_group)* RPARENTHESIS	<formal_parameter_list>	Parameter formal.	(a: integer)
17	parameter_group -> identifier_list COLON type	<parameter_group>	Satu grup parameter.	x, y: real

18	compound_statement -> KEYWORD("mulai") statement_list KEYWORD("selesai")	<compound_statement>	Blok statement.	mulai ... selesai
19	block -> (declaration_part)? compound_statement	<block>	Bagian subprogram.	-
20	statement_list -> statement (SEMICOLON statement)* (stop jika ; diikuti selesai)	<statement-list>	Daftar statement.	x := 1; y := 2
21	statement -> compound_statement | if_statement | while_statement | for_statement | assignment_statement | procedure_or_function_call	<statement>	Segala bentuk statement.	x := 2
22	assignment_statement -> IDENTIFIER (array_bucket)? ASSIGN_OPERATOR(":=") expression	<assignment-statement>	Assignment.	a := 3
23	array_bucket -> LBRACKET (NUMBER | IDENTIFIER) RBRACKET	<array-bucket>	Index array.	[i]
24	if_statement -> KEYWORD("jika") expression KEYWORD("maka") statement (KEYWORD("selain_itu") statement)?	<if-statement>	If-else.	jika x>0 maka ...

25	while_statement -> KEYWORD("selama") expression KEYWORD("lakukan") statement	<while-statement>	Perulangan while.	selama x<10 lakukan ...
26	for_statement -> KEYWORD("untuk") IDENTIFIER ASSIGN_OPERATOR(":=") expression (KEYWORD("ke") | KEYWORD("turun_ke")) expression KEYWORD("lakukan") statement	<for-statement>	For naik/turun.	untuk i:=1 ke 10 lakukan ...
27	procedure_or_function_call -> IDENTIFIER LPARENTHESIS (parameter_list)? RPARENTHESIS	<procedure/function-call>	Pemanggilan prosedur/fungsi.	p(1,2)
28	parameter_list -> (expression | STRING_LITERAL | CHAR_LITERAL) (COMMA expression)*	<parameter-list>	Parameter aktual.	1, "abc"
29	expression -> simple_expression (RELATIONAL_OPERATOR simple_expression)?	<expression>	Ekspresi relational.	a + b < c
30	simple_expression -> (ARITHMETIC_OPERATOR("+")|"-")? term (additive_operator term)*	<simple-expression>	Term + (+ / - / atau).	a - b
31	term -> factor (multiplicative_oper	<term>	Faktor * / bagi / mod / dan.	a * b

	ator factor)*			
32	factor -> IDENTIFIER (LPARENTHESIS ...) | IDENTIFIER array_bucket | IDENTIFIER | NUMBER | CHAR_LITERAL | STRING_LITERAL | LPARENTHESIS expression RPARENTHESIS | LOGICAL_OPERATOR("tidak") factor | KEYWORD("true"| "false")	<factor>	Faktor.	(a+b)
33	relational_operator -> RELATIONAL_OPERATOR("=" | "<>" | "<" | "<=" | ">" | ">=")	<relational-operator>	Operator relasi.	<>
34	additive_operator -> ARITHMETIC_OPERATOR("+"|"-") | LOGICAL_OPERATOR("atau")	<additive-operator>	+, -, atau atau.	a + b
35	multiplicative_operator -> ARITHMETIC_OPERATOR("*"|"/"|"mod"|"bagi") | LOGICAL_OPERATOR("dan")	<multiplicative-operator>	*, /, mod, dan.	a * b

Bab III Pengujian

1. example.pas

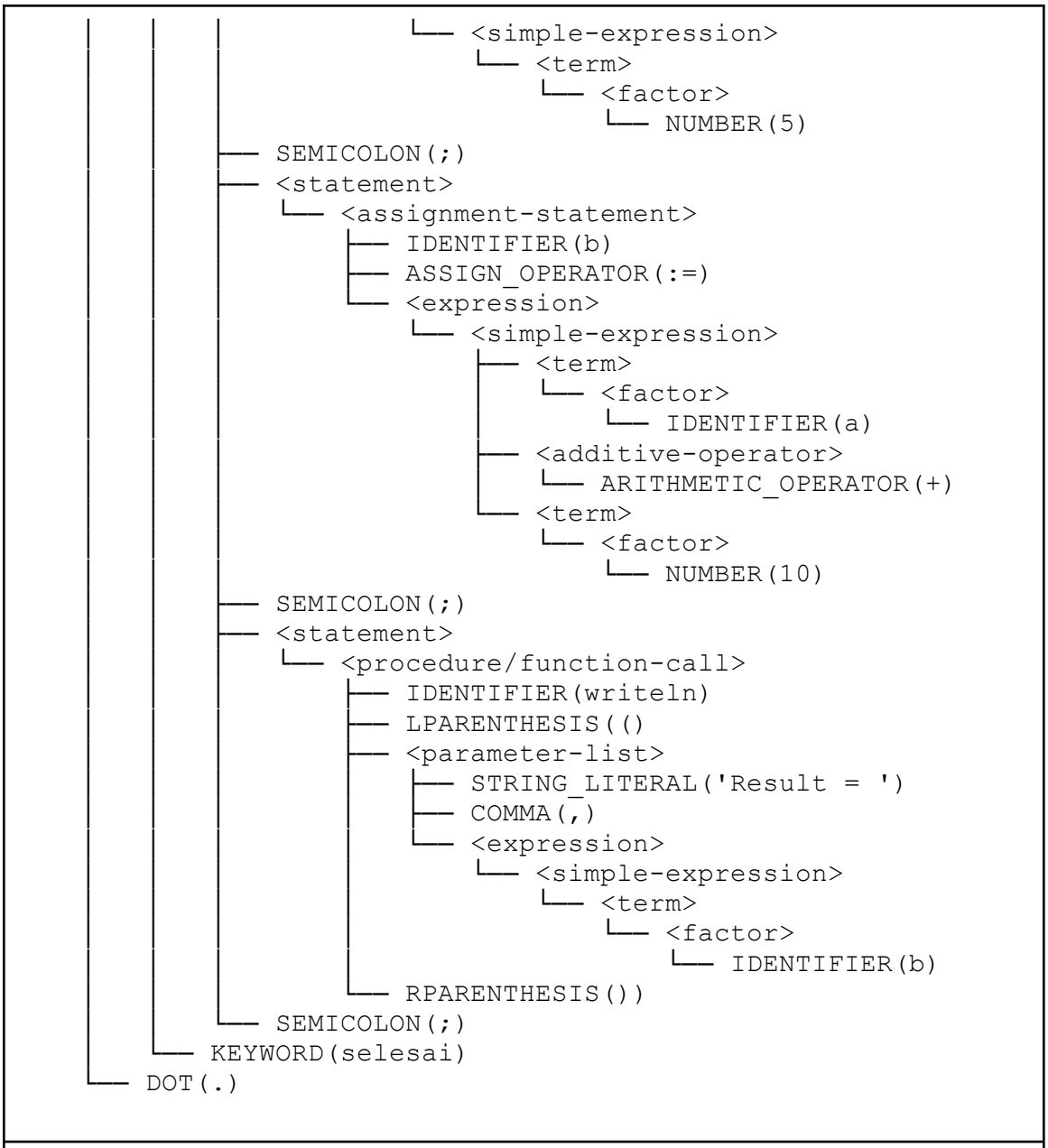
Kasus uji ini mencakup kasus yang dicantumkan pada dokumen *Spesifikasi Milestone 2* sebagai salah satu contoh hasil yang diharapkan.

Input

```
program Hello;

variabel
  a, b: integer;

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

Output

```
└── <program>
    ├── <program_header>
    │   ├── KEYWORD(program)
    │   ├── IDENTIFIER(Hello)
    │   └── SEMICOLON(;)
    ├── <declaration_part>
    │   └── <var_declaration>
    │       ├── KEYWORD(variabel)
    │       ├── <identifier_list>
    │       │   ├── IDENTIFIER(a)
    │       │   ├── COMMA(,)
    │       │   └── IDENTIFIER(b)
    │       ├── COLON(:)
    │       ├── <type>
    │       │   └── KEYWORD(integer)
    │       └── SEMICOLON(;)
    ├── <compound_statement>
    │   ├── KEYWORD(mulai)
    │   ├── <statement-list>
    │   │   ├── <statement>
    │   │   │   └── <assignment-statement>
    │   │   │       ├── IDENTIFIER(a)
    │   │   │       ├── ASSIGN_OPERATOR(:=)
    │   │   │       └── <expression>
```

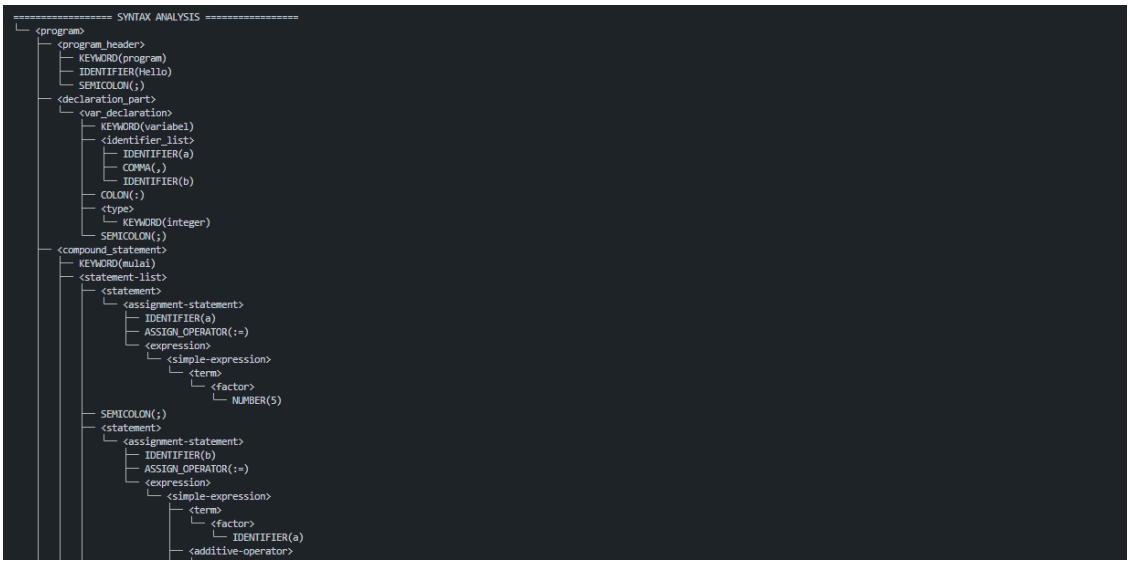


Bukti Input (Screenshot)

test > milestone-2 > example.pas

```
1   program Hello;
2
3   variabel
4     a, b: integer;
5
6   mulai
7     a := 5;
8     b := a + 10;
9     writeln('Result = ', b);
10  selesai.
11
```

Bukti Output (Screenshot)



```
================ SYNTAX ANALYSIS ================
└── <program>
    ├── <program_header>
    │   ├── KEYWORD(program)
    │   ├── IDENTIFIER(Hello)
    │   └── SEMICOLON(;)
    ├── <declaration_part>
    │   └── <var_declaration>
    │       ├── KEYWORD(variabel)
    │       ├── <identifier_list>
    │       │   ├── IDENTIFIER(a)
    │       │   ├── COMMA(,)
    │       │   └── IDENTIFIER(b)
    │       ├── COLON(:)
    │       ├── <type>
    │       │   └── KEYWORD(integer)
    │       └── SEMICOLON(;)
    ├── <compound_statement>
    │   ├── KEYWORD(mulai)
    │   ├── <statement-list>
    │   │   ├── <statement>
    │   │   │   └── <assignment-statement>
    │   │   │       ├── IDENTIFIER(a)
    │   │   │       ├── ASSIGN_OPERATOR(:=)
    │   │   │       └── <expression>
    │   │   │           └── <simple-expression>
    │   │   │               └── <term>
    │   │   │                   └── <factor>
    │   │   │                       └── NUMBER(5)
    │   │   ├── SEMICOLON(;)
    │   │   ├── <statement>
    │   │   │   └── <assignment-statement>
    │   │   │       ├── IDENTIFIER(b)
    │   │   │       ├── ASSIGN_OPERATOR(:=)
    │   │   │       └── <expression>
    │   │   │           └── <simple-expression>
    │   │   │               ├── <term>
    │   │   │               │   └── <factor>
    │   │   │               │       └── IDENTIFIER(a)
    │   │   │               ├── <additive-operator>
    │   │   │               │   └── ARITHMETIC_OPERATOR(+)
    │   │   │               └── <term>
    │   │   │                   └── <factor>
    │   │   │                       └── NUMBER(10)
    │   │   ├── SEMICOLON(;)
    │   │   ├── <statement>
    │   │   │   └── <procedure/function-call>
    │   │   │       ├── IDENTIFIER(writeln)
    │   │   │       ├── LPARENTHESIS(()
    │   │   │       ├── <parameter-list>
    │   │   │       │   ├── STRING_LITERAL('Result = ')
    │   │   │       │   ├── COMMA(,)
    │   │   │       │   └── <expression>
    │   │   │       │       └── <simple-expression>
    │   │   │       │           └── <term>
    │   │   │       │               └── <factor>
    │   │   │       │                   └── IDENTIFIER(b)
    │   │   │       └── RPARENTHESIS())
    │   │   └── SEMICOLON(;)
    │   └── KEYWORD(selesai)
    └── DOT(.)
PS D:\Repository\Tugas Kuliah\Teori Bahasa Formal Dan Otomata (TBFO)\SMN-Tubes-TF2224>
```

2. no_semicolon_error.pas

Kasus uji ini memperlihatkan kasus penanganan kegagalan parsing karena hilangnya titik koma (semicolon) di akhir statement line 8

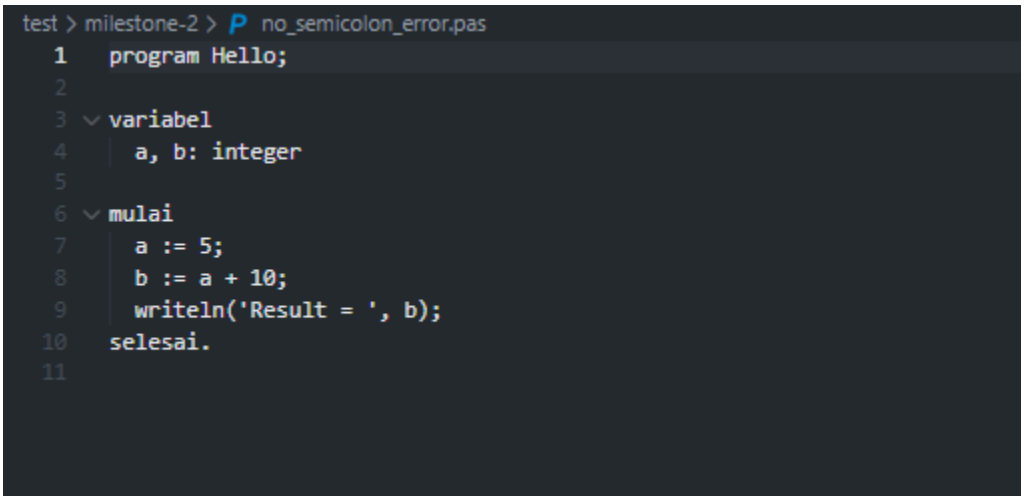
Input

```
program Hello;

variabel
  a, b: integer

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

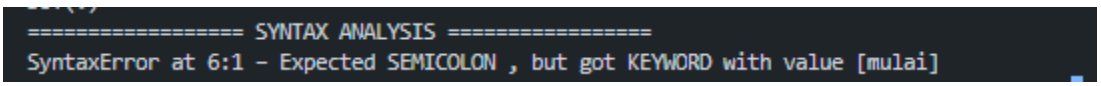
Output

```
SyntaxError at 6:1 - Expected SEMICOLON , but got KEYWORD with value [mulai]
```

Bukti Input (Screenshot)



Bukti Output (Screenshot)



3. no_dot_error.pas

Kasus uji ini memperlihatkan kasus penanganan kegagalan parsing karena hilangnya titik keyword dot (.) di akhir program

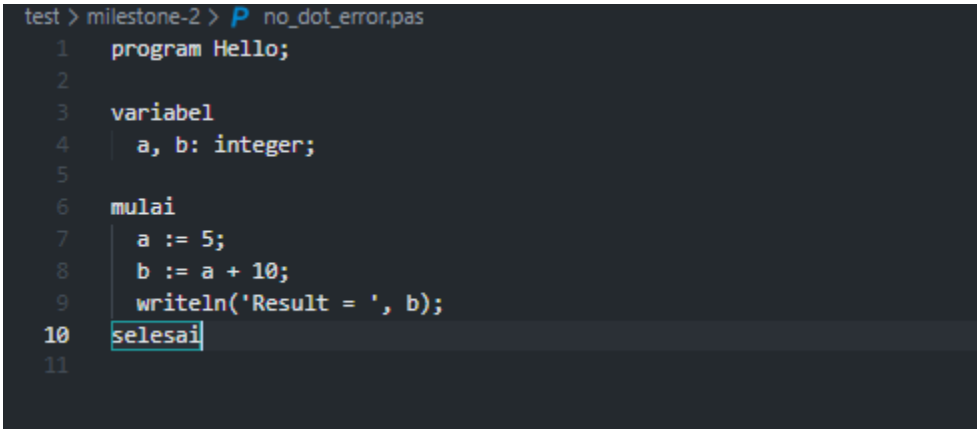
Input

```
program Hello;

variabel
  a, b: integer;

mulai
  a := 5;
  b := a + 10
  writeln('Result = ', b);
selesai
```

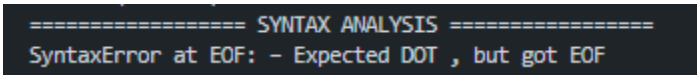
Output

```
SyntaxError at EOF: - Expected DOT , but got EOF
```

Bukti Input (Screenshot)



Bukti Output (Screenshot)



4. unknown_type_error.pas

Kasus uji ini memperlihatkan kasus penanganan kegagalan parsing karena terdapat tipe yang asing pada deklarasi variabel

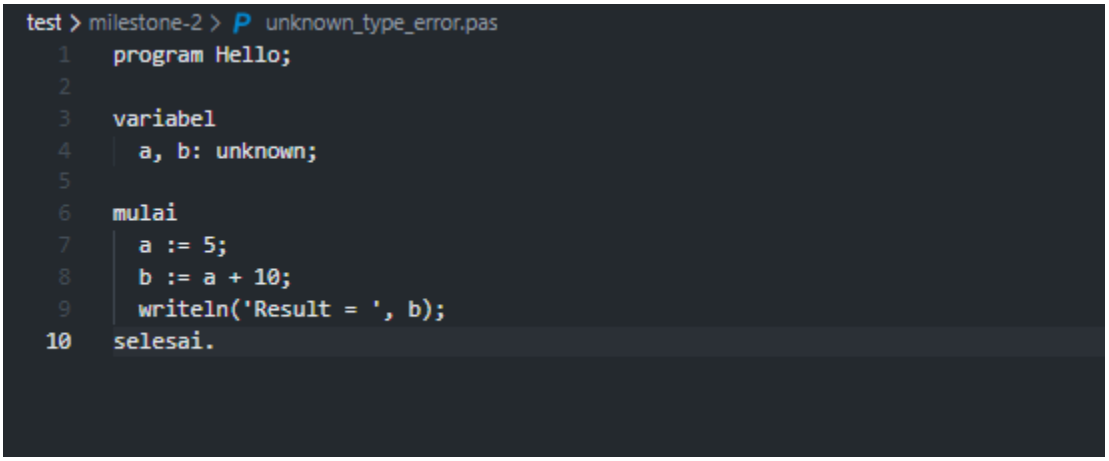
Input

```
program Hello;

variabel
  a, b: unknown;

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

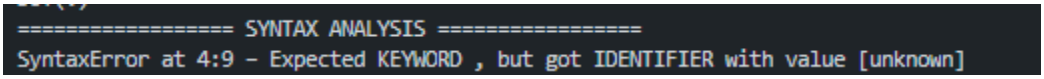
Output

```
SyntaxError at 4:9 - Expected KEYWORD , but got IDENTIFIER
with value [unknown]
```

Bukti Input (Screenshot)



Bukti Output (Screenshot)



5. all.pas

Kasus uji ini mencakup kasus yang memiliki setidaknya semua node yang mungkin untuk dibentuk

Input

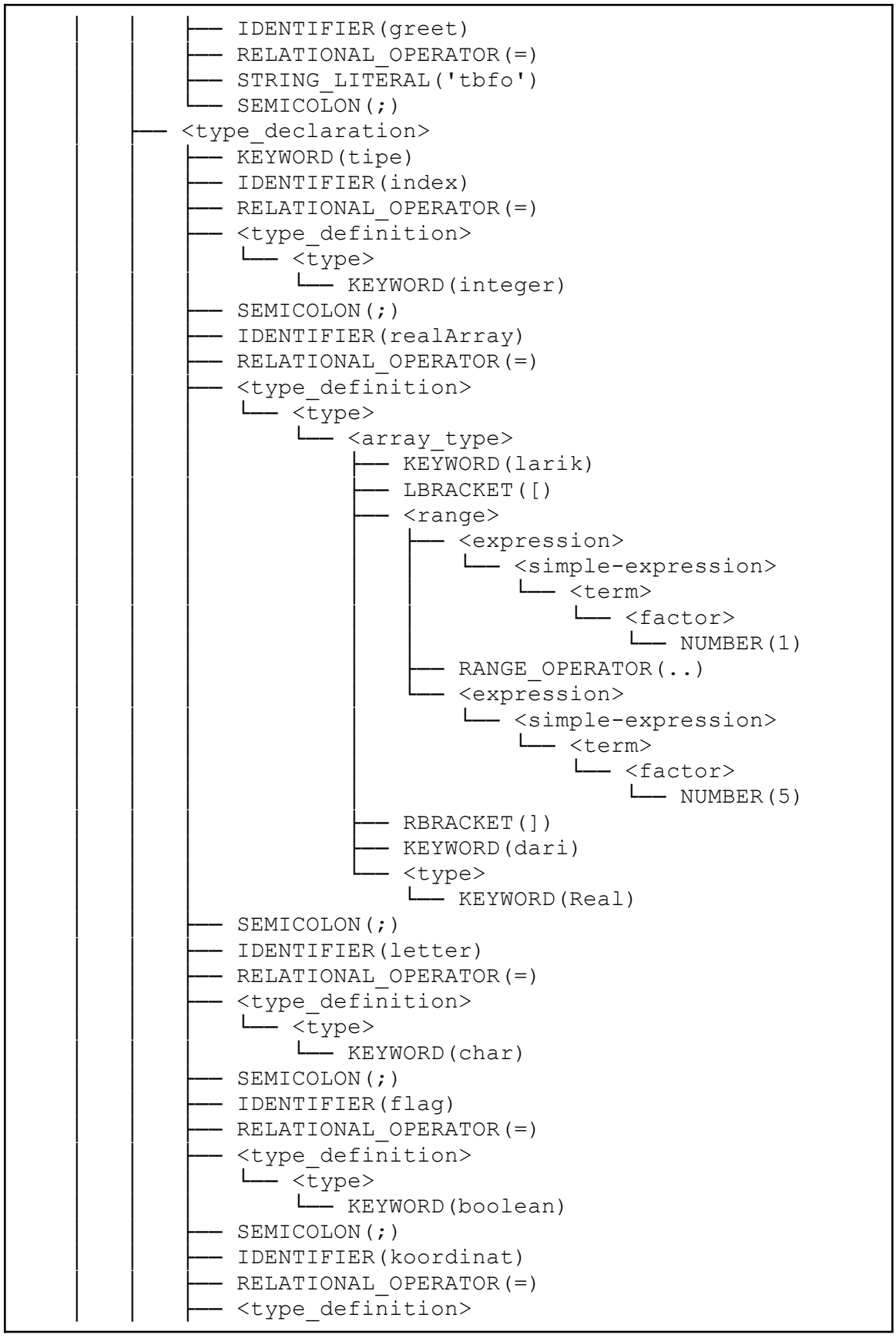
```
program test_full_tree;
```

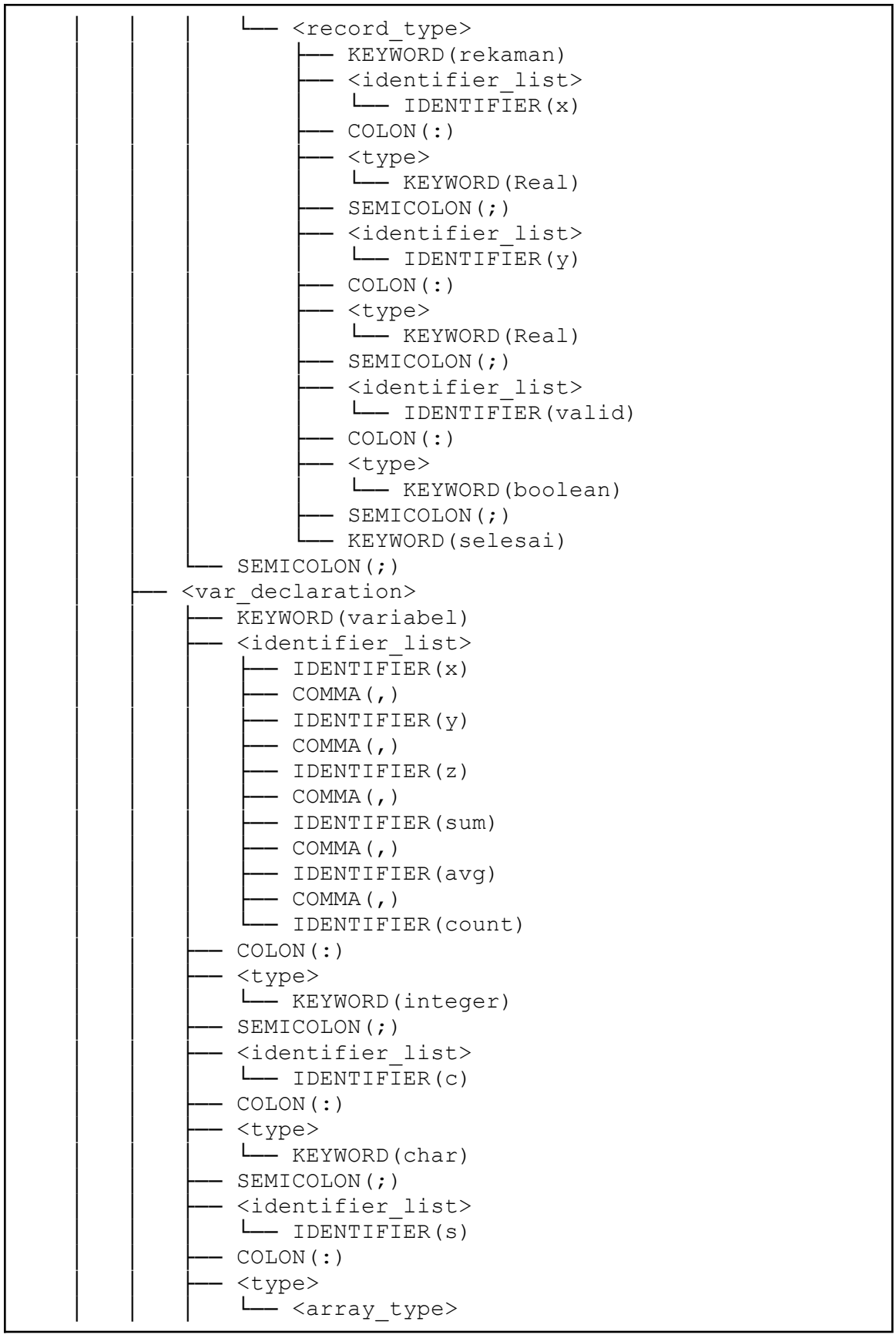
```
konstanta
  maxVal = 100;
  piVal  = 3.14;
  greet  = 'tbfo';

tipe
  index = integer;
  realArray = larik[1..5] dari Real;
  letter = char;
  flag = boolean;
  koordinat = rekaman
    x: Real;
    y: Real;
    valid: boolean;
  selesai;


variabel
  x, y, z, sum, avg, count: integer;
  c: char;
  s: larik[1..10] dari char;
  arr: integer;
  done: boolean;

prosedur ok(msg: char);
mulai
  jika tidak done maka
    writeln('Program ', msg, ' seru sekali')
  selain_itu
    writeln('Selesai');
selesai;

fungsi add(a, b: integer): integer;
mulai
  add := a + b;
selesai;

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

  done := false;

  jika (sum >= maxVal) dan tidak done maka
    done := true
  selain_itu jika (sum < maxVal) atau (x <> y) maka
    done := false;
```

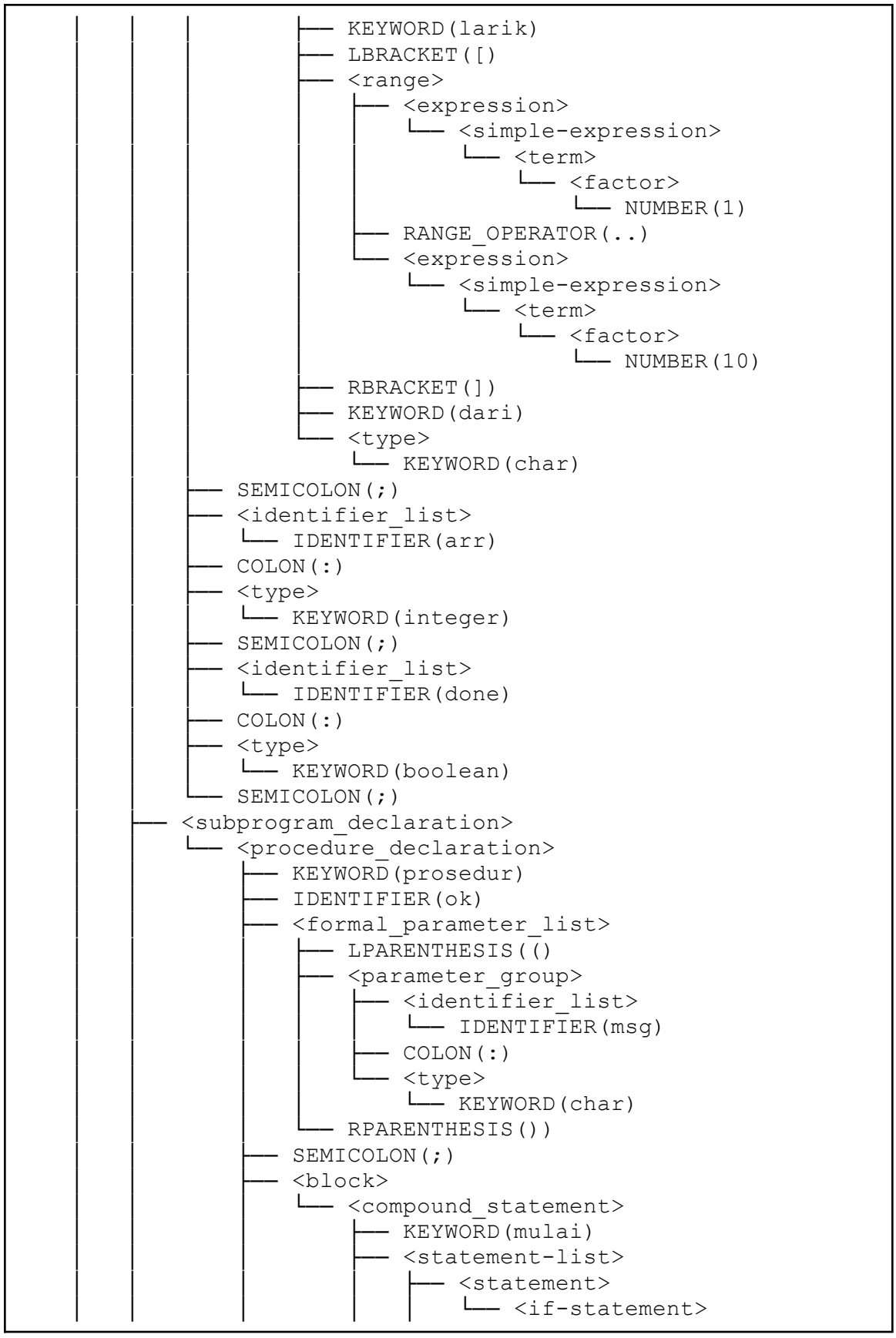
```
  jika (x <= y) maka
    done := false;

  selama (x > 0) lakukan
  mulai
    x := x - 1;
  selesai;

  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';

  untuk count := 1 ke 5 lakukan
    sum := sum + count;

  untuk count := 5 turun_ke 1 lakukan
    avg := avg + arr[count];

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

selesai.
```

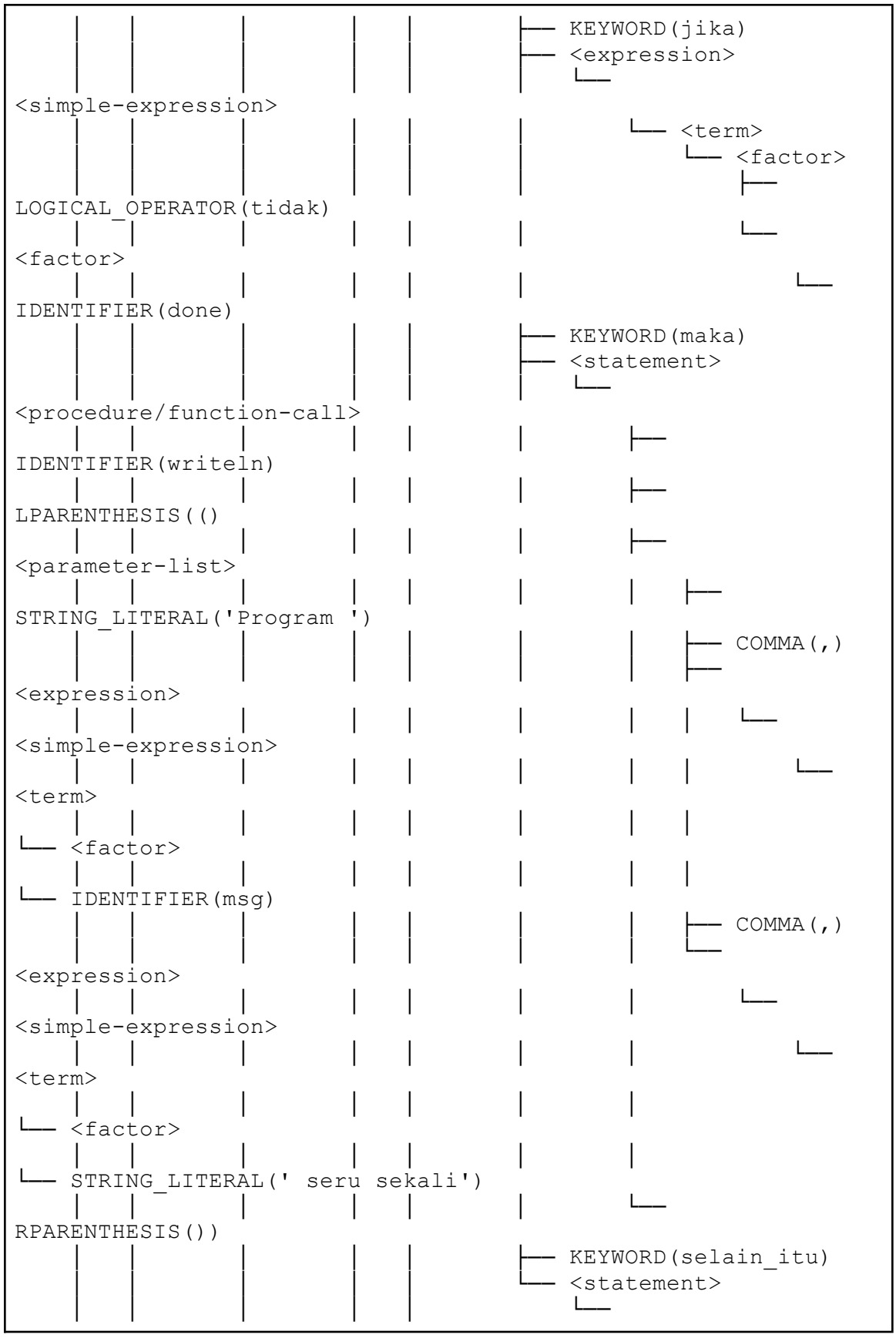
Output

```
└── <program>
    ├── <program_header>
    │   ├── KEYWORD(program)
    │   ├── IDENTIFIER(test_full_tree)
    │   └── SEMICOLON(;)
    ├── <declaration_part>
    │   ├── <const_declaration>
    │   │   ├── KEYWORD(konstanta)
    │   │   ├── IDENTIFIER(maxVal)
    │   │   ├── RELATIONAL_OPERATOR(=)
    │   │   ├── NUMBER(100)
    │   │   ├── SEMICOLON(;)
    │   │   ├── IDENTIFIER(piVal)
    │   │   ├── RELATIONAL_OPERATOR(=)
    │   │   ├── NUMBER(3.14)
    │   │   ├── SEMICOLON(;)
```

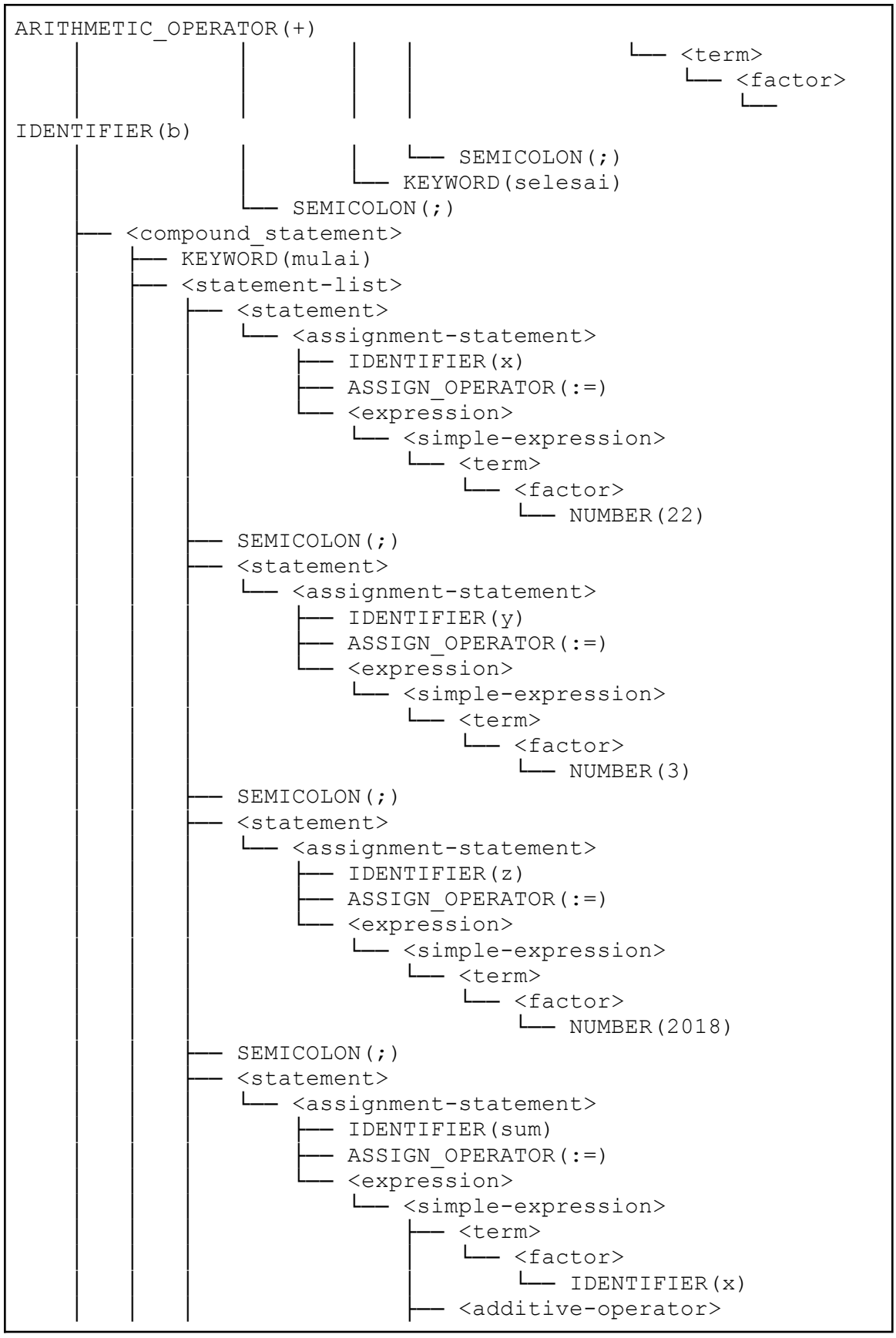

IDENTIFIER(greet)
RELATIONAL_OPERATOR(=)
STRING_LITERAL('tbfo')
SEMICOLON(;)
<type_declaration>
KEYWORD(tipe)
IDENTIFIER(index)
RELATIONAL_OPERATOR(=)
<type_definition>
<type>
KEYWORD(integer)
SEMICOLON(;)
IDENTIFIER(realArray)
RELATIONAL_OPERATOR(=)
<type_definition>
<type>
<array_type>
KEYWORD(larik)
LBRACKET([)
<range>
<expression>
<simple-expression>
<term>
<factor>
NUMBER(1)
RANGE_OPERATOR(..)
<expression>
<simple-expression>
<term>
<factor>
NUMBER(5)
RBRACKET(])
KEYWORD(dari)
<type>
KEYWORD(Real)
SEMICOLON(;)
IDENTIFIER(letter)
RELATIONAL_OPERATOR(=)
<type_definition>
<type>
KEYWORD(char)
SEMICOLON(;)
IDENTIFIER(flag)
RELATIONAL_OPERATOR(=)
<type_definition>
<type>
KEYWORD(boolean)
SEMICOLON(;)
IDENTIFIER(koordinat)
RELATIONAL_OPERATOR(=)
<type_definition>

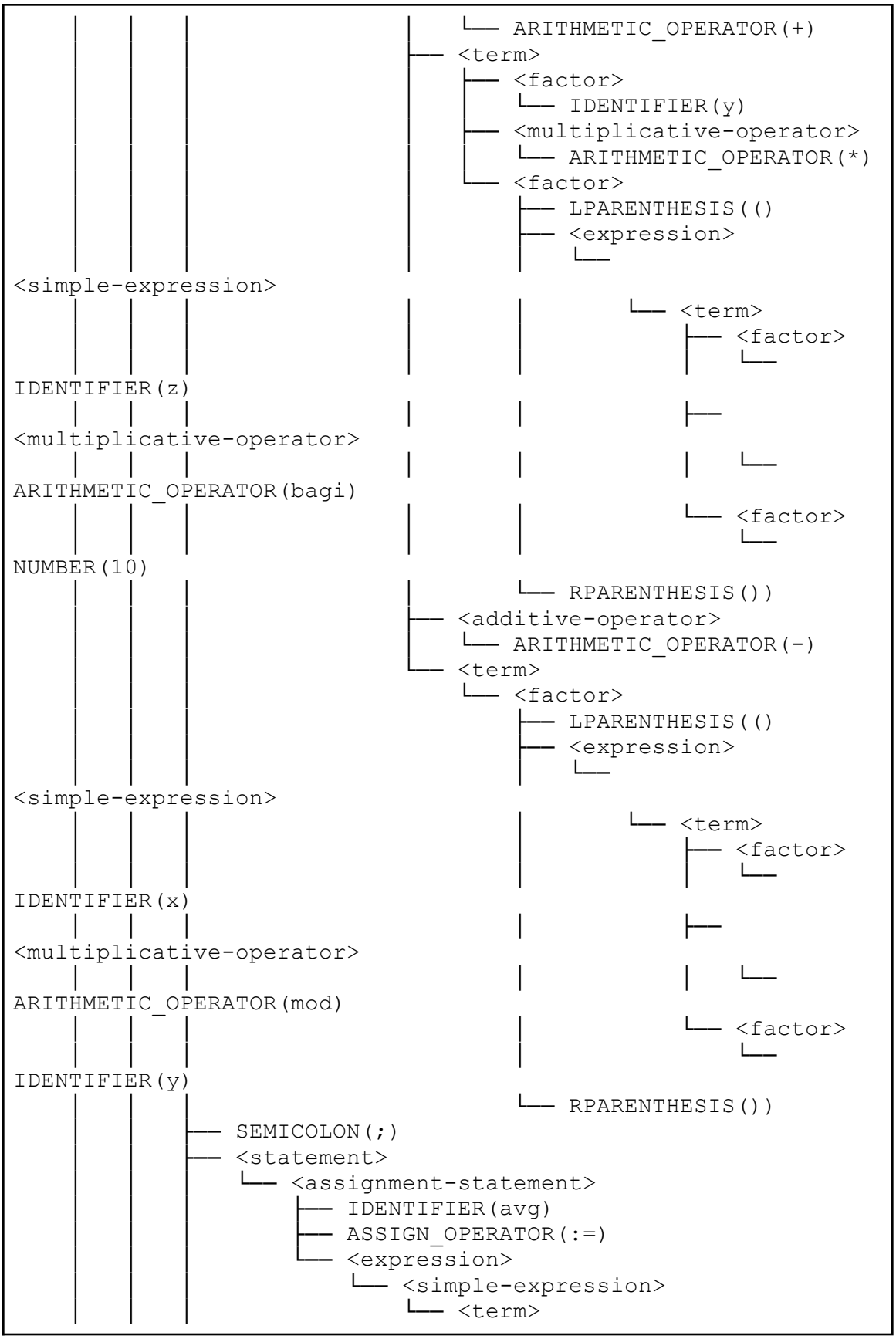

└── <record_type>
├── KEYWORD(rekaman)
├── <identifier_list>
└── IDENTIFIER(x)
├── COLON(:)
├── <type>
└── KEYWORD(Real)
├── SEMICOLON(;)
├── <identifier_list>
└── IDENTIFIER(y)
├── COLON(:)
├── <type>
└── KEYWORD(Real)
├── SEMICOLON(;)
├── <identifier_list>
└── IDENTIFIER(valid)
├── COLON(:)
├── <type>
└── KEYWORD(boolean)
├── SEMICOLON(;)
└── KEYWORD(selesai)
└── SEMICOLON(;)
├── <var_declaration>
├── KEYWORD(variabel)
├── <identifier_list>
├── IDENTIFIER(x)
├── COMMA(,)
├── IDENTIFIER(y)
├── COMMA(,)
├── IDENTIFIER(z)
├── COMMA(,)
├── IDENTIFIER(sum)
├── COMMA(,)
├── IDENTIFIER(avg)
├── COMMA(,)
└── IDENTIFIER(count)
├── COLON(:)
├── <type>
└── KEYWORD(integer)
├── SEMICOLON(;)
├── <identifier_list>
└── IDENTIFIER(c)
├── COLON(:)
├── <type>
└── KEYWORD(char)
├── SEMICOLON(;)
├── <identifier_list>
└── IDENTIFIER(s)
├── COLON(:)
├── <type>
└── <array_type>

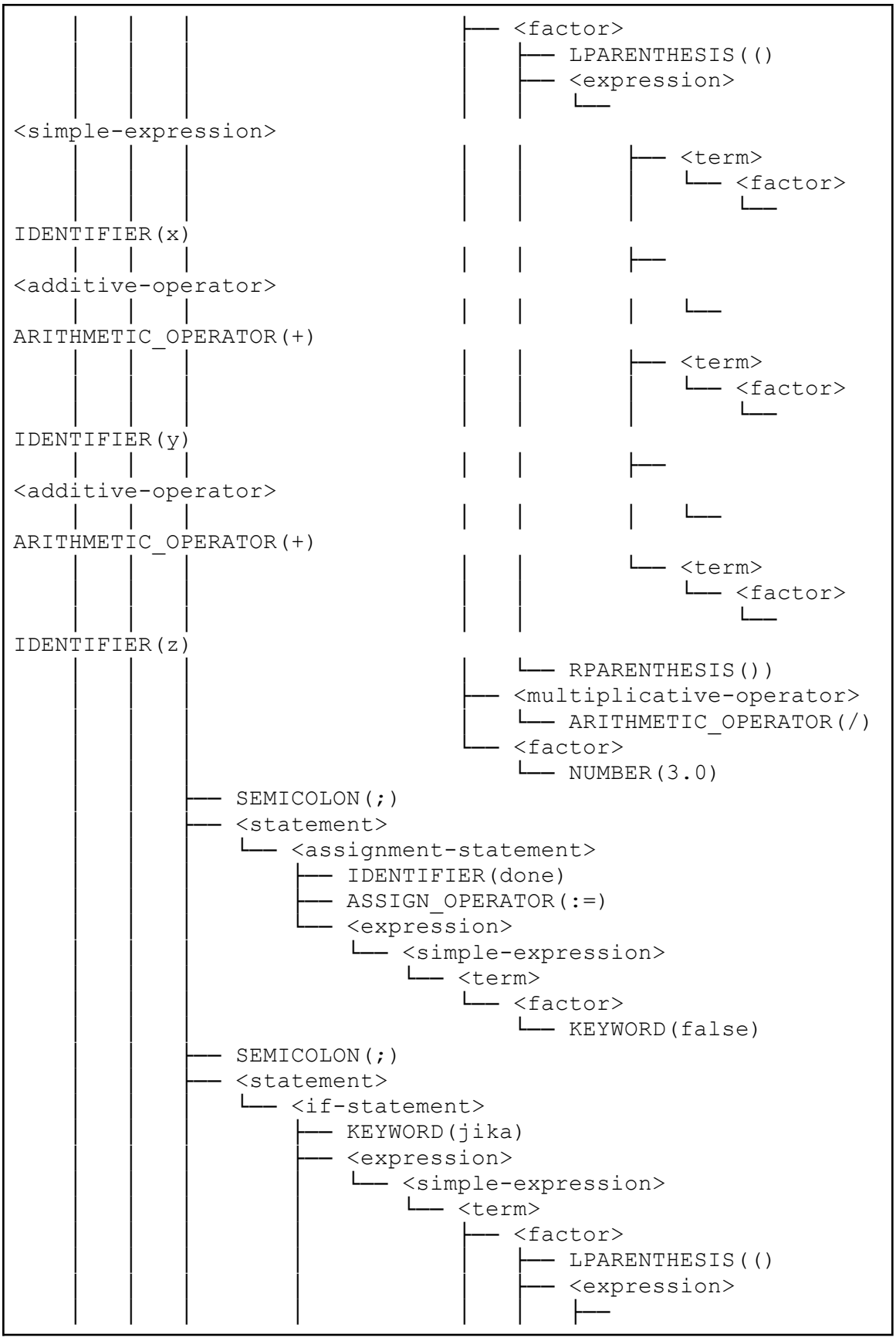

KEYWORD(larik)
LBRACKET([)
<range>
<expression>
<simple-expression>
<term>
<factor>
NUMBER(1)
RANGE_OPERATOR(..)
<expression>
<simple-expression>
<term>
<factor>
NUMBER(10)
RBRACKET(])
KEYWORD(dari)
<type>
KEYWORD(char)
SEMICOLON(;)
<identifier_list>
IDENTIFIER(arr)
COLON(:)
<type>
KEYWORD(integer)
SEMICOLON(;)
<identifier_list>
IDENTIFIER(done)
COLON(:)
<type>
KEYWORD(boolean)
SEMICOLON(;)
<subprogram_declaration>
<procedure_declaration>
KEYWORD(prosedur)
IDENTIFIER(ok)
<formal_parameter_list>
LPARENTHESIS(()
<parameter_group>
<identifier_list>
IDENTIFIER(msg)
COLON(:)
<type>
KEYWORD(char)
RPARENTHESIS())
SEMICOLON(;)
<block>
<compound_statement>
KEYWORD(mulai)
<statement-list>
<statement>
<if-statement>

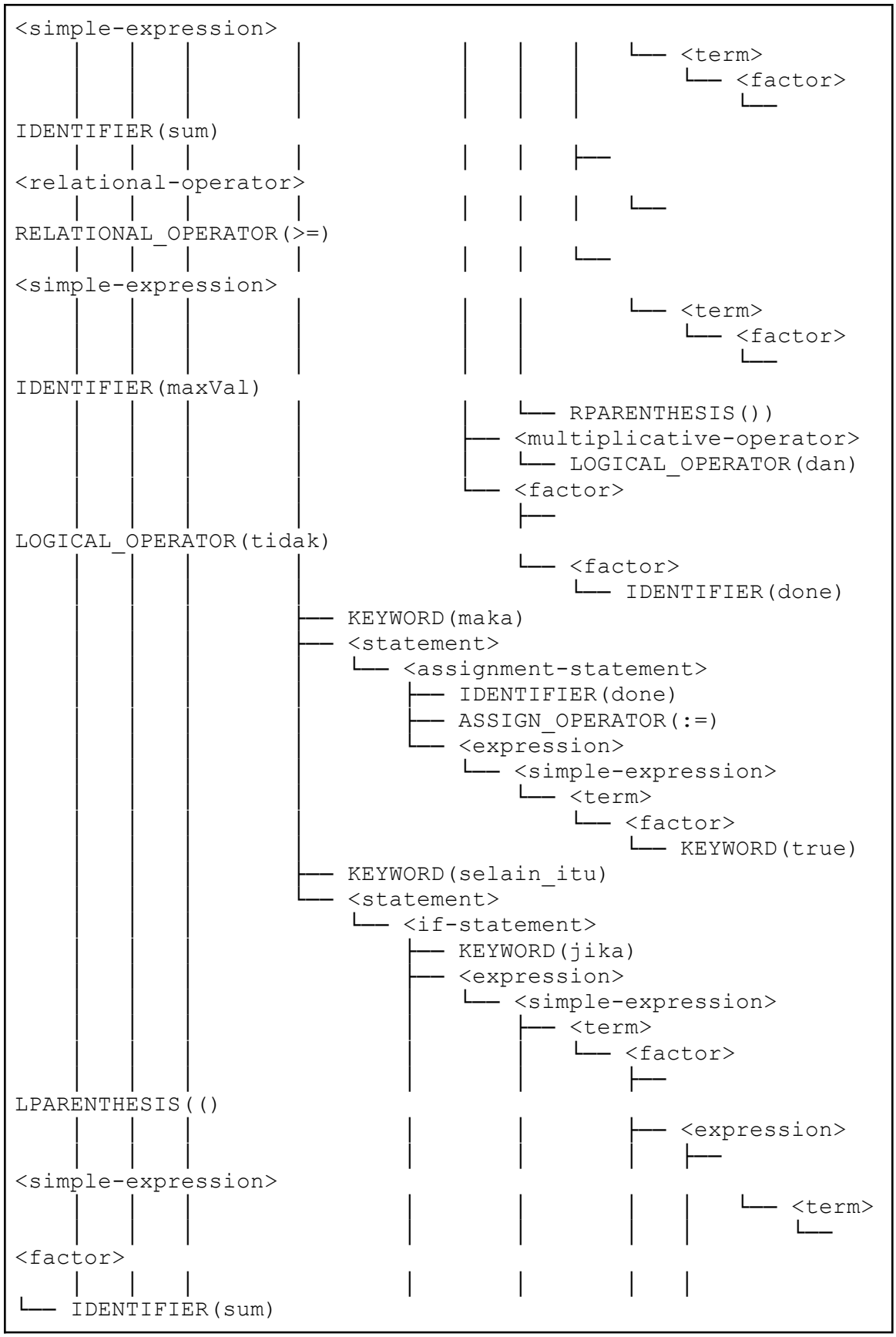

KEYWORD(jika)
<expression>
<simple-expression>
<term>
<factor>
LOGICAL_OPERATOR(tidak)
<factor>
IDENTIFIER(done)
KEYWORD(maka)
<statement>
<procedure/function-call>
IDENTIFIER(writeln)
LPARENTHESIS(()
<parameter-list>
STRING_LITERAL('Program ')
COMMA(,)
<expression>
<simple-expression>
<term>
<factor>
IDENTIFIER(msg)
COMMA(,)
<expression>
<simple-expression>
<term>
<factor>
STRING_LITERAL(' seru sekali')
RPARENTHESIS())
KEYWORD(selain_itu)
<statement>

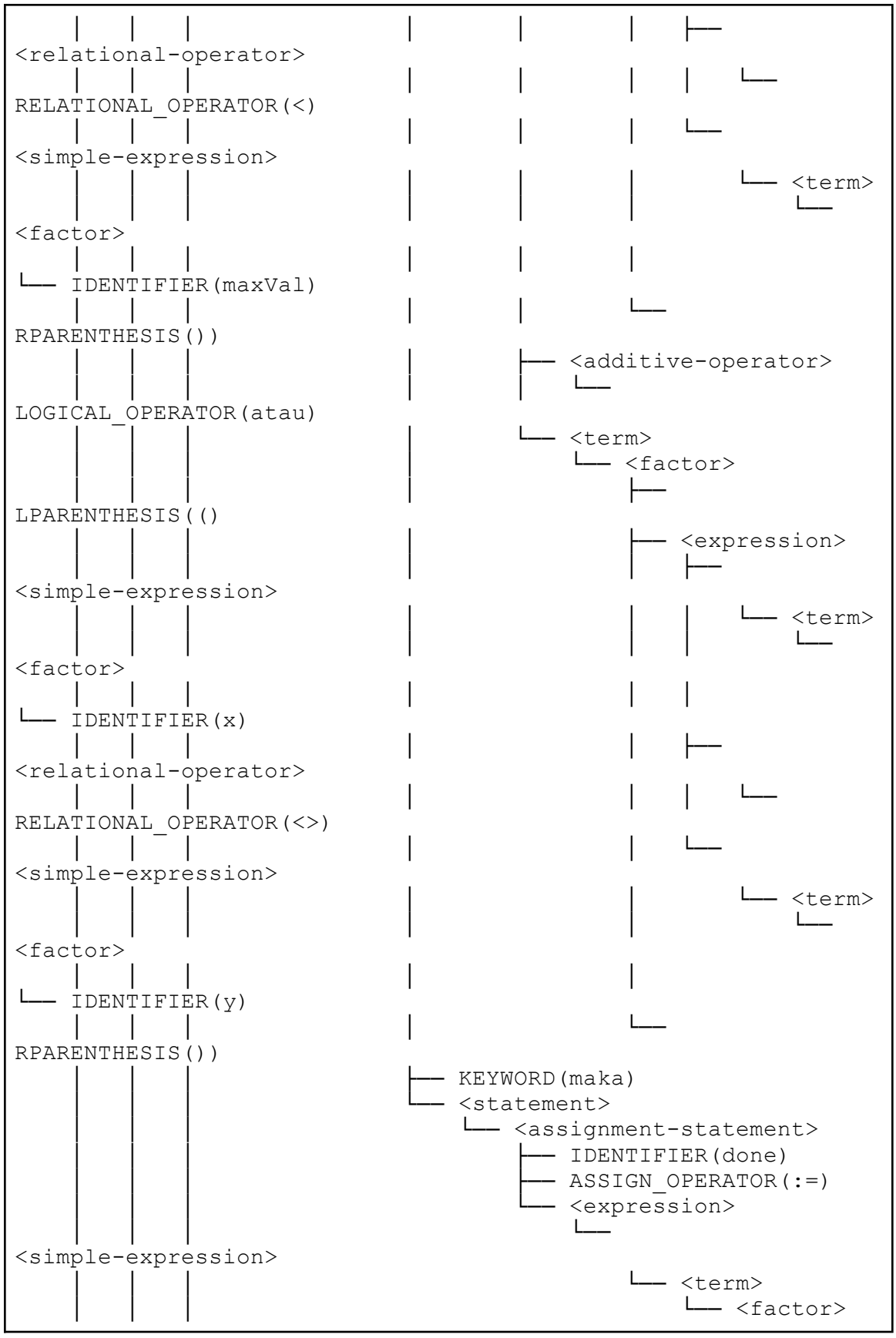

<procedure/function-call>
IDENTIFIER(writeln)
LPARENTHESIS(()
<parameter-list>
STRING_LITERAL('Selesai')
RPARENTHESIS())
SEMICOLON(;)
KEYWORD(selesai)
SEMICOLON(;)
<subprogram_declaration>
<function_declaration>
KEYWORD(fungsi)
IDENTIFIER(add)
<formal_parameter_list>
LPARENTHESIS(()
<parameter_group>
<identifier_list>
IDENTIFIER(a)
COMMA(,)
IDENTIFIER(b)
COLON(:)
<type>
KEYWORD(integer)
RPARENTHESIS())
COLON(:)
<type>
KEYWORD(integer)
SEMICOLON(;)
<block>
<compound_statement>
KEYWORD(mulai)
<statement-list>
<statement>
<assignment-statement>
IDENTIFIER(add)
ASSIGN_OPERATOR(:=)
<expression>
<simple-expression>
<term>
<factor>
IDENTIFIER(a)
<additive-operator>

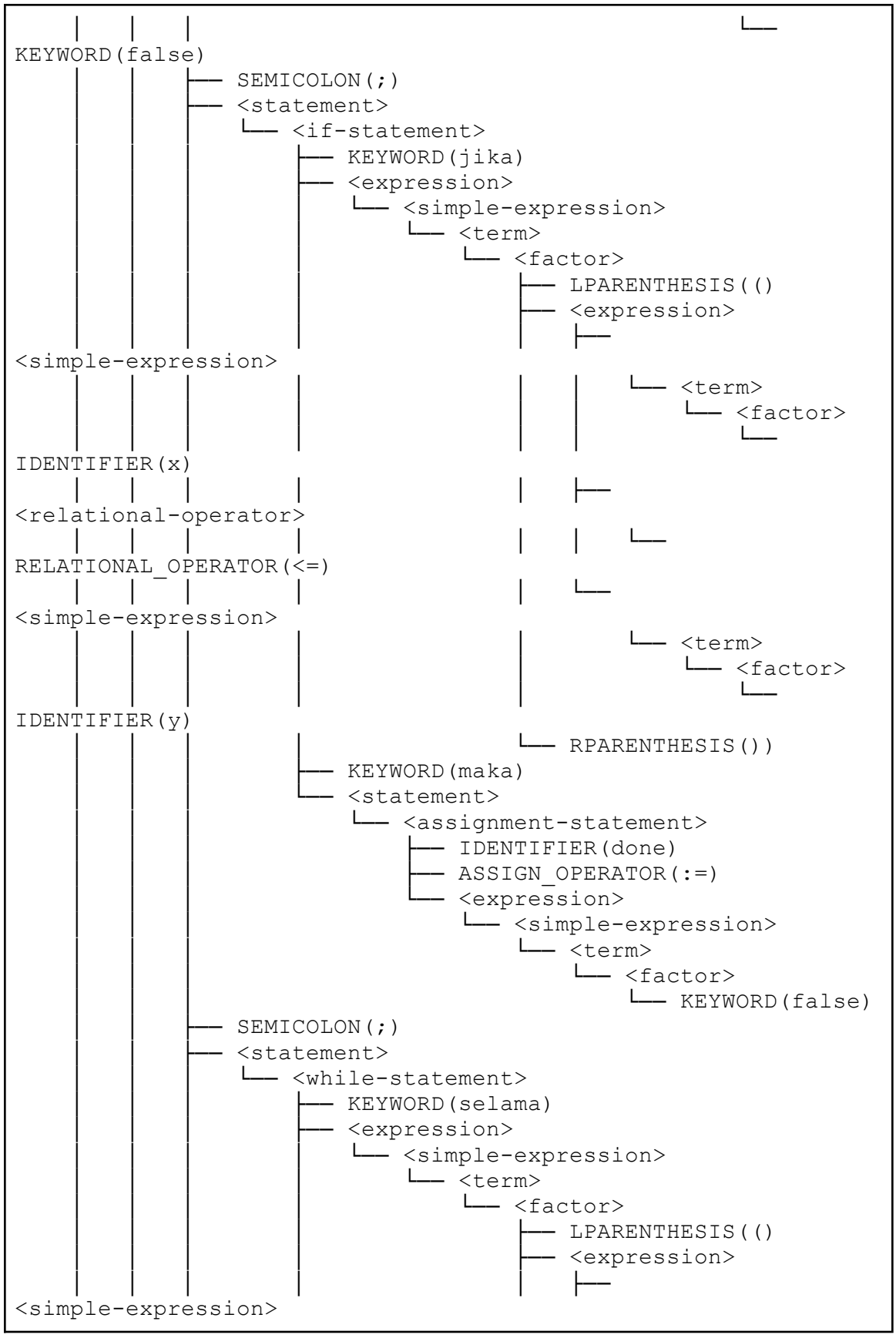

ARITHMETIC_OPERATOR(+)
<term>
<factor>
IDENTIFIER(b)
SEMICOLON(;)
KEYWORD(selesai)
SEMICOLON(;)
<compound_statement>
KEYWORD(mulai)
<statement-list>
<statement>
<assignment-statement>
IDENTIFIER(x)
ASSIGN_OPERATOR(:=)
<expression>
<simple-expression>
<term>
<factor>
NUMBER(22)
SEMICOLON(;)
<statement>
<assignment-statement>
IDENTIFIER(y)
ASSIGN_OPERATOR(:=)
<expression>
<simple-expression>
<term>
<factor>
NUMBER(3)
SEMICOLON(;)
<statement>
<assignment-statement>
IDENTIFIER(z)
ASSIGN_OPERATOR(:=)
<expression>
<simple-expression>
<term>
<factor>
NUMBER(2018)
SEMICOLON(;)
<statement>
<assignment-statement>
IDENTIFIER(sum)
ASSIGN_OPERATOR(:=)
<expression>
<simple-expression>
<term>
<factor>
IDENTIFIER(x)
<additive-operator>

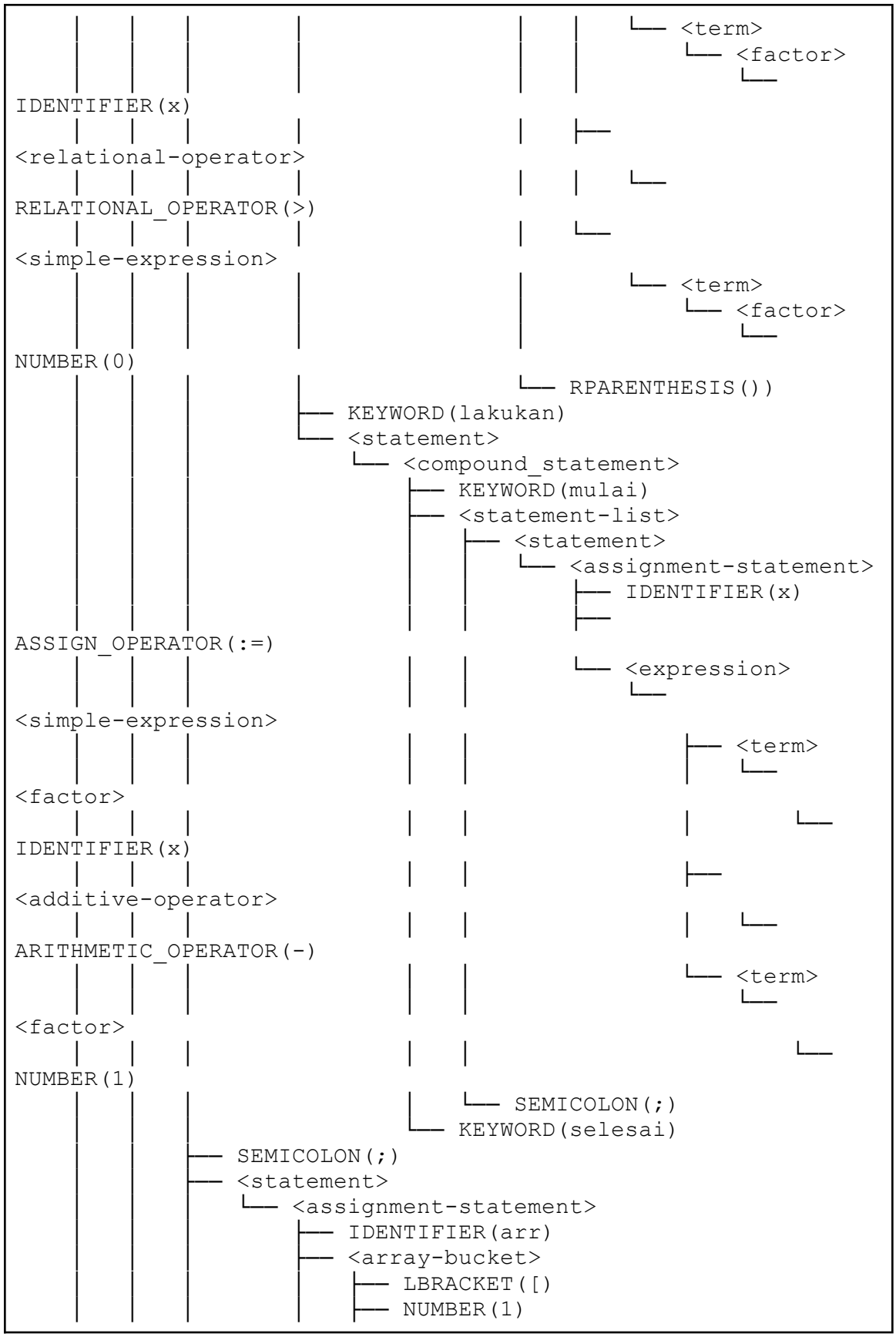

└── ARITHMETIC_OPERATOR(+)
├── <term>
├── <factor>
└── IDENTIFIER(y)
├── <multiplicative-operator>
└── ARITHMETIC_OPERATOR(*)
└── <factor>
├── LPARENTHESIS(()
├── <expression>
└──
<simple-expression>
└── <term>
├── <factor>
└──
IDENTIFIER(z)
├──
<multiplicative-operator>
└──
ARITHMETIC_OPERATOR(bagi)
└── <factor>
└──
NUMBER(10)
└── RPARENTHESIS())
├── <additive-operator>
└── ARITHMETIC_OPERATOR(-)
└── <term>
└── <factor>
├── LPARENTHESIS(()
├── <expression>
└──
<simple-expression>
└── <term>
├── <factor>
└──
IDENTIFIER(x)
├──
<multiplicative-operator>
└──
ARITHMETIC_OPERATOR(mod)
└── <factor>
└──
IDENTIFIER(y)
└── RPARENTHESIS())
├── SEMICOLON(;)
├── <statement>
└── <assignment-statement>
├── IDENTIFIER(avg)
├── ASSIGN_OPERATOR(:=)
└── <expression>
└── <simple-expression>
└── <term>

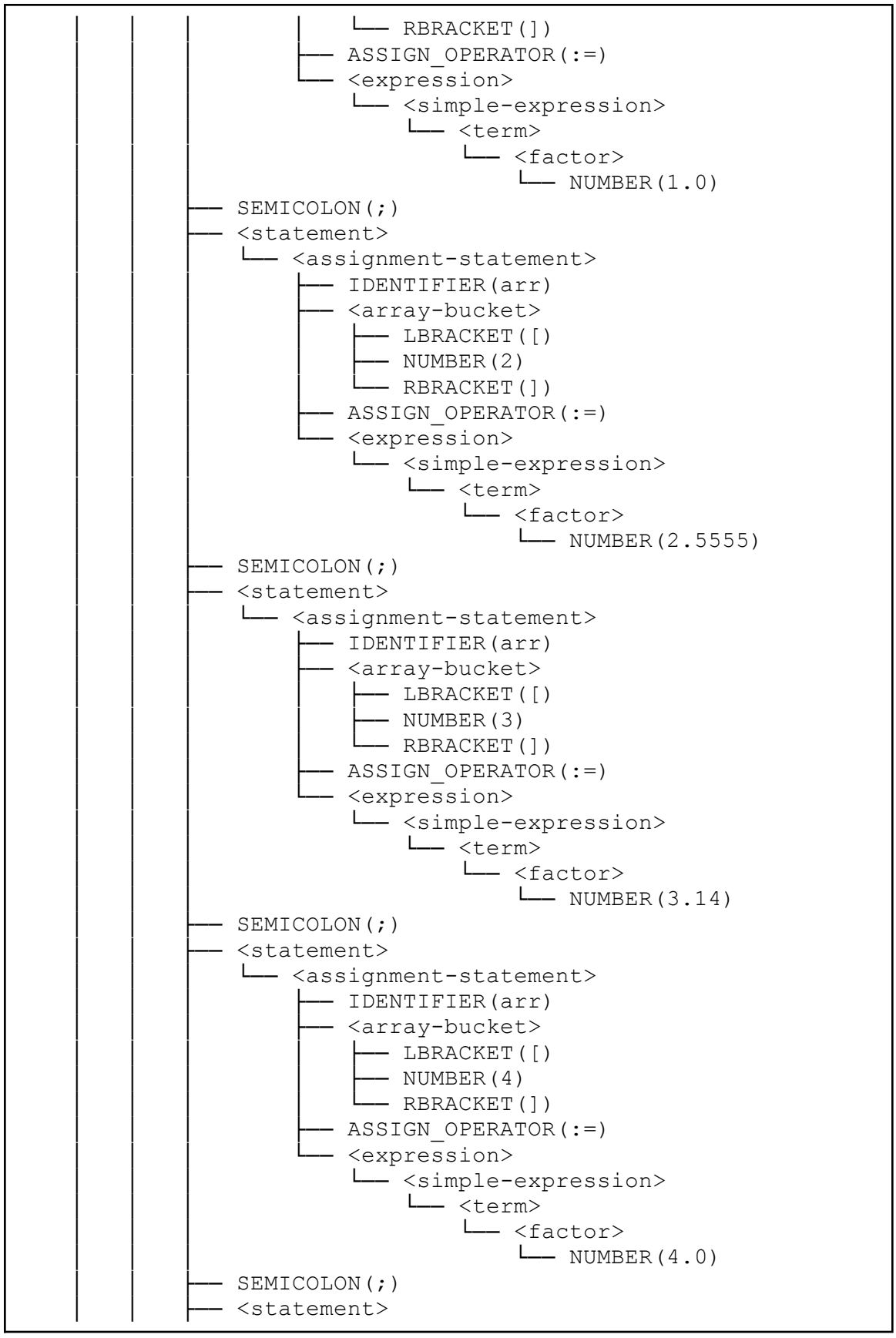

├── <factor>
│ ├── LPARENTHESIS(()
│ ├── <expression>
│ │ └──
<simple-expression>
│ │ ├── <term>
│ │ │ └── <factor>
│ │ │ └──
IDENTIFIER(x)
│ │ ├──
<additive-operator>
│ │ │ └──
ARITHMETIC_OPERATOR(+)
│ │ ├── <term>
│ │ │ └── <factor>
│ │ │ └──
IDENTIFIER(y)
│ │ ├──
<additive-operator>
│ │ │ └──
ARITHMETIC_OPERATOR(+)
│ │ └── <term>
│ │ └── <factor>
│ │ └──
IDENTIFIER(z)
│ └── RPARENTHESIS())
├── <multiplicative-operator>
│ └── ARITHMETIC_OPERATOR(/)
└── <factor>
└── NUMBER(3.0)
├── SEMICOLON(;)
├── <statement>
└── <assignment-statement>
├── IDENTIFIER(done)
├── ASSIGN_OPERATOR(:=)
└── <expression>
└── <simple-expression>
└── <term>
└── <factor>
└── KEYWORD(false)
├── SEMICOLON(;)
├── <statement>
└── <if-statement>
├── KEYWORD(jika)
├── <expression>
└── <simple-expression>
└── <term>
├── <factor>
│ ├── LPARENTHESIS(()
│ ├── <expression>
│ ├──


<simple-expression>
IDENTIFIER(sum)
<relational-operator>
RELATIONAL_OPERATOR(>=)
<simple-expression>
IDENTIFIER(maxVal)
LOGICAL_OPERATOR(tidak)
KEYWORD(maka)
<statement>
<assignment-statement>
IDENTIFIER(done)
ASSIGN_OPERATOR(:=)
<expression>
<simple-expression>
<term>
<factor>
KEYWORD(true)
KEYWORD(selain_itu)
<statement>
<if-statement>
KEYWORD(jika)
<expression>
<simple-expression>
<term>
<factor>
LPARENTHESIS(()
<expression>
<simple-expression>
<term>
<factor>
IDENTIFIER(sum)
RPARENTHESIS())
<multiplicative-operator>
LOGICAL_OPERATOR(dan)
<factor>
<factor>
IDENTIFIER(done)
<term>
<factor>

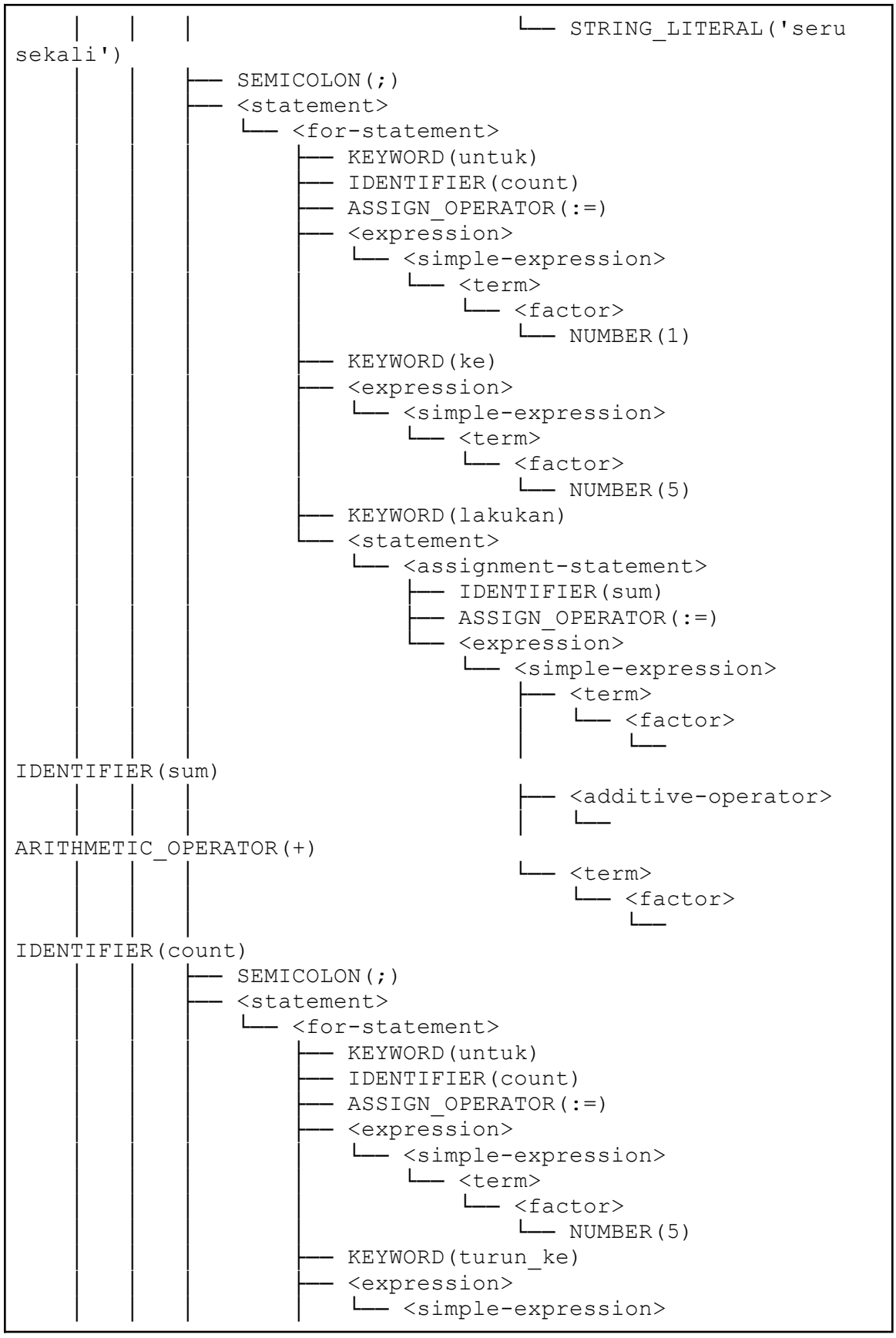

├──
<relational-operator>
└──
RELATIONAL_OPERATOR(<)
└──
<simple-expression>
└── <term>
└──
<factor>
└── IDENTIFIER(maxVal)
└──
RPARENTHESIS())
├── <additive-operator>
└──
LOGICAL_OPERATOR(atau)
└── <term>
└── <factor>
├──
LPARENTHESIS(()
├── <expression>
├──
<simple-expression>
└── <term>
└──
<factor>
└── IDENTIFIER(x)
├──
<relational-operator>
└──
RELATIONAL_OPERATOR(<>)
└──
<simple-expression>
└── <term>
└──
<factor>
└── IDENTIFIER(y)
└──
RPARENTHESIS())
├── KEYWORD(maka)
└── <statement>
└── <assignment-statement>
├── IDENTIFIER(done)
├── ASSIGN_OPERATOR(:=)
└── <expression>
└──
<simple-expression>
└── <term>
└── <factor>

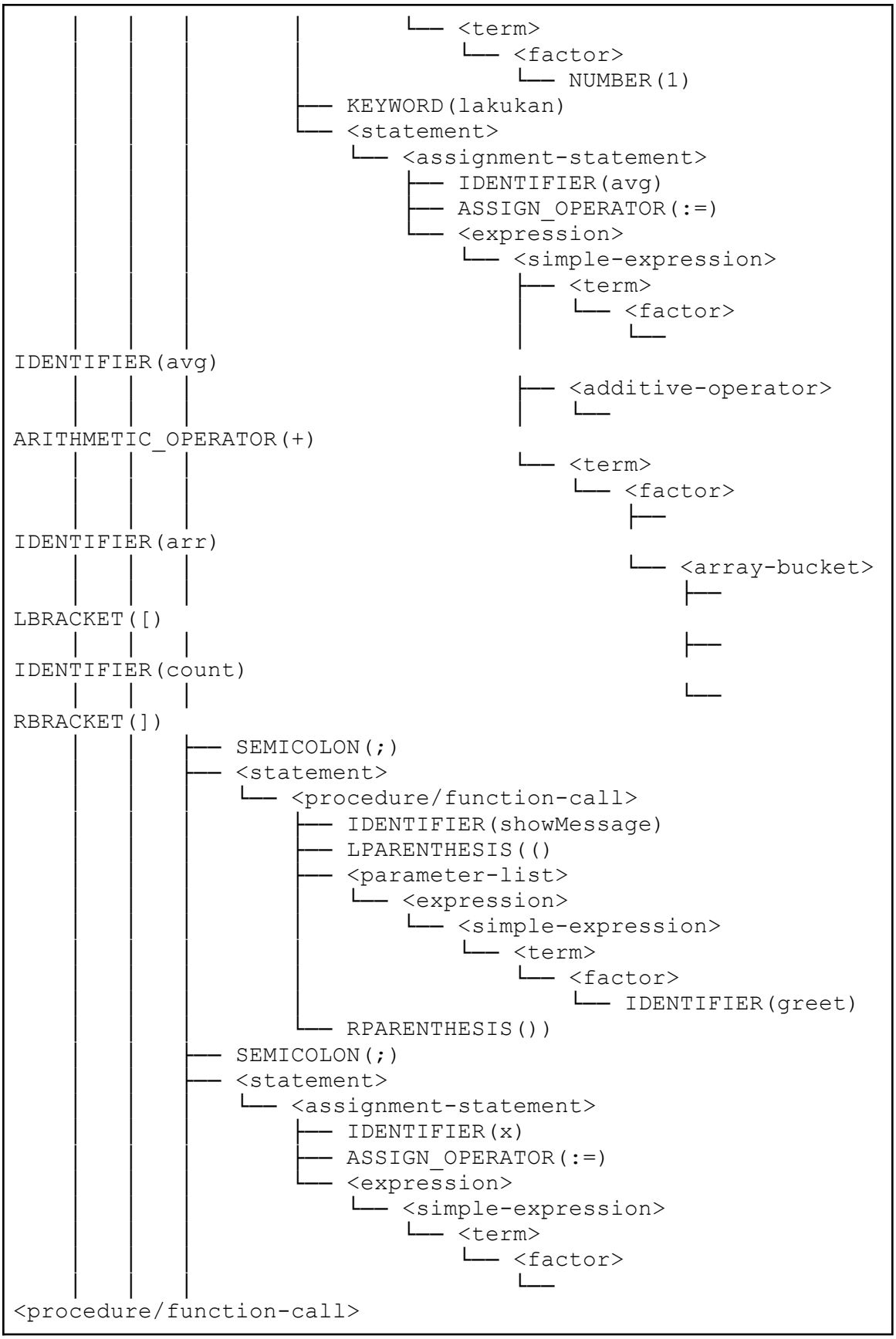

└──
KEYWORD(false)
├── SEMICOLON(;)
├── <statement>
└── <if-statement>
├── KEYWORD(jika)
├── <expression>
└── <simple-expression>
└── <term>
└── <factor>
├── LPARENTHESIS(()
├── <expression>
├──
<simple-expression>
└── <term>
└── <factor>
└──
IDENTIFIER(x)
├──
<relational-operator>
└──
RELATIONAL_OPERATOR(<=)
└──
<simple-expression>
└── <term>
└── <factor>
└──
IDENTIFIER(y)
└── RPARENTHESIS())
├── KEYWORD(maka)
└── <statement>
└── <assignment-statement>
├── IDENTIFIER(done)
├── ASSIGN_OPERATOR(:=)
└── <expression>
└── <simple-expression>
└── <term>
└── <factor>
└── KEYWORD(false)
├── SEMICOLON(;)
├── <statement>
└── <while-statement>
├── KEYWORD(selama)
├── <expression>
└── <simple-expression>
└── <term>
└── <factor>
├── LPARENTHESIS(()
├── <expression>
├──
<simple-expression>

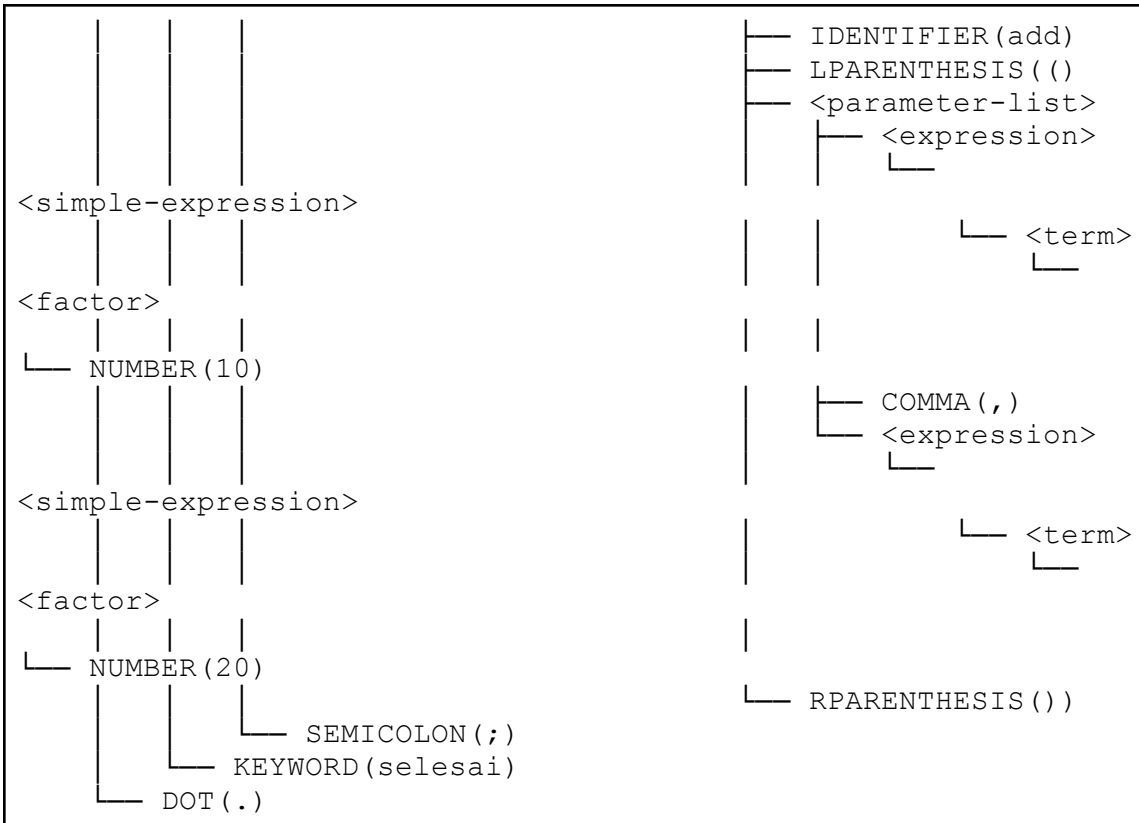

└── <term>
└── <factor>
└──
IDENTIFIER(x)
├──
<relational-operator>
└──
RELATIONAL_OPERATOR(>)
└──
<simple-expression>
└── <term>
└── <factor>
└──
NUMBER(0)
└── RPARENTHESIS())
├── KEYWORD(lakukan)
└── <statement>
└── <compound_statement>
├── KEYWORD(mulai)
├── <statement-list>
├── <statement>
└── <assignment-statement>
├── IDENTIFIER(x)
├──
ASSIGN_OPERATOR(:=)
└── <expression>
└──
<simple-expression>
├── <term>
└──
<factor>
└──
IDENTIFIER(x)
├──
<additive-operator>
└──
ARITHMETIC_OPERATOR(-)
└── <term>
└──
<factor>
└──
NUMBER(1)
└── SEMICOLON(;)
└── KEYWORD(selesai)
├── SEMICOLON(;)
├── <statement>
└── <assignment-statement>
├── IDENTIFIER(arr)
├── <array-bucket>
├── LBRACKET([)
├── NUMBER(1)


RBRACKET(])
ASSIGN_OPERATOR(:=)
<expression>
<simple-expression>
<term>
<factor>
NUMBER(1.0)
SEMICOLON(;)
<statement>
<assignment-statement>
IDENTIFIER(arr)
<array-bucket>
LBRACKET([)
NUMBER(2)
RBRACKET(])
ASSIGN_OPERATOR(:=)
<expression>
<simple-expression>
<term>
<factor>
NUMBER(2.5555)
SEMICOLON(;)
<statement>
<assignment-statement>
IDENTIFIER(arr)
<array-bucket>
LBRACKET([)
NUMBER(3)
RBRACKET(])
ASSIGN_OPERATOR(:=)
<expression>
<simple-expression>
<term>
<factor>
NUMBER(3.14)
SEMICOLON(;)
<statement>
<assignment-statement>
IDENTIFIER(arr)
<array-bucket>
LBRACKET([)
NUMBER(4)
RBRACKET(])
ASSIGN_OPERATOR(:=)
<expression>
<simple-expression>
<term>
<factor>
NUMBER(4.0)
SEMICOLON(;)
<statement>

```
    │   │   │   └── <assignment-statement>
    │   │   │       ├── IDENTIFIER(arr)
    │   │   │       ├── <array-bucket>
    │   │   │       │   ├── LBRACKET([)
    │   │   │       │   ├── NUMBER(5)
    │   │   │       │   └── RBRACKET(])
    │   │   │       ├── ASSIGN_OPERATOR(:=)
    │   │   │       └── <expression>
    │   │   │           └── <simple-expression>
    │   │   │               └── <term>
    │   │   │                   └── <factor>
    │   │   │                       └── NUMBER(5.0)
    │   │   ├── SEMICOLON(;)
    │   │   ├── <statement>
    │   │   │   └── <assignment-statement>
    │   │   │       ├── IDENTIFIER(c)
    │   │   │       ├── ASSIGN_OPERATOR(:=)
    │   │   │       └── <expression>
    │   │   │           └── <simple-expression>
    │   │   │               └── <term>
    │   │   │                   └── <factor>
    │   │   │                       └── CHAR_LITERAL('a')
    │   │   ├── SEMICOLON(;)
    │   │   ├── <statement>
    │   │   │   └── <assignment-statement>
    │   │   │       ├── IDENTIFIER(c)
    │   │   │       ├── ASSIGN_OPERATOR(:=)
    │   │   │       └── <expression>
    │   │   │           └── <simple-expression>
    │   │   │               └── <term>
    │   │   │                   └── <factor>
    │   │   │                       └── CHAR_LITERAL('b')
    │   │   ├── SEMICOLON(;)
    │   │   ├── <statement>
    │   │   │   └── <assignment-statement>
    │   │   │       ├── IDENTIFIER(c)
    │   │   │       ├── ASSIGN_OPERATOR(:=)
    │   │   │       └── <expression>
    │   │   │           └── <simple-expression>
    │   │   │               └── <term>
    │   │   │                   └── <factor>
    │   │   │                       └── CHAR_LITERAL('c')
    │   │   ├── SEMICOLON(;)
    │   │   ├── <statement>
    │   │   │   └── <assignment-statement>
    │   │   │       ├── IDENTIFIER(s)
    │   │   │       ├── ASSIGN_OPERATOR(:=)
    │   │   │       └── <expression>
    │   │   │           └── <simple-expression>
    │   │   │               └── <term>
    │   │   │                   └── <factor>
    │   │   │
```


└── STRING_LITERAL('seru
sekali')
├── SEMICOLON(;)
├── <statement>
└── <for-statement>
├── KEYWORD(untuk)
├── IDENTIFIER(count)
├── ASSIGN_OPERATOR(:=)
├── <expression>
└── <simple-expression>
└── <term>
└── <factor>
└── NUMBER(1)
├── KEYWORD(ke)
├── <expression>
└── <simple-expression>
└── <term>
└── <factor>
└── NUMBER(5)
├── KEYWORD(lakukan)
└── <statement>
└── <assignment-statement>
├── IDENTIFIER(sum)
├── ASSIGN_OPERATOR(:=)
└── <expression>
└── <simple-expression>
├── <term>
└── <factor>
└──
IDENTIFIER(sum)
├── <additive-operator>
└──
ARITHMETIC_OPERATOR(+)
└── <term>
└── <factor>
└──
IDENTIFIER(count)
├── SEMICOLON(;)
├── <statement>
└── <for-statement>
├── KEYWORD(untuk)
├── IDENTIFIER(count)
├── ASSIGN_OPERATOR(:=)
├── <expression>
└── <simple-expression>
└── <term>
└── <factor>
└── NUMBER(5)
├── KEYWORD(turun_ke)
├── <expression>
└── <simple-expression>


└── <term>
└── <factor>
└── NUMBER(1)
├── KEYWORD(lakukan)
└── <statement>
└── <assignment-statement>
├── IDENTIFIER(avg)
├── ASSIGN_OPERATOR(:=)
└── <expression>
└── <simple-expression>
├── <term>
└── <factor>
IDENTIFIER(avg)
├── <additive-operator>
ARITHMETIC_OPERATOR(+)
└── <term>
└── <factor>
IDENTIFIER(arr)
└── <array-bucket>
LBRACKET([)
IDENTIFIER(count)
RBRACKET(])
├── SEMICOLON(;)
├── <statement>
└── <procedure/function-call>
├── IDENTIFIER(showMessage)
├── LPARENTHESIS(()
├── <parameter-list>
└── <expression>
└── <simple-expression>
└── <term>
└── <factor>
└── IDENTIFIER(greet)
└── RPARENTHESIS())
├── SEMICOLON(;)
├── <statement>
└── <assignment-statement>
├── IDENTIFIER(x)
├── ASSIGN_OPERATOR(:=)
└── <expression>
└── <simple-expression>
└── <term>
└── <factor>
<procedure/function-call>



Bukti Input (Screenshot)

test > milestone-2 > success_all.pas

```
program test_full_tree;

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

tipe
  index = integer;
  realArray = larik[1..5] dari Real;
  letter = char;
  flag = boolean;
  koordinat = rekaman
    x: Real;
    y: Real;
    valid: boolean;
  selesai;

variabel
  x, y, z, sum, avg, count: integer;
  c: char;
  s: larik[1..10] dari char;
  arr: integer;
  done: boolean;

prosedur ok(msg: char);
mulai
  jika tidak done maka
    writeln('Program ', msg, ' seru sekali')
  selain_itu
    writeln('Selesai');
selesai;

fungsi add(a, b: integer): integer;
mulai
  add := a + b;
selesai;
```

```
fungsi add(a, b: integer): integer;
mulai
  add := a + b;
selesai;

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

  done := false;

  jika (sum >= maxVal) dan tidak done maka
    done := true
  selain_itu jika (sum < maxVal) atau (x <> y) maka
    done := false;

  jika (x <= y) maka
    done := false;

  selama (x > 0) lakukan
  mulai
    x := x - 1;
  selesai;

  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';
```

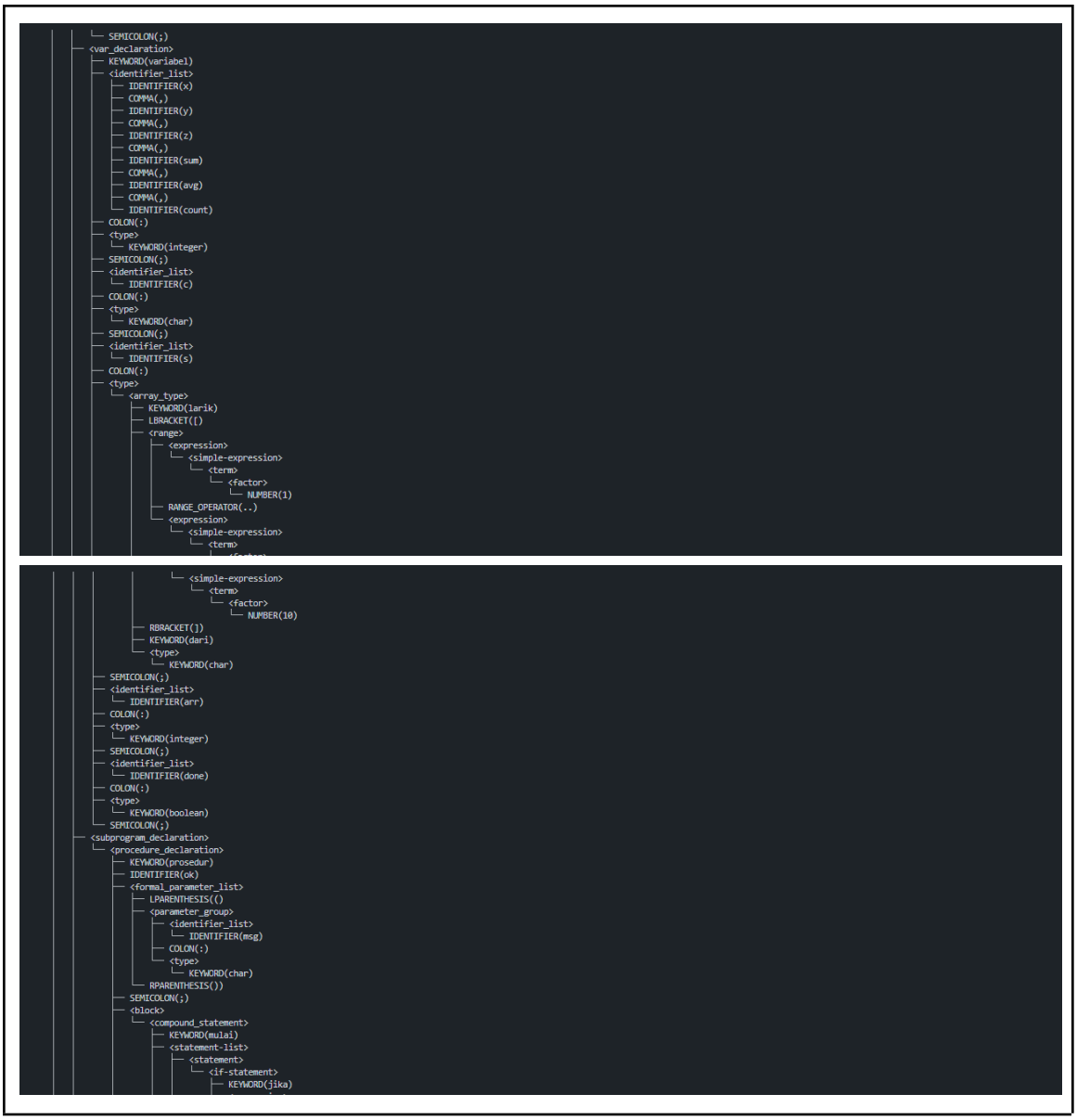
```
  arr[1] := 1.0;
  arr[2] := 2.5555;
  arr[3] := 3.14;
  arr[4] := 4.0;
  arr[5] := 5.0;

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

  untuk count := 1 ke 5 lakukan
    sum := sum + count;

  untuk count := 5 turun_ke 1 lakukan
    avg := avg + arr[count];

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

selesai.
```

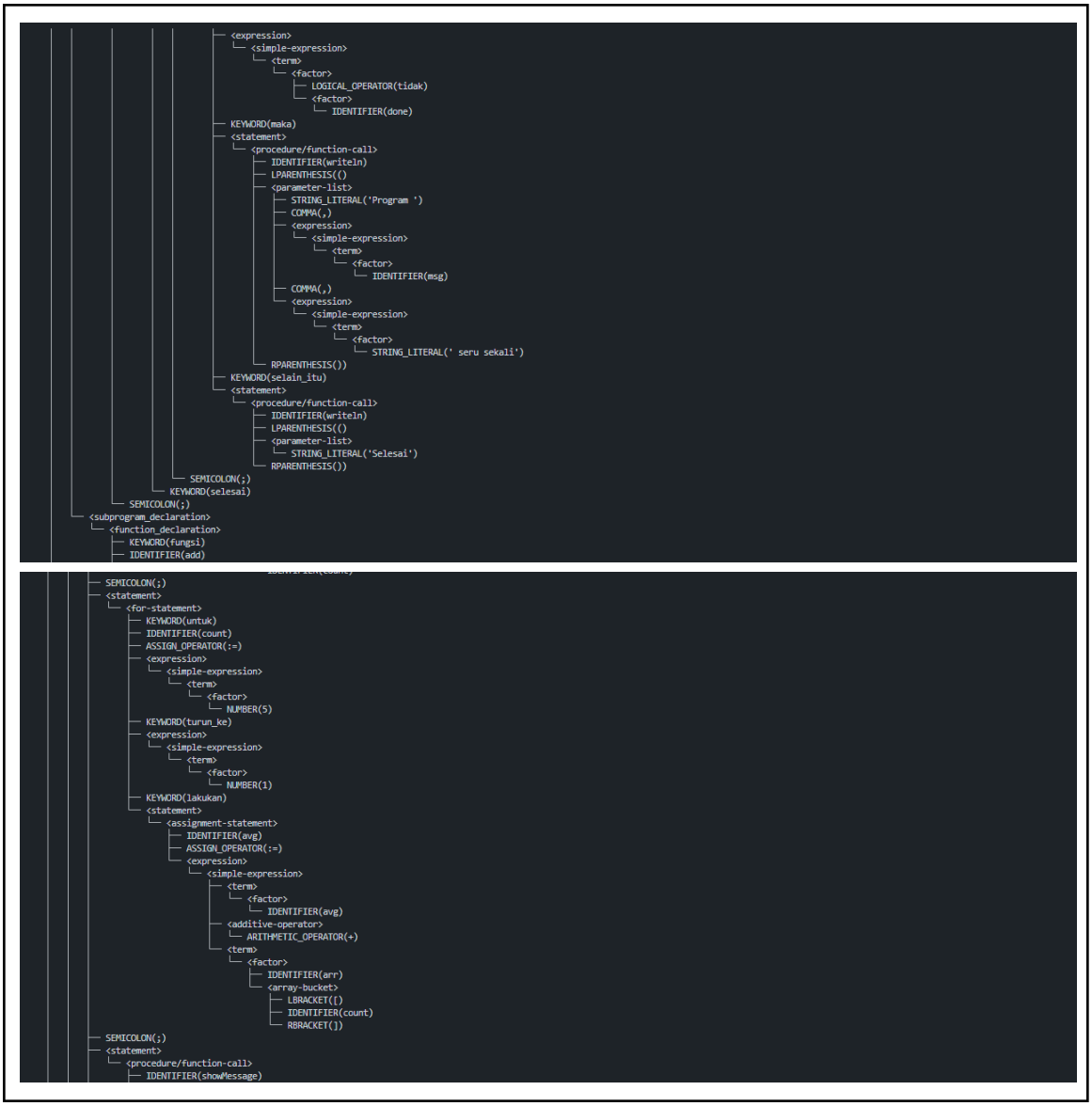
Bukti Output (Screenshot)

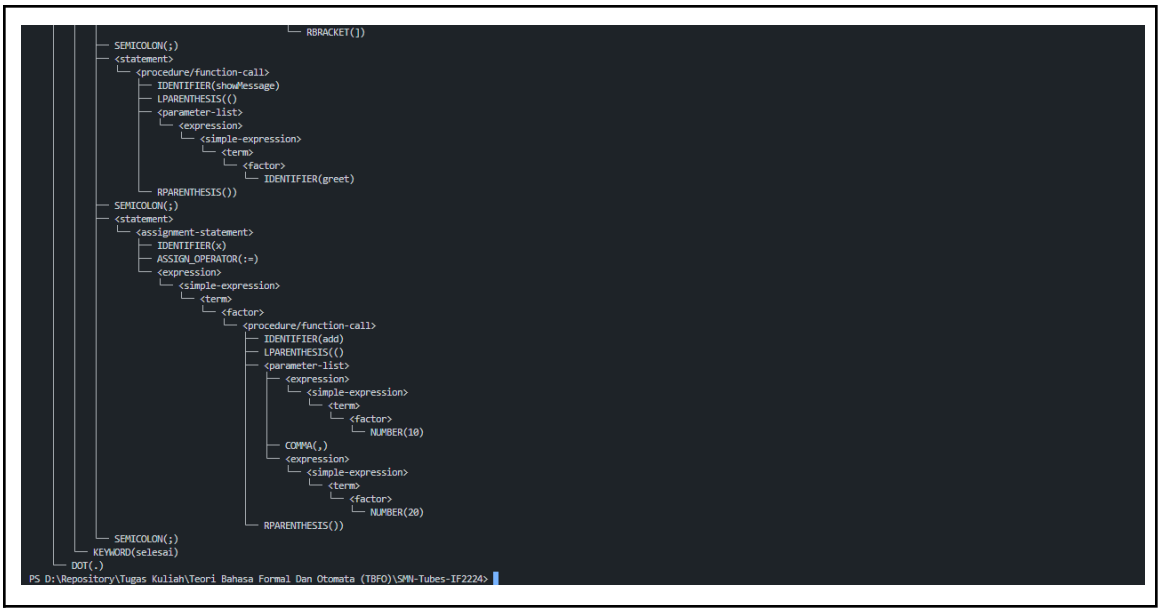
```
================ SYNTAX ANALYSIS ================
└── <program>
    ├── <program_header>
    │   ├── KEYWORD(program)
    │   ├── IDENTIFIER(test_full_tree)
    │   └── SEMICOLON(;)
    ├── <declaration_part>
    │   ├── <const_declaration>
    │   │   ├── KEYWORD(konstanta)
    │   │   ├── IDENTIFIER(maxVal)
    │   │   ├── RELATIONAL_OPERATOR(=)
    │   │   ├── NUMBER(100)
    │   │   ├── SEMICOLON(;)
    │   │   ├── IDENTIFIER(piVal)
    │   │   ├── RELATIONAL_OPERATOR(=)
    │   │   ├── NUMBER(3.14)
    │   │   ├── SEMICOLON(;)
    │   │   ├── IDENTIFIER(greet)
    │   │   ├── RELATIONAL_OPERATOR(=)
    │   │   ├── STRING_LITERAL('tbfo')
    │   │   └── SEMICOLON(;)
    │   ├── <type_declaration>
    │   │   ├── KEYWORD(tipe)
    │   │   ├── IDENTIFIER(index)
    │   │   ├── RELATIONAL_OPERATOR(=)
    │   │   ├── <type_definition>
    │   │   │   └── <type>
    │   │   │       └── KEYWORD(integer)
    │   │   ├── SEMICOLON(;)
    │   │   ├── IDENTIFIER(realArray)
    │   │   ├── RELATIONAL_OPERATOR(=)
    │   │   ├── <type_definition>
    │   │   │   └── <type>
    │   │   │       └── <array_type>
    │   │   │           ├── KEYWORD(larik)
    │   │   │           ├── LBRACKET([)
    │   │   │           ├── <range>
    │   │   │           │   ├── <expression>
    │   │   │           │   │   └── <simple-expression>
    │   │   │           │   │       └── <term>
    │   │   │           │   │           └── <factor>
    │   │   │           │   │               └── NUMBER(1)
```

```
    │   │   │           │   │               └── <factor>
    │   │   │           │   │                   └── NUMBER(1)
    │   │   │           │   ├── RANGE_OPERATOR(..)
    │   │   │           │   └── <expression>
    │   │   │           │       └── <simple-expression>
    │   │   │           │           └── <term>
    │   │   │           │               └── <factor>
    │   │   │           │                   └── NUMBER(5)
    │   │   │           ├── RBRACKET(])
    │   │   │           ├── KEYWORD(dari)
    │   │   │           └── <type>
    │   │   │               └── KEYWORD(Real)
    │   │   ├── SEMICOLON(;)
    │   │   ├── IDENTIFIER(letter)
    │   │   ├── RELATIONAL_OPERATOR(=)
    │   │   ├── <type_definition>
    │   │   │   └── <type>
    │   │   │       └── KEYWORD(char)
    │   │   ├── SEMICOLON(;)
    │   │   ├── IDENTIFIER(flag)
    │   │   ├── RELATIONAL_OPERATOR(=)
    │   │   ├── <type_definition>
    │   │   │   └── <type>
    │   │   │       └── KEYWORD(boolean)
    │   │   ├── SEMICOLON(;)
    │   │   ├── IDENTIFIER(koordinat)
    │   │   ├── RELATIONAL_OPERATOR(=)
    │   │   ├── <type_definition>
    │   │   │   └── <record_type>
    │   │   │       ├── KEYWORD(rekaman)
    │   │   │       ├── <identifier_list>
    │   │   │       │   └── IDENTIFIER(x)
    │   │   │       ├── COLON(:)
    │   │   │       ├── <type>
    │   │   │       │   └── KEYWORD(Real)
    │   │   │       ├── SEMICOLON(;)
    │   │   │       ├── <identifier_list>
    │   │   │       │   └── IDENTIFIER(y)
    │   │   │       ├── COLON(:)
    │   │   │       ├── <type>
    │   │   │       │   └── KEYWORD(Real)
    │   │   │       └── SEMICOLON(;)
```

```
    │   │   ├── <type_definition>
    │   │   │   └── <record_type>
    │   │   │       ├── KEYWORD(rekaman)
    │   │   │       ├── <identifier_list>
    │   │   │       │   └── IDENTIFIER(x)
    │   │   │       ├── COLON(:)
    │   │   │       ├── <type>
    │   │   │       │   └── KEYWORD(Real)
    │   │   │       ├── SEMICOLON(;)
    │   │   │       ├── <identifier_list>
    │   │   │       │   └── IDENTIFIER(y)
    │   │   │       ├── COLON(:)
    │   │   │       ├── <type>
    │   │   │       │   └── KEYWORD(Real)
    │   │   │       ├── SEMICOLON(;)
    │   │   │       ├── <identifier_list>
    │   │   │       │   └── IDENTIFIER(valid)
    │   │   │       ├── COLON(:)
    │   │   │       ├── <type>
    │   │   │       │   └── KEYWORD(boolean)
    │   │   │       ├── SEMICOLON(;)
    │   │   │       └── KEYWORD(selesai)
    │   │   └── SEMICOLON(;)
    │   ├── <var_declaration>
    │   │   ├── KEYWORD(variabel)
    │   │   ├── <identifier_list>
    │   │   │   ├── IDENTIFIER(x)
    │   │   │   ├── COMMA(,)
    │   │   │   ├── IDENTIFIER(y)
    │   │   │   ├── COMMA(,)
```


└── SEMICOLON(;)
<var_declaration>
├── KEYWORD(variabel)
<identifier_list>
├── IDENTIFIER(x)
├── COMMA(,)
├── IDENTIFIER(y)
├── COMMA(,)
├── IDENTIFIER(z)
├── COMMA(,)
├── IDENTIFIER(sum)
├── COMMA(,)
├── IDENTIFIER(avg)
├── COMMA(,)
└── IDENTIFIER(count)
├── COLON(:)
<type>
└── KEYWORD(integer)
├── SEMICOLON(;)
<identifier_list>
└── IDENTIFIER(c)
├── COLON(:)
<type>
└── KEYWORD(char)
├── SEMICOLON(;)
<identifier_list>
└── IDENTIFIER(s)
├── COLON(:)
<type>
└── <array_type>
├── KEYWORD(larik)
├── LBRACKET([)
<range>
<expression>
└── <simple-expression>
└── <term>
└── <factor>
└── NUMBER(1)
├── RANGE_OPERATOR(..)
└── <expression>
└── <simple-expression>
└── <term>
└── <simple-expression>
└── <term>
└── <factor>
└── NUMBER(10)
├── RBRACKET(])
├── KEYWORD(dari)
└── <type>
└── KEYWORD(char)
├── SEMICOLON(;)
<identifier_list>
└── IDENTIFIER(arr)
├── COLON(:)
<type>
└── KEYWORD(integer)
├── SEMICOLON(;)
<identifier_list>
└── IDENTIFIER(done)
├── COLON(:)
<type>
└── KEYWORD(boolean)
└── SEMICOLON(;)
<subprogram_declaration>
└── <procedure_declaration>
├── KEYWORD(prosedur)
├── IDENTIFIER(ok)
<formal_parameter_list>
├── LPARENTHESIS(()
<parameter_group>
<identifier_list>
└── IDENTIFIER(msg)
├── COLON(:)
<type>
└── KEYWORD(char)
└── RPARENTHESIS())
├── SEMICOLON(;)
<block>
└── <compound_statement>
├── KEYWORD(mulai)
<statement-list>
<statement>
└── <if-statement>
├── KEYWORD(jika)


├── <expression>
└── <simple-expression>
└── <term>
└── <factor>
├── LOGICAL_OPERATOR(tidak)
└── <factor>
└── IDENTIFIER(done)
├── KEYWORD(maka)
├── <statement>
└── <procedure/function-call>
├── IDENTIFIER(writeln)
├── LPARENTHESIS(()
├── <parameter-list>
├── STRING_LITERAL('Program ')
├── COMMA(,)
├── <expression>
└── <simple-expression>
└── <term>
└── <factor>
└── IDENTIFIER(msg)
├── COMMA(,)
└── <expression>
└── <simple-expression>
└── <term>
└── <factor>
└── STRING_LITERAL(' seru sekali')
└── RPARENTHESIS())
├── KEYWORD(selain_itu)
└── <statement>
└── <procedure/function-call>
├── IDENTIFIER(writeln)
├── LPARENTHESIS(()
├── <parameter-list>
└── STRING_LITERAL('Selesai')
└── RPARENTHESIS())
└── SEMICOLON(;)
└── KEYWORD(selesai)
└── SEMICOLON(;)
└── <subprogram_declaration>
└── <function_declaration>
├── KEYWORD(fungsi)
├── IDENTIFIER(add)
├── SEMICOLON(;)
├── <statement>
└── <for-statement>
├── KEYWORD(untuk)
├── IDENTIFIER(count)
├── ASSIGN_OPERATOR(:=)
├── <expression>
└── <simple-expression>
└── <term>
└── <factor>
└── NUMBER(5)
├── KEYWORD(turun_ke)
├── <expression>
└── <simple-expression>
└── <term>
└── <factor>
└── NUMBER(1)
├── KEYWORD(lakukan)
└── <statement>
└── <assignment-statement>
├── IDENTIFIER(avg)
├── ASSIGN_OPERATOR(:=)
└── <expression>
└── <simple-expression>
├── <term>
└── <factor>
└── IDENTIFIER(avg)
├── <additive-operator>
└── ARITHMETIC_OPERATOR(+)
└── <term>
└── <factor>
├── IDENTIFIER(arr)
└── <array-bucket>
├── LBRACKET([)
├── IDENTIFIER(count)
└── RBRACKET(])
├── SEMICOLON(;)
├── <statement>
└── <procedure/function-call>
├── IDENTIFIER(showMessage)


└── RBRACKET(])
├── SEMICOLON(;)
├── <statement>
└── <procedure/function-call>
├── IDENTIFIER(showMessage)
├── LPARENTHESIS(()
├── <parameter-list>
└── <expression>
└── <simple-expression>
└── <term>
└── <factor>
└── IDENTIFIER(greet)
└── RPARENTHESIS())
├── SEMICOLON(;)
├── <statement>
└── <assignment-statement>
├── IDENTIFIER(x)
├── ASSIGN_OPERATOR(:=)
└── <expression>
└── <simple-expression>
└── <term>
└── <factor>
└── <procedure/function-call>
├── IDENTIFIER(add)
├── LPARENTHESIS(()
├── <parameter-list>
├── <expression>
└── <simple-expression>
└── <term>
└── <factor>
└── NUMBER(10)
├── COMMA(,)
└── <expression>
└── <simple-expression>
└── <term>
└── <factor>
└── NUMBER(20)
└── RPARENTHESIS())
└── SEMICOLON(;)
└── KEYWORD(selesai)
└── DOT(.)
PS D:\Repository\Tugas Kuliah\Teori Bahasa Formal Dan Otomata (TBFO)\SMN-Tubes-IF2224>

Kesimpulan dan Saran

1. Kesimpulan

Pada milestone kedua tugas besar ini, parser untuk bahasa Pascal-S berhasil diimplementasikan menggunakan pendekatan *recursive descent parsing* berdasarkan aturan grammar yang telah ditentukan. Parser mampu membaca deretan token yang dihasilkan oleh lexical analyzer dan membentuk *parse tree* secara struktural untuk setiap konstruksi bahasa seperti deklarasi, ekspresi aritmetika, pernyataan kontrol, serta prosedur/fungsi. Pengujian menunjukkan bahwa parser dapat mengenali berbagai bentuk input valid dan memberikan pesan error sintaks yang jelas ketika terjadi kesalahan. Selain itu, mekanisme *error handling* yang diterapkan memastikan program tidak langsung berhenti, sehingga proses debugging menjadi lebih mudah

2. Saran

Untuk milestone berikutnya, disarankan untuk menambahkan komponen *semantic analysis* yang dapat memanfaatkan struktur parse tree yang telah terbentuk, seperti pengecekan tipe data, *identifier resolution*, serta validasi terhadap aturan-aturan semantik Pascal-S.

Lampiran

1. Pranala Repositori Github

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

2. Pembagian Tugas

NIM	Nama	Tugas	Persentase
13523002	Refki Alfarizi	- Inisiasi projek dan implementasi fungsionalitas dasar - Laporan	25%
13523028	Muhammad Aditya Rahmadeni	- Merancang aturan DFA untuk token range - Laporan	25%
13523088	Aryo Bama Wiratama	- Implementasi setengah grammar - *Testing*	25%
13523116	Fityatul Haq Rosyidi	- Implementasi parsetree dan setengah grammar - *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