Laporan Milestone 2
Tugas Besar 1 IF3170 Intelegensi Artifisial
Syntax Analysis

Disusun oleh:
Maggie Zeta Rosida S - 13521117
Buege Mahara Putra - 13523037
Hanif Kalyana Aditya - 13523041
Jethro Jens Norbert Simatupang - 13523081

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132
2025

BAB 1
LANDASAN TEORI

1.1 Analisis Sintaks atau Parser

Pada Milestone 1 sebelumnya, telah diimplementasikan tahapan pertama dari proses kompilasi yaitu *Lexical Analysis*. Di tahap kedua ini, setelah program membaca input sebagai rangkaian karakter mentah lalu mengelompokkan karakter-karakter tersebut menjadi unit-unit bermakna yaitu token, token tersebut kemudian diperiksa urutannya agar sesuai dengan aturan tata bahasa (*grammar*) yang disebut *Syntax Analysis* atau *Parser*. *Parser* tersebut akan membangun struktur sintaks seperti *Parse Tree* yang merepresentasikan struktur hierarkis program. Selain itu Parser bertujuan untuk :

- Memastikan urutan token membentuk struktur sintaks yang valid.
- Mendeteksi dan melaporkan kesalahan sintaks (syntax error).
- Mempersiapkan representasi program untuk tahap berikutnya.

1.2 Recursive Descent Parser

Dalam melakukan analisis sintaksis, digunakan Recursive Descent Parser. Recursive Descent Parser adalah *top-down parser* yang memproses masukan berdasarkan serangkaian fungsi rekursif yang setiap fungsinya bersesuaian dengan aturan tata bahasa. Parser ini mengurai masukan dari kiri ke kanan, membangun pohon parse dengan mencocokkan aturan produksi tata bahasa. Parser ini mudah diimplementasikan dan cocok untuk tata bahasa sehingga keputusan dapat dibuat berdasarkan satu token *lookahead* [1].

1.3 Parse tree

Parse Tree atau yang juga dikenal sebagai *syntax tree* adalah representasi hierarkis menyerupai pohon dari turunan sebuah *string* menurut tata bahasa formal. *Parse Tree* dirancang sedemikian rupa sehingga penelusuran berurutan yaitu penelusuran dari kiri, akar, kanan hingga menghasilkan string input asli. Pohon parse merupakan dasar fundamental untuk menjelaskan struktur sintaksis ke kode sumber dan membantu dalam deteksi kesalahan serta tahap kompilasi selanjutnya. Adapun aturan dalam menggambar sebuah Parse Tree antara lain :

- Semua simpul daun harus berupa terminal.
- Semua simpul interior harus berupa non-terminal.
- Penelusuran berurutan menghasilkan string input asli [2].

Dalam Milestone 2 Tugas Besar TBFO ini, setelah kode melewati tahap *lexical analysis* lalu proses kompilasi berlanjut ke tahap analisis sintaksis yang menghasilkan *Parse Tree*, *Parse Tree* tersebut nantinya akan menjadi masukan bagi Milestone 3 atau tahap *semantic analysis* dalam proses kompilasi.

BAB 2
PERANCANGAN & IMPLEMENTASI

2.1 Perancangan Sistem

2.1.1 Struktur Program

Sebelumnya, tahapan *Lexical Analysis* (lexer.py) telah selesai diimplementasikan yang dihubungkan pada compiler.py. Hasil dari tahap tersebut adalah kumpulan token yang selanjutnya diproses untuk tahapan Syntax Analysis sehingga dihubungkan dengan parser.py. Di dalam file ini, parser membangun parse tree dengan menggunakan Recursive Descent Algorithm sesuai dengan grammar yang didefinisikan (dari spesifikasi). Dalam parse tree tersebut setiap node-nya direpresentasikan dengan class ParseNode dari parse_tree.py yang memiliki tiga buah field antara lain name, children, dan juga token. Lebih spesifiknya, children direpresentasikan sebagai list of ParseNode mengingat hampir semua rule memiliki > 1 anak. Sementara itu, token tetap sebagai Token yang diambil dari tokens.py.

Setelah tahap *Lexical Analysis* selesai, parse tree selanjutnya ditampilkan menggunakan method dari parse_tree.py. Kedepannya parse tree ini akan dijadikan sebagai masukan dari tahap ketiga dari sebuah compiler yaitu *Semantic Analysis*.

2.1.2 Grammar Pascal-S

Grammar Pascal-S yang diimplementasikan mengikuti spesifikasi yang diberikan dan di-hardcode dalam program. Berikut adalah aturan produksi utama:

```
<program> ::= <program-header> <declaration-part>
<compound-statement> DOT

<program-header> ::= KEYWORD(program) IDENTIFIER SEMICOLON

<declaration-part> ::=
    { <const-declaration> }
    { <type-declaration> }
    { <var-declaration> }
    { <subprogram-declaration> }

<const-declaration> ::= KEYWORD(konstanta) <const-item> {
<const-item> }

<const-item> ::= IDENTIFIER RELATIONAL_OPERATOR(=) <const-value>
SEMICOLON

<type-declaration> ::= KEYWORD(tipe) <type-item> { <type-item> }

<type-item> ::= IDENTIFIER RELATIONAL_OPERATOR(=) <type-definition>
SEMICOLON

<var-declaration> ::= KEYWORD(variabel) <var-item> { <var-item> }

<var-item> ::= <identifier-list> COLON <type> SEMICOLON
```

```
<compound-statement> ::= KEYWORD(mulai) <statement-list>
KEYWORD(selesai)

<statement> ::=
    <assignment-statement>
  | <compound-statement>
  | <if-statement>
  | <while-statement>
  | <for-statement>
  | <repeat-statement>
  | <case-statement>
  | <procedure-call>
```

2.1.3 Struktur Recursive Descent Parser

Parser dirancang sebagai predictive recursive descent parser dengan karakteristik:

- Setiap non-terminal diimplementasikan sebagai method terpisah
- Menggunakan lookahead token untuk menentukan production rule
- Error detection dan reporting dengan informasi posisi yang detail
- Membangun parse tree lengkap sesuai spesifikasi node

2.2 Implementasi Program

2.2.1 Struktur Parser

File: src/parser.py

a. Inisialisasi dan Utility Functions

```
class Parser:
    def __init__(self, tokens: List[Token]):
        self.tokens = tokens
        self.pos = 0

    def current(self) -> Optional[Token]:
        if self.pos < len(self.tokens):
            return self.tokens[self.pos]
        return None

    def expect(self, expected_type: TokenType) -> Token:
        tok = self.current()
        if tok is None:
            raise ParserError(f"Expected {expected_type.name},
but found EOF")
        if tok.type != expected_type:
            raise ParserError(
                f"Expected {expected_type.name}, but found
{tok.type.name} "
                f"at line {tok.line}, column {tok.column}"
            )
        self.pos += 1
        return tok
```

b. Entry Point Parsing

```
def parse(self) -> ParseNode:
    # <program> ::= <program-header> <declaration-part>
<compound-statement> DOT
    root = self.parse_program()

    # Validasi tidak ada token tersisa
    if not self.at_end():
        tok = self.current()
```

```
        raise ParserError(f"Unexpected token
{tok.type.name}({tok.value})")

    return root
```

2.2.2 Implementasi Grammar Utama

a. Program Structure

```
def parse_program(self) -> ParseNode:
    node = ParseNode("<program>")
    node.children.append(self.parse_program_header())
    node.children.append(self.parse_declaration_part())
    node.children.append(self.parse_compound_statement())
    dot_tok = self.expect(TokenType.DOT)
    node.children.append(ParseNode("DOT", token=dot_tok))
    return node

def parse_program_header(self) -> ParseNode:
    node = ParseNode("<program-header>")
    kw_prog = self.expect_keyword("program")
    node.children.append(ParseNode("KEYWORD(program)",
token=kw_prog))
    ident = self.expect(TokenType.IDENTIFIER)
    node.children.append(ParseNode("IDENTIFIER", token=ident))
    semi = self.expect(TokenType.SEMICOLON)
    node.children.append(ParseNode("SEMICOLON", token=semi))
    return node
```

b. Declaration Parts

```
def parse_declaration_part(self) -> ParseNode:
    node = ParseNode("<declaration-part>")
    # { <const-declaration> }
    while self.check_keyword("konstanta"):
        node.children.append(self.parse_const_declaration())
    # { <type-declaration> }
    while self.check_keyword("tipe"):
        node.children.append(self.parse_type_declaration())
    # { <var-declaration> }
    while self.check_keyword("variabel"):
        node.children.append(self.parse_var_declaration())
    # { <subprogram-declaration> }
    while self.check_keyword("prosedur") or
self.check_keyword("fungsi"):
        node.children.append(self.parse_subprogram_declaration())
    return node
```

c. Statements

```
def parse_assignment_statement(self) -> ParseNode:
    node = ParseNode("<assignment-statement>")
    node.children.append(self.parse_variable())
    assign = self.expect(TokenType.ASSIGN_OPERATOR)
    node.children.append(ParseNode("ASSIGN_OPERATOR(:=)",
token=assign))
    node.children.append(self.parse_expression())
    return node

def parse_if_statement(self) -> ParseNode:
    node = ParseNode("<if-statement>")
    kj = self.expect_keyword("jika")
    node.children.append(ParseNode("KEYWORD(jika)", token=kj))
    node.children.append(self.parse_expression())
    km = self.expect_keyword("maka")
    node.children.append(ParseNode("KEYWORD(maka)", token=km))
```

```
    node.children.append(self.parse_statement())
    if self.check_keyword("selainitu"):
        ke = self.expect_keyword("selainitu")
        node.children.append(ParseNode("KEYWORD(selainitu)",
token=ke))
        node.children.append(self.parse_statement())
    return node
```

d. Expressions dengan Precedence

```
def parse_expression(self) -> ParseNode:
    # <expression> ::= <simple-expression>
    #                  { ( <relational-operator> |
LOGICAL_OPERATOR ) <simple-expression> }
    node = ParseNode("<expression>")
    node.children.append(self.parse_simple_expression())
    while True:
        tok = self.current()
        # Handle relational and logical operators
        if self._is_relational_or_logical_operator(tok):
            node.children.append(self.parse_operator())
            node.children.append(self.parse_simple_expression())
        else:
            break
    return node
```

2.2.3 Parser Tree Representation

File: src/parse_tree.py

a. Struktur ParseNode

```
@dataclass
class ParseNode:
    name: str
    children: List["ParseNode"] = field(default_factory=list)
    token: Optional[Token] = None

    def add_child(self, child: "ParseNode") -> None:
        self.children.append(child)
```

b. Visualisasi Parse Tree

```
def print_tree(root: ParseNode) -> None:
    print(root.name)
    print_tree_recursive(root.children, "")

def print_tree_recursive(nodes: List[ParseNode], prefix: str) ->
None:
    for i, node in enumerate(nodes):
        is_last = (i == len(nodes) - 1)
        connector = "└── " if is_last else "├── "

        if node.token is not None:
            label = f"{node.token.type.name}({node.token.value})"
        else:
            label = node.name

        print(prefix + connector + label)
        new_prefix = prefix + ("    " if is_last else "│   ")
        print_tree_recursive(node.children, new_prefix)
```

2.3 Error Handling

2.3.1 Error Detection

- Missing tokens (semicolon, keyword, operator)
- Unexpected tokens
- Structural errors dalam statement dan expression
- Violation of production rules

2.3.2 Informative Error Messages

```
Syntax error: Expected SEMICOLON, but found IDENTIFIER at line 5,
column 12
Syntax error: Unexpected token DOT(.), expected SEMICOLON(;) at line
3, column 15
```

2.4 Teknik Parsing Khusus

2.4.1 Lookahead untuk Disambiguasi

```
def parse_statement(self) -> ParseNode:
    # Gunakan lookahead untuk membedakan assignment vs procedure
call
    if self.current().type == TokenType.IDENTIFIER:
        if self.lookahead(1) and self.lookahead(1).type ==
TokenType.ASSIGN_OPERATOR:
            return self.parse_assignment_statement()
        else:
            return self.parse_procedure_call()
```

2.4.2 Backtracking untuk Ambiguity Resolution

```
def parse_type_definition(self) -> ParseNode:
    # Coba parse sebagai range dulu
    saved_pos = self.pos
    try:
        return self.parse_range()
    except ParserError:
        # Jika gagal, backtrack dan parse sebagai type biasa
        self.pos = saved_pos
        return self.parse_type()
```

BAB 3
PENGUJIAN

3.1 Kasus Uji 1

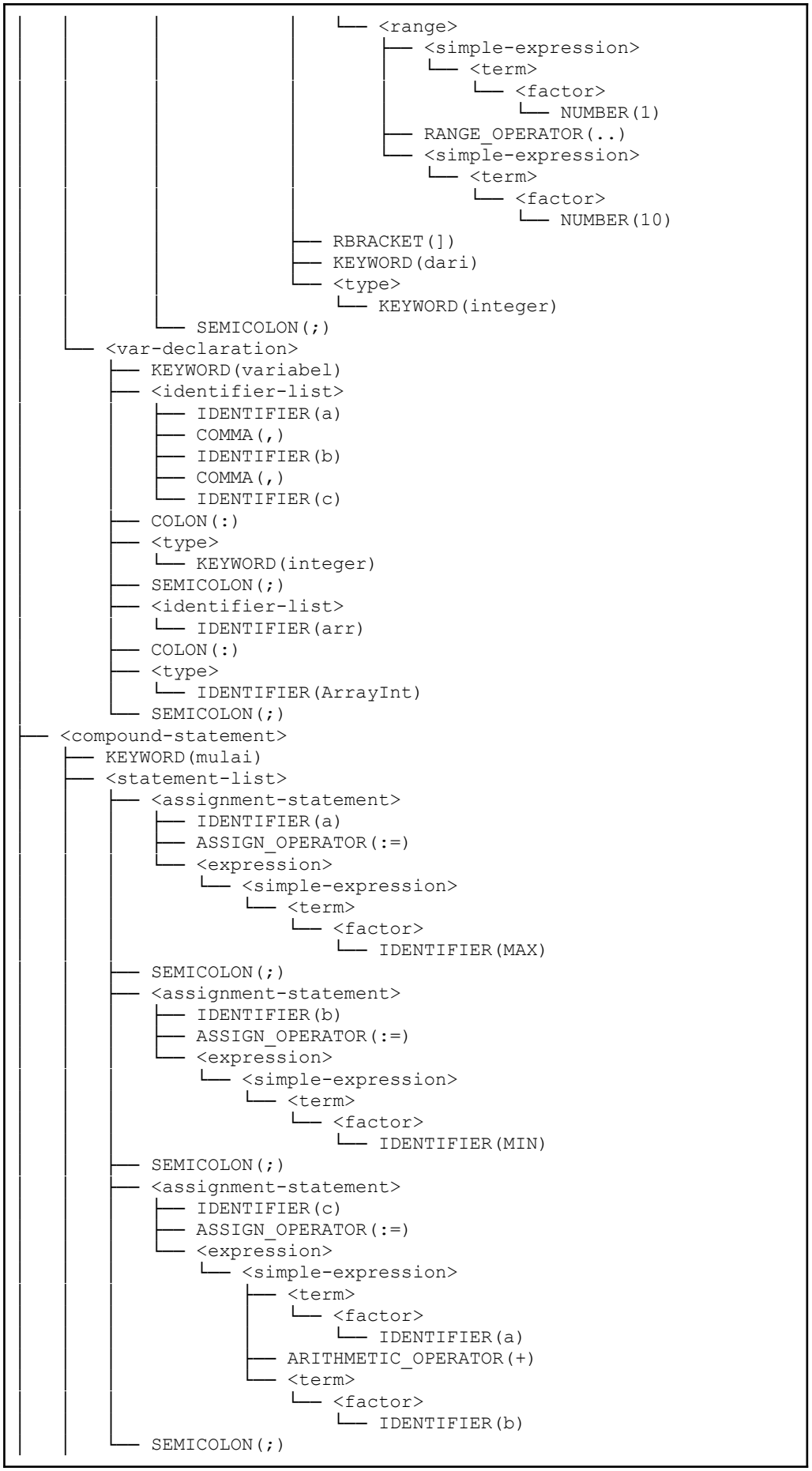
Input 1:

```
program Input1;
konstanta
  MAX = 100;
  MIN = 1;
tipe
  Range = 1..10;
  ArrayInt = larik[1..10] dari integer;
variabel
  a, b, c: integer;
  arr: ArrayInt;
mulai
  a := MAX;
  b := MIN;
  c := a + b;
selesai.
```

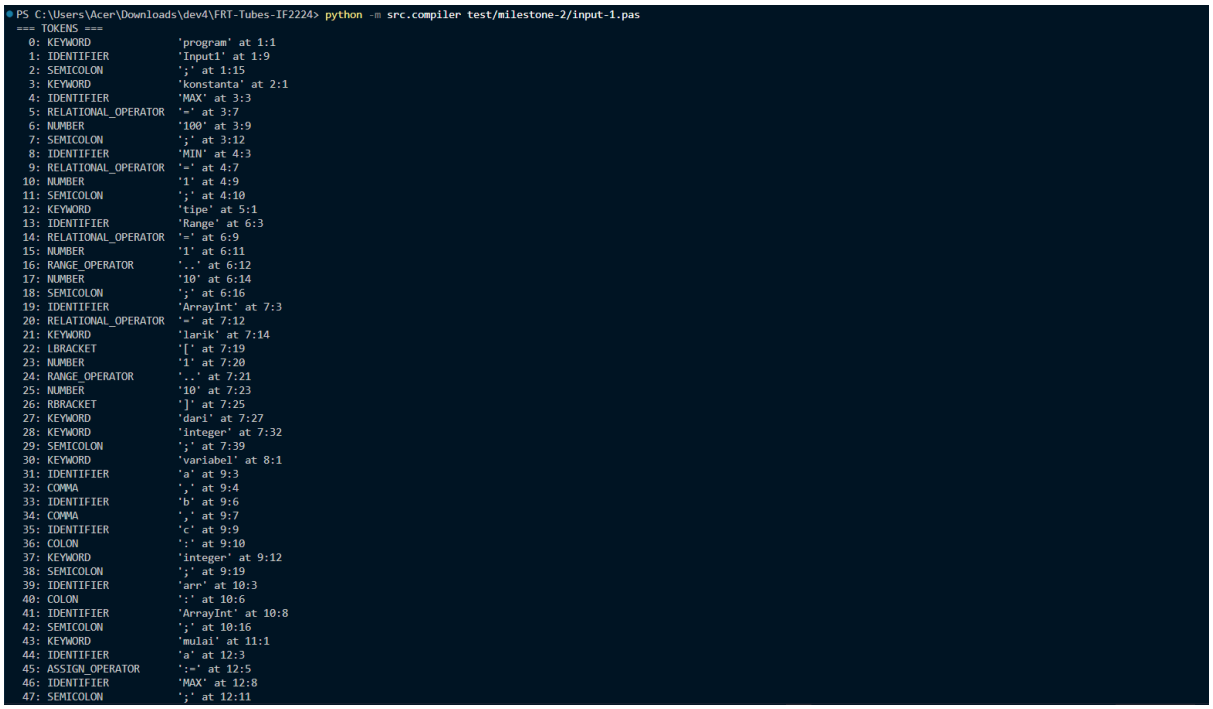
Output 1:

```
=== TOKENS ===
  0: KEYWORD              'program' at 1:1
  1: IDENTIFIER           'Input1' at 1:9
  2: SEMICOLON            ';' at 1:15
  3: KEYWORD              'konstanta' at 2:1
  4: IDENTIFIER           'MAX' at 3:3
  5: RELATIONAL_OPERATOR  '=' at 3:7
  6: NUMBER               '100' at 3:9
  7: SEMICOLON            ';' at 3:12
  8: IDENTIFIER           'MIN' at 4:3
  9: RELATIONAL_OPERATOR  '=' at 4:7
 10: NUMBER               '1' at 4:9
 11: SEMICOLON            ';' at 4:10
 12: KEYWORD              'tipe' at 5:1
 13: IDENTIFIER           'Range' at 6:3
 14: RELATIONAL_OPERATOR  '=' at 6:9
 15: NUMBER               '1' at 6:11
 16: RANGE_OPERATOR       '..' at 6:12
 17: NUMBER               '10' at 6:14
 18: SEMICOLON            ';' at 6:16
 19: IDENTIFIER           'ArrayInt' at 7:3
 20: RELATIONAL_OPERATOR  '=' at 7:12
 21: KEYWORD              'larik' at 7:14
 22: LBRACKET             '[' at 7:19
 23: NUMBER               '1' at 7:20
 24: RANGE_OPERATOR       '..' at 7:21
 25: NUMBER               '10' at 7:23
 26: RBRACKET             ']' at 7:25
 27: KEYWORD              'dari' at 7:27
 28: KEYWORD              'integer' at 7:32
 29: SEMICOLON            ';' at 7:39
 30: KEYWORD              'variabel' at 8:1
 31: IDENTIFIER           'a' at 9:3
 32: COMMA                ',' at 9:4
 33: IDENTIFIER           'b' at 9:6
 34: COMMA                ',' at 9:7
 35: IDENTIFIER           'c' at 9:9
 36: COLON                ':' at 9:10
 37: KEYWORD              'integer' at 9:12
 38: SEMICOLON            ';' at 9:19
 39: IDENTIFIER           'arr' at 10:3
```

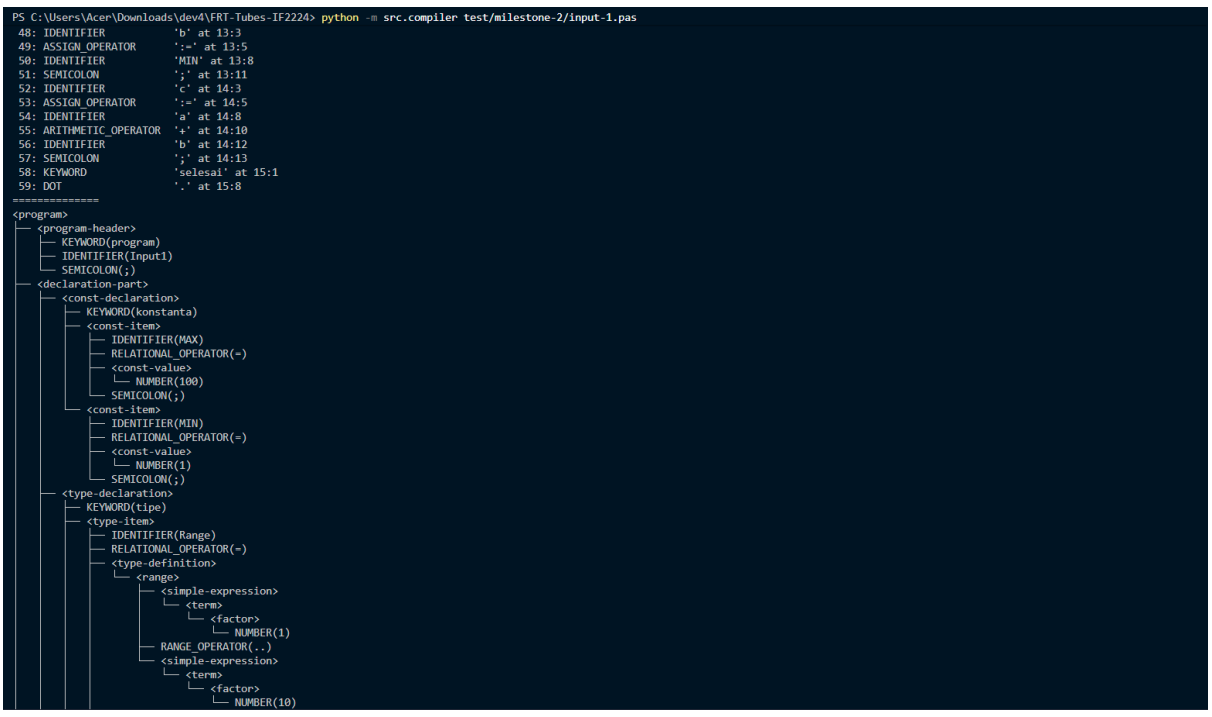
```
 40: COLON                 ':' at 10:6
 41: IDENTIFIER            'ArrayInt' at 10:8
 42: SEMICOLON             ';' at 10:16
 43: KEYWORD               'mulai' at 11:1
 44: IDENTIFIER            'a' at 12:3
 45: ASSIGN_OPERATOR       ':=' at 12:5
 46: IDENTIFIER            'MAX' at 12:8
 47: SEMICOLON             ';' at 12:11
 48: IDENTIFIER            'b' at 13:3
 49: ASSIGN_OPERATOR       ':=' at 13:5
 50: IDENTIFIER            'MIN' at 13:8
 51: SEMICOLON             ';' at 13:11
 52: IDENTIFIER            'c' at 14:3
 53: ASSIGN_OPERATOR       ':=' at 14:5
 54: IDENTIFIER            'a' at 14:8
 55: ARITHMETIC_OPERATOR   '+' at 14:10
 56: IDENTIFIER            'b' at 14:12
 57: SEMICOLON             ';' at 14:13
 58: KEYWORD               'selesai' at 15:1
 59: DOT                   '.' at 15:8
==============
<program>
├── <program-header>
│   ├── KEYWORD(program)
│   ├── IDENTIFIER(Input1)
│   └── SEMICOLON(;)
├── <declaration-part>
│   ├── <const-declaration>
│   │   ├── KEYWORD(konstanta)
│   │   ├── <const-item>
│   │   │   ├── IDENTIFIER(MAX)
│   │   │   ├── RELATIONAL_OPERATOR(=)
│   │   │   ├── <const-value>
│   │   │   │   └── NUMBER(100)
│   │   │   └── SEMICOLON(;)
│   │   └── <const-item>
│   │       ├── IDENTIFIER(MIN)
│   │       ├── RELATIONAL_OPERATOR(=)
│   │       ├── <const-value>
│   │       │   └── NUMBER(1)
│   │       └── SEMICOLON(;)
│   ├── <type-declaration>
│   │   ├── KEYWORD(tipe)
│   │   ├── <type-item>
│   │   │   ├── IDENTIFIER(Range)
│   │   │   ├── RELATIONAL_OPERATOR(=)
│   │   │   ├── <type-definition>
│   │   │   │   └── <range>
│   │   │   │       ├── <simple-expression>
│   │   │   │       │   └── <term>
│   │   │   │       │       └── <factor>
│   │   │   │       │           └── NUMBER(1)
│   │   │   │       ├── RANGE_OPERATOR(..)
│   │   │   │       └── <simple-expression>
│   │   │   │           └── <term>
│   │   │   │               └── <factor>
│   │   │   │                   └── NUMBER(10)
│   │   │   └── SEMICOLON(;)
│   │   └── <type-item>
│   │       ├── IDENTIFIER(ArrayInt)
│   │       ├── RELATIONAL_OPERATOR(=)
│   │       ├── <type-definition>
│   │       │   └── <type>
│   │       │       └── <array-type>
│   │       │           ├── KEYWORD(larik)
│   │       │           ├── LBRACKET([)
│   │       │           ├── <index-specification>
```


└── <range>
├── <simple-expression>
└── <term>
└── <factor>
└── NUMBER(1)
├── RANGE_OPERATOR(..)
└── <simple-expression>
└── <term>
└── <factor>
└── NUMBER(10)
├── RBRACKET(])
├── KEYWORD(dari)
└── <type>
└── KEYWORD(integer)
└── SEMICOLON(;)
└── <var-declaration>
├── KEYWORD(variabel)
├── <identifier-list>
├── IDENTIFIER(a)
├── COMMA(,)
├── IDENTIFIER(b)
├── COMMA(,)
└── IDENTIFIER(c)
├── COLON(:)
├── <type>
└── KEYWORD(integer)
├── SEMICOLON(;)
├── <identifier-list>
└── IDENTIFIER(arr)
├── COLON(:)
├── <type>
└── IDENTIFIER(ArrayInt)
└── SEMICOLON(;)
├── <compound-statement>
├── KEYWORD(mulai)
├── <statement-list>
├── <assignment-statement>
├── IDENTIFIER(a)
├── ASSIGN_OPERATOR(:=)
└── <expression>
└── <simple-expression>
└── <term>
└── <factor>
└── IDENTIFIER(MAX)
├── SEMICOLON(;)
├── <assignment-statement>
├── IDENTIFIER(b)
├── ASSIGN_OPERATOR(:=)
└── <expression>
└── <simple-expression>
└── <term>
└── <factor>
└── IDENTIFIER(MIN)
├── SEMICOLON(;)
├── <assignment-statement>
├── IDENTIFIER(c)
├── ASSIGN_OPERATOR(:=)
└── <expression>
└── <simple-expression>
├── <term>
└── <factor>
└── IDENTIFIER(a)
├── ARITHMETIC_OPERATOR(+)
└── <term>
└── <factor>
└── IDENTIFIER(b)
└── SEMICOLON(;)

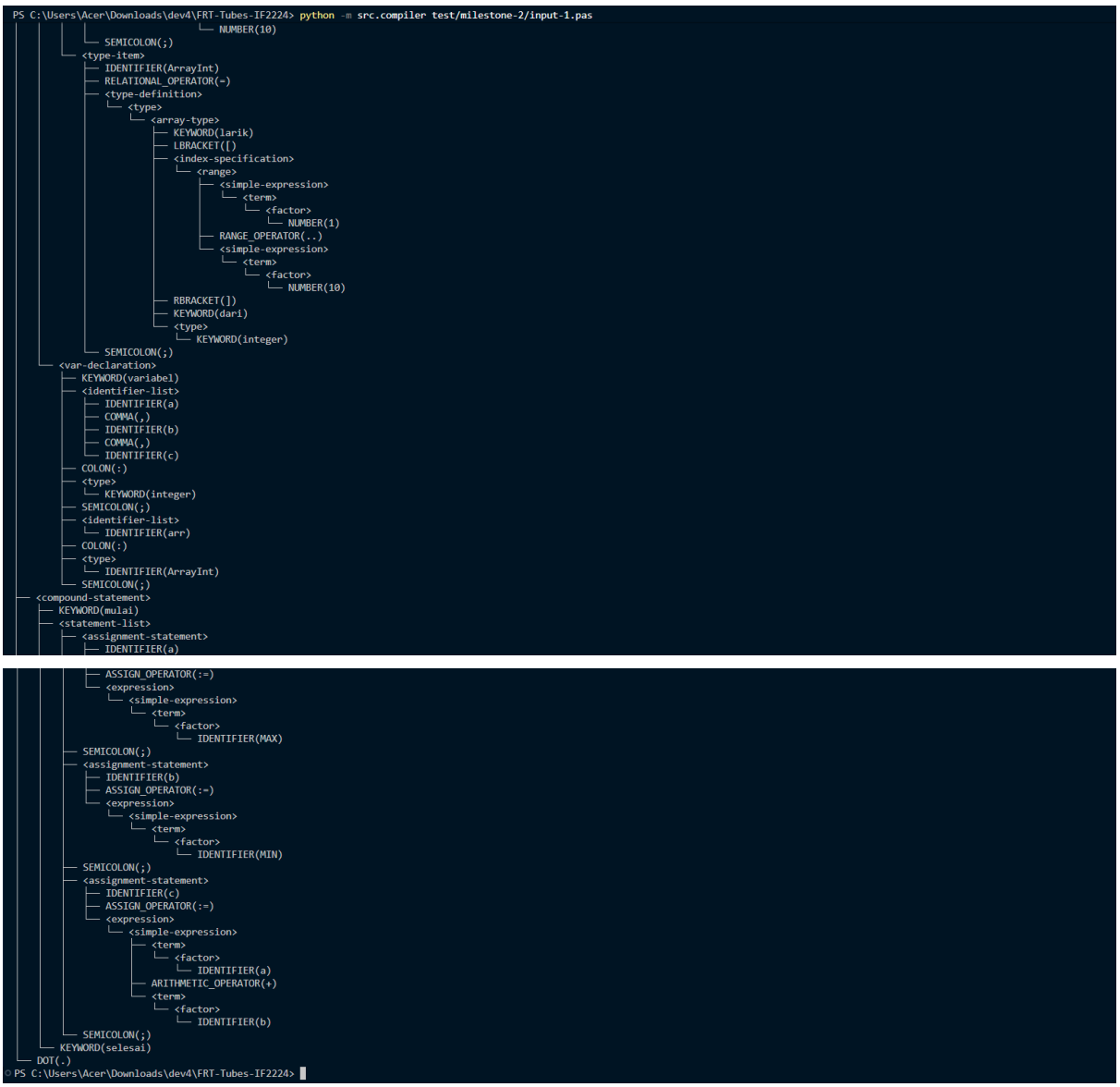
Screenshot 1:



```
PS C:\Users\Acer\Downloads\dev4\FRT-Tubes-IF2224> python -m src.compiler test/milestone-2/input-1.pas
=== TOKENS ===
  0: KEYWORD              'program' at 1:1
  1: IDENTIFIER           'Input1' at 1:9
  2: SEMICOLON            ';' at 1:15
  3: KEYWORD              'konstanta' at 2:1
  4: IDENTIFIER           'MAX' at 3:3
  5: RELATIONAL_OPERATOR  '=' at 3:7
  6: NUMBER               '100' at 3:9
  7: SEMICOLON            ';' at 3:12
  8: IDENTIFIER           'MIN' at 4:3
  9: RELATIONAL_OPERATOR  '=' at 4:7
 10: NUMBER               '1' at 4:9
 11: SEMICOLON            ';' at 4:10
 12: KEYWORD              'tipe' at 5:1
 13: IDENTIFIER           'Range' at 6:3
 14: RELATIONAL_OPERATOR  '=' at 6:9
 15: NUMBER               '1' at 6:11
 16: RANGE_OPERATOR       '..' at 6:12
 17: NUMBER               '10' at 6:14
 18: SEMICOLON            ';' at 6:16
 19: IDENTIFIER           'ArrayInt' at 7:3
 20: RELATIONAL_OPERATOR  '=' at 7:12
 21: KEYWORD              'larik' at 7:14
 22: LBRACKET             '[' at 7:19
 23: NUMBER               '1' at 7:20
 24: RANGE_OPERATOR       '..' at 7:21
 25: NUMBER               '10' at 7:23
 26: RBRACKET             ']' at 7:25
 27: KEYWORD              'dari' at 7:27
 28: KEYWORD              'integer' at 7:32
 29: SEMICOLON            ';' at 7:39
 30: KEYWORD              'variabel' at 8:1
 31: IDENTIFIER           'a' at 9:3
 32: COMMA                ',' at 9:4
 33: IDENTIFIER           'b' at 9:6
 34: COMMA                ',' at 9:7
 35: IDENTIFIER           'c' at 9:9
 36: COLON                ':' at 9:10
 37: KEYWORD              'integer' at 9:12
 38: SEMICOLON            ';' at 9:19
 39: IDENTIFIER           'arr' at 10:3
 40: COLON                ':' at 10:6
 41: IDENTIFIER           'ArrayInt' at 10:8
 42: SEMICOLON            ';' at 10:16
 43: KEYWORD              'mulai' at 11:1
 44: IDENTIFIER           'a' at 12:3
 45: ASSIGN_OPERATOR      ':=' at 12:5
 46: IDENTIFIER           'MAX' at 12:8
 47: SEMICOLON            ';' at 12:11
```



```
PS C:\Users\Acer\Downloads\dev4\FRT-Tubes-IF2224> python -m src.compiler test/milestone-2/input-1.pas
 48: IDENTIFIER           'b' at 13:3
 49: ASSIGN_OPERATOR      ':=' at 13:5
 50: IDENTIFIER           'MIN' at 13:8
 51: SEMICOLON            ';' at 13:11
 52: IDENTIFIER           'c' at 14:3
 53: ASSIGN_OPERATOR      ':=' at 14:5
 54: IDENTIFIER           'a' at 14:8
 55: ARITHMETIC_OPERATOR  '+' at 14:10
 56: IDENTIFIER           'b' at 14:12
 57: SEMICOLON            ';' at 14:13
 58: KEYWORD              'selesai' at 15:1
 59: DOT                  '.' at 15:8
==============
<program>
├── <program-header>
│   ├── KEYWORD(program)
│   ├── IDENTIFIER(Input1)
│   └── SEMICOLON(;)
├── <declaration-part>
│   ├── <const-declaration>
│   │   ├── KEYWORD(konstanta)
│   │   ├── <const-item>
│   │   │   ├── IDENTIFIER(MAX)
│   │   │   ├── RELATIONAL_OPERATOR(=)
│   │   │   ├── <const-value>
│   │   │   │   └── NUMBER(100)
│   │   │   └── SEMICOLON(;)
│   │   └── <const-item>
│   │       ├── IDENTIFIER(MIN)
│   │       ├── RELATIONAL_OPERATOR(=)
│   │       ├── <const-value>
│   │       │   └── NUMBER(1)
│   │       └── SEMICOLON(;)
│   ├── <type-declaration>
│   │   ├── KEYWORD(tipe)
│   │   ├── <type-item>
│   │   │   ├── IDENTIFIER(Range)
│   │   │   ├── RELATIONAL_OPERATOR(=)
│   │   │   ├── <type-definition>
│   │   │   │   └── <range>
│   │   │   │       ├── <simple-expression>
│   │   │   │       │   └── <term>
│   │   │   │       │       └── <factor>
│   │   │   │       │           └── NUMBER(1)
│   │   │   │       ├── RANGE_OPERATOR(..)
│   │   │   │       └── <simple-expression>
│   │   │   │           └── <term>
│   │   │   │               └── <factor>
│   │   │   │                   └── NUMBER(10)
```



3.2 Kasus Uji 2

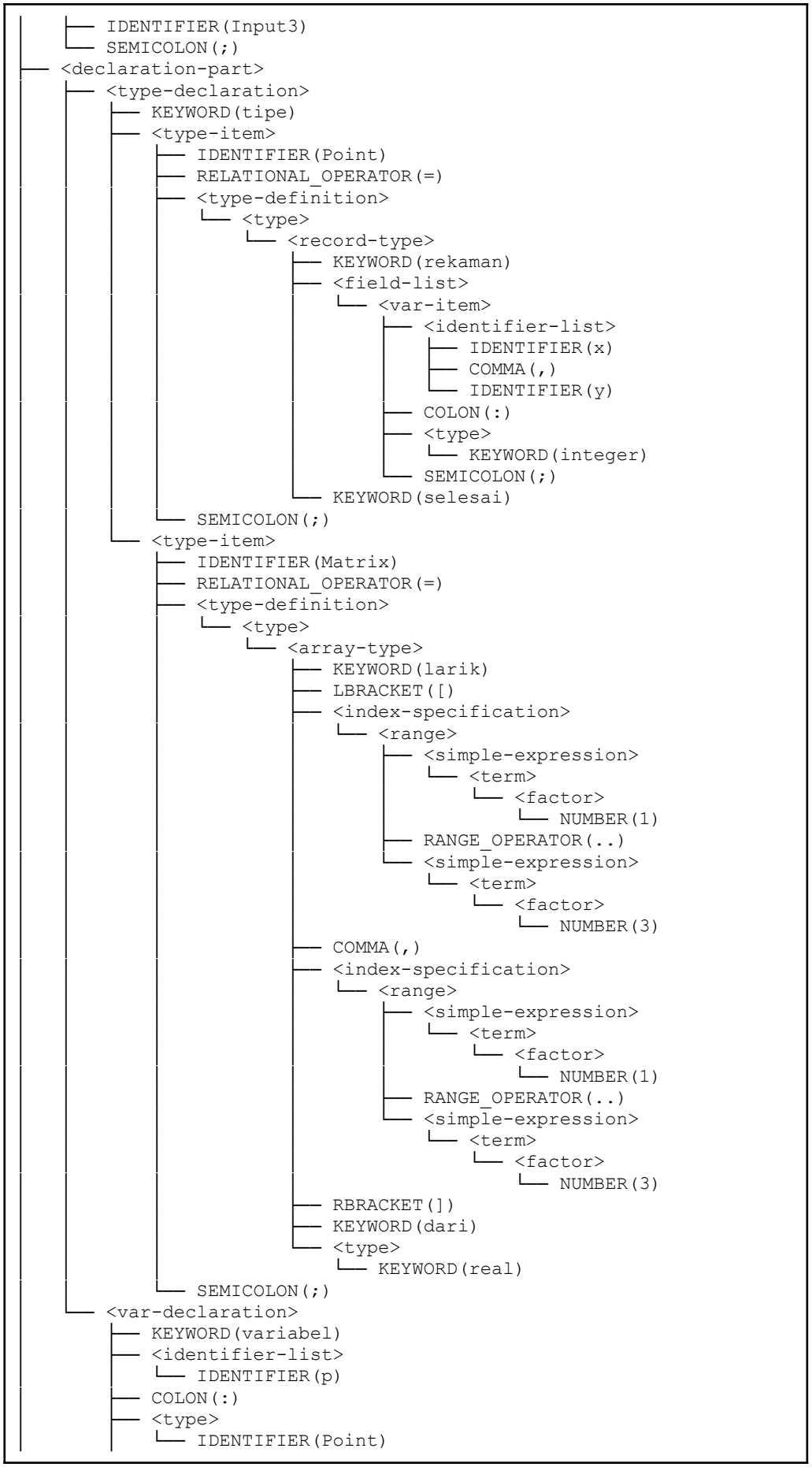
Input 2:

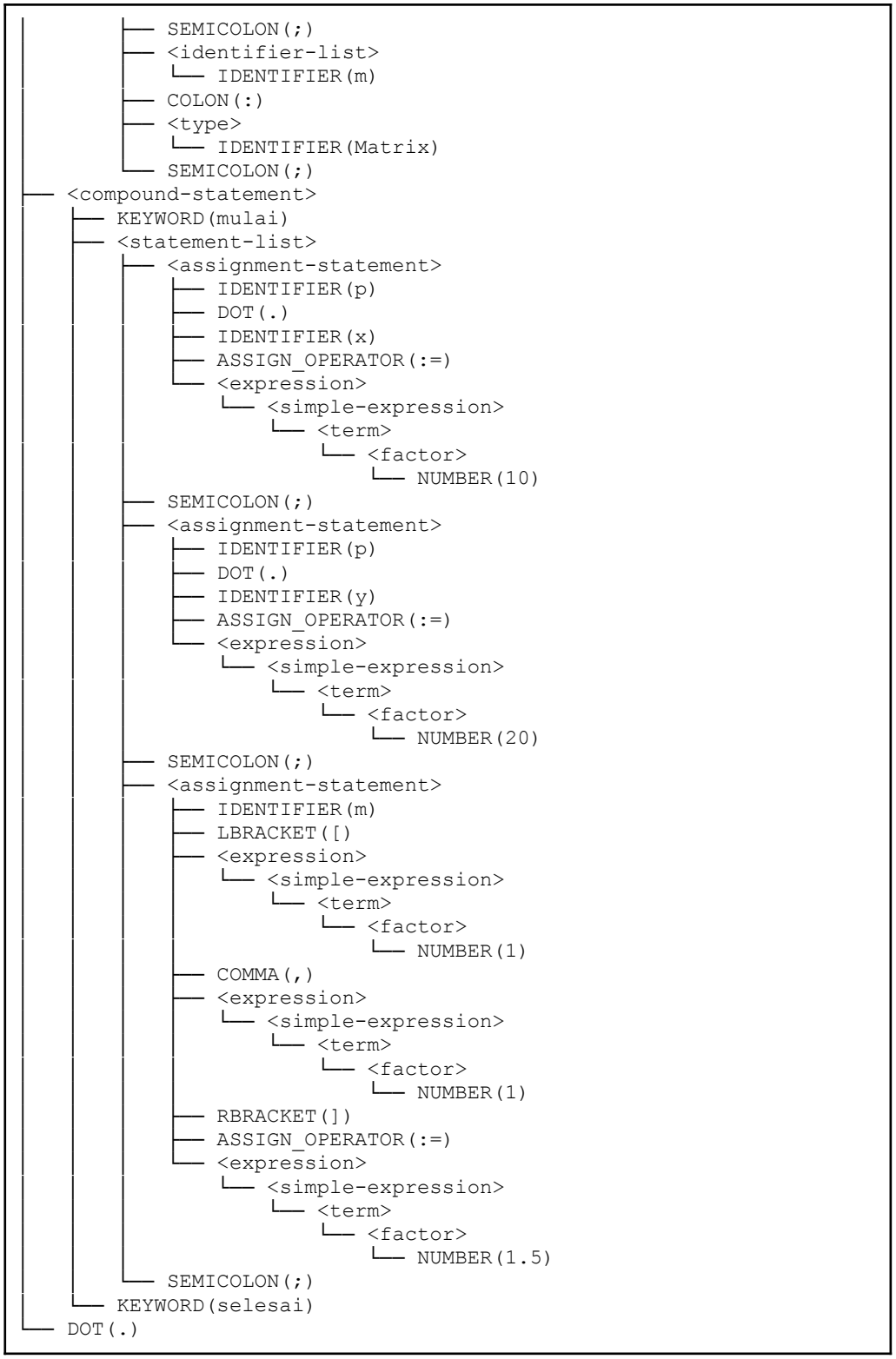
```
program Input3;
tipe
  Point = rekaman
    x, y: integer;
  selesai;
  Matrix = larik[1..3, 1..3] dari real;
variabel
  p: Point;
  m: Matrix;
mulai
  p.x := 10;
  p.y := 20;
  m[1,1] := 1.5;
selesai.
```

Output 2:

```
=== TOKENS ===
```

```
  0: KEYWORD              'program' at 1:1
  1: IDENTIFIER           'Input3' at 1:9
  2: SEMICOLON            ';' at 1:15
  3: KEYWORD              'tipe' at 2:1
  4: IDENTIFIER           'Point' at 3:3
  5: RELATIONAL_OPERATOR  '=' at 3:9
  6: KEYWORD              'rekaman' at 3:11
  7: IDENTIFIER           'x' at 4:5
  8: COMMA                ',' at 4:6
  9: IDENTIFIER           'y' at 4:8
 10: COLON                ':' at 4:9
 11: KEYWORD              'integer' at 4:11
 12: SEMICOLON            ';' at 4:18
 13: KEYWORD              'selesai' at 5:3
 14: SEMICOLON            ';' at 5:10
 15: IDENTIFIER           'Matrix' at 6:3
 16: RELATIONAL_OPERATOR  '=' at 6:10
 17: KEYWORD              'larik' at 6:12
 18: LBRACKET             '[' at 6:17
 19: NUMBER               '1' at 6:18
 20: RANGE_OPERATOR       '..' at 6:19
 21: NUMBER               '3' at 6:21
 22: COMMA                ',' at 6:22
 23: NUMBER               '1' at 6:24
 24: RANGE_OPERATOR       '..' at 6:25
 25: NUMBER               '3' at 6:27
 26: RBRACKET             ']' at 6:28
 27: KEYWORD              'dari' at 6:30
 28: KEYWORD              'real' at 6:35
 29: SEMICOLON            ';' at 6:39
 30: KEYWORD              'variabel' at 7:1
 31: IDENTIFIER           'p' at 8:3
 32: COLON                ':' at 8:4
 33: IDENTIFIER           'Point' at 8:6
 34: SEMICOLON            ';' at 8:11
 35: IDENTIFIER           'm' at 9:3
 36: COLON                ':' at 9:4
 37: IDENTIFIER           'Matrix' at 9:6
 38: SEMICOLON            ';' at 9:12
 39: KEYWORD              'mulai' at 10:1
 40: IDENTIFIER           'p' at 11:3
 41: DOT                  '.' at 11:4
 42: IDENTIFIER           'x' at 11:5
 43: ASSIGN_OPERATOR      ':=' at 11:7
 44: NUMBER               '10' at 11:10
 45: SEMICOLON            ';' at 11:12
 46: IDENTIFIER           'p' at 12:3
 47: DOT                  '.' at 12:4
 48: IDENTIFIER           'y' at 12:5
 49: ASSIGN_OPERATOR      ':=' at 12:7
 50: NUMBER               '20' at 12:10
 51: SEMICOLON            ';' at 12:12
 52: IDENTIFIER           'm' at 13:3
 53: LBRACKET             '[' at 13:4
 54: NUMBER               '1' at 13:5
 55: COMMA                ',' at 13:6
 56: NUMBER               '1' at 13:7
 57: RBRACKET             ']' at 13:8
 58: ASSIGN_OPERATOR      ':=' at 13:10
 59: NUMBER               '1.5' at 13:13
 60: SEMICOLON            ';' at 13:16
 61: KEYWORD              'selesai' at 14:1
 62: DOT                  '.' at 14:8
==============
<program>
├── <program-header>
│   ├── KEYWORD(program)
```

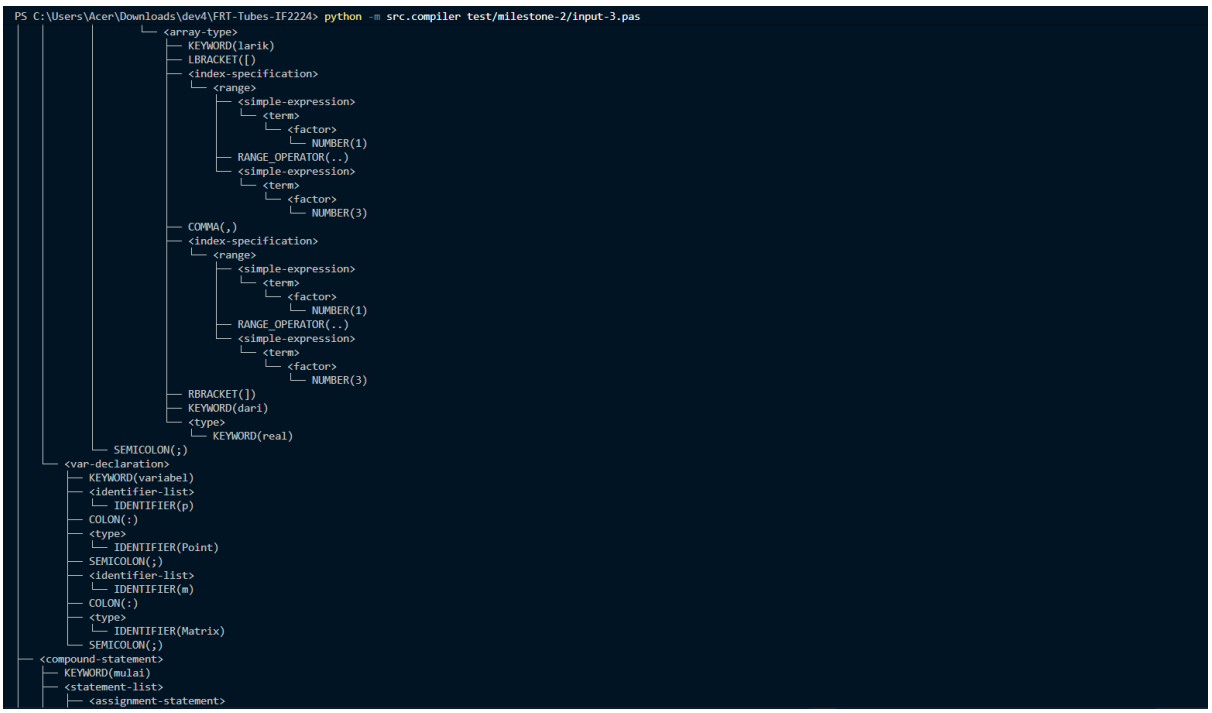

IDENTIFIER(Input3)
SEMICOLON(;)
<declaration-part>
<type-declaration>
KEYWORD(tipe)
<type-item>
IDENTIFIER(Point)
RELATIONAL_OPERATOR(=)
<type-definition>
<type>
<record-type>
KEYWORD(rekaman)
<field-list>
<var-item>
<identifier-list>
IDENTIFIER(x)
COMMA(,)
IDENTIFIER(y)
COLON(:)
<type>
KEYWORD(integer)
SEMICOLON(;)
KEYWORD(selesai)
SEMICOLON(;)
<type-item>
IDENTIFIER(Matrix)
RELATIONAL_OPERATOR(=)
<type-definition>
<type>
<array-type>
KEYWORD(larik)
LBRACKET([)
<index-specification>
<range>
<simple-expression>
<term>
<factor>
NUMBER(1)
RANGE_OPERATOR(..)
<simple-expression>
<term>
<factor>
NUMBER(3)
COMMA(,)
<index-specification>
<range>
<simple-expression>
<term>
<factor>
NUMBER(1)
RANGE_OPERATOR(..)
<simple-expression>
<term>
<factor>
NUMBER(3)
RBRACKET(])
KEYWORD(dari)
<type>
KEYWORD(real)
SEMICOLON(;)
<var-declaration>
KEYWORD(variabel)
<identifier-list>
IDENTIFIER(p)
COLON(:)
<type>
IDENTIFIER(Point)

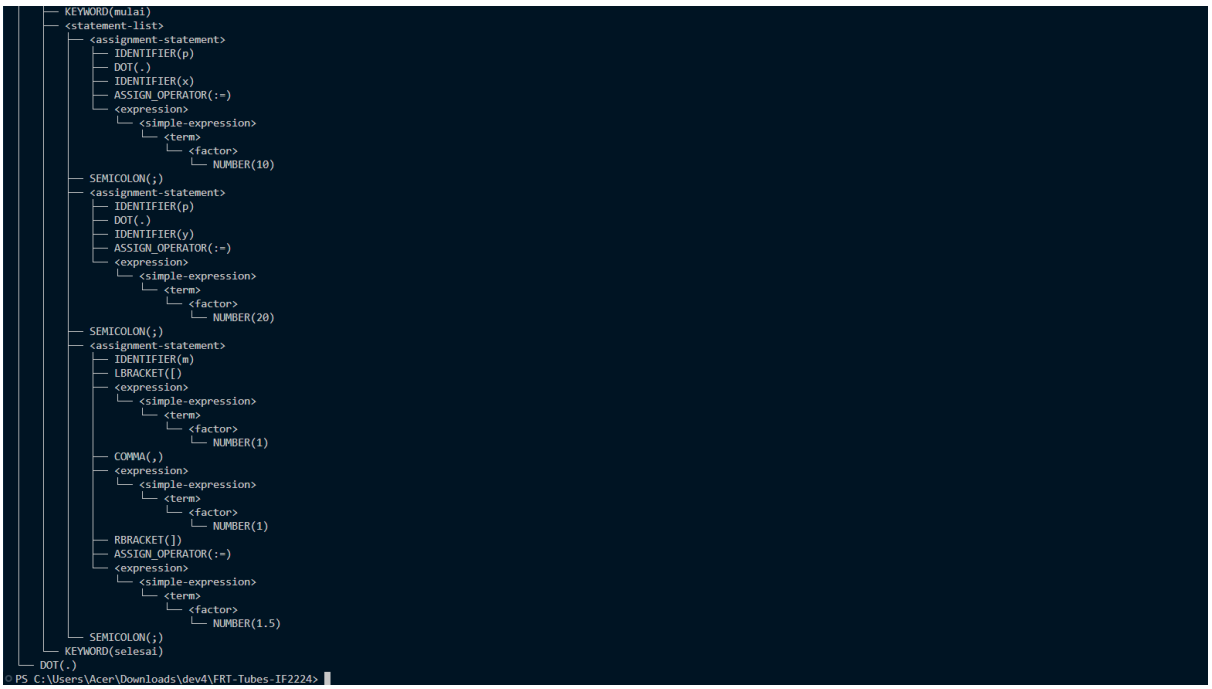


Screenshot 2:

```
PS C:\Users\Acer\Downloads\dev4\FRT-Tubes-IF2224> python -m src.compiler test/milestone-2/input-3.pas
=== TOKENS ===
  0: KEYWORD             'program' at 1:1
  1: IDENTIFIER          'Input3' at 1:9
  2: SEMICOLON           ';' at 1:15
  3: KEYWORD             'tipe' at 2:1
  4: IDENTIFIER          'Point' at 3:3
  5: RELATIONAL_OPERATOR '=' at 3:9
  6: KEYWORD             'rekaman' at 3:11
  7: IDENTIFIER          'x' at 4:5
  8: COMMA               ',' at 4:6
  9: IDENTIFIER          'y' at 4:8
 10: COLON               ':' at 4:9
 11: KEYWORD             'integer' at 4:11
 12: SEMICOLON           ';' at 4:18
 13: KEYWORD             'selesai' at 5:3
 14: SEMICOLON           ';' at 5:10
 15: IDENTIFIER          'Matrix' at 6:3
 16: RELATIONAL_OPERATOR '=' at 6:10
 17: KEYWORD             'larik' at 6:12
 18: LBRACKET            '[' at 6:17
 19: NUMBER              '1' at 6:18
 20: RANGE_OPERATOR      '..' at 6:19
 21: NUMBER              '3' at 6:21
 22: COMMA               ',' at 6:22
 23: NUMBER              '1' at 6:24
 24: RANGE_OPERATOR      '..' at 6:25
 25: NUMBER              '3' at 6:27
 26: RBRACKET            ']' at 6:28
 27: KEYWORD             'dari' at 6:30
 28: KEYWORD             'real' at 6:35
 29: SEMICOLON           ';' at 6:39
 30: KEYWORD             'variabel' at 7:1
 31: IDENTIFIER          'p' at 8:3
 32: COLON               ':' at 8:4
 33: IDENTIFIER          'Point' at 8:6
 34: SEMICOLON           ';' at 8:11
 35: IDENTIFIER          'm' at 9:3
 36: COLON               ':' at 9:4
 37: IDENTIFIER          'Matrix' at 9:6
 38: SEMICOLON           ';' at 9:12
 39: KEYWORD             'mulai' at 10:1
 40: IDENTIFIER          'p' at 11:3
 41: DOT                 '.' at 11:4
 42: IDENTIFIER          'x' at 11:5
 43: ASSIGN_OPERATOR     ':=' at 11:7
 44: NUMBER              '10' at 11:10
 45: SEMICOLON           ';' at 11:12
 46: IDENTIFIER          'p' at 12:3
 47: DOT                 '.' at 12:4
```

```
PS C:\Users\Acer\Downloads\dev4\FRT-Tubes-IF2224> python -m src.compiler test/milestone-2/input-3.pas
 48: IDENTIFIER          'y' at 12:5
 49: ASSIGN_OPERATOR     ':=' at 12:7
 50: NUMBER              '20' at 12:10
 51: SEMICOLON           ';' at 12:12
 52: IDENTIFIER          'm' at 13:3
 53: LBRACKET            '[' at 13:4
 54: NUMBER              '1' at 13:5
 55: COMMA               ',' at 13:6
 56: NUMBER              '1' at 13:7
 57: RBRACKET            ']' at 13:8
 58: ASSIGN_OPERATOR     ':=' at 13:10
 59: NUMBER              '1.5' at 13:13
 60: SEMICOLON           ';' at 13:16
 61: KEYWORD             'selesai' at 14:1
 62: DOT                 '.' at 14:8
==============
<program>
├── <program-header>
│   ├── KEYWORD(program)
│   ├── IDENTIFIER(Input3)
│   └── SEMICOLON(;)
├── <declaration-part>
│   ├── <type-declaration>
│   │   ├── KEYWORD(tipe)
│   │   ├── <type-item>
│   │   │   ├── IDENTIFIER(Point)
│   │   │   ├── RELATIONAL_OPERATOR(=)
│   │   │   ├── <type-definition>
│   │   │   │   └── <type>
│   │   │   │       └── <record-type>
│   │   │   │           ├── KEYWORD(rekaman)
│   │   │   │           ├── <field-list>
│   │   │   │           │   └── <var-item>
│   │   │   │           │       ├── <identifier-list>
│   │   │   │           │       │   ├── IDENTIFIER(x)
│   │   │   │           │       │   ├── COMMA(,)
│   │   │   │           │       │   └── IDENTIFIER(y)
│   │   │   │           │       ├── COLON(:)
│   │   │   │           │       ├── <type>
│   │   │   │           │       │   └── KEYWORD(integer)
│   │   │   │           │       └── SEMICOLON(;)
│   │   │   │           └── KEYWORD(selesai)
│   │   └── SEMICOLON(;)
│   └── <type-item>
│       ├── IDENTIFIER(Matrix)
│       ├── RELATIONAL_OPERATOR(=)
│       ├── <type-definition>
│       │   └── <type>
│       │       └── <array-type>
```





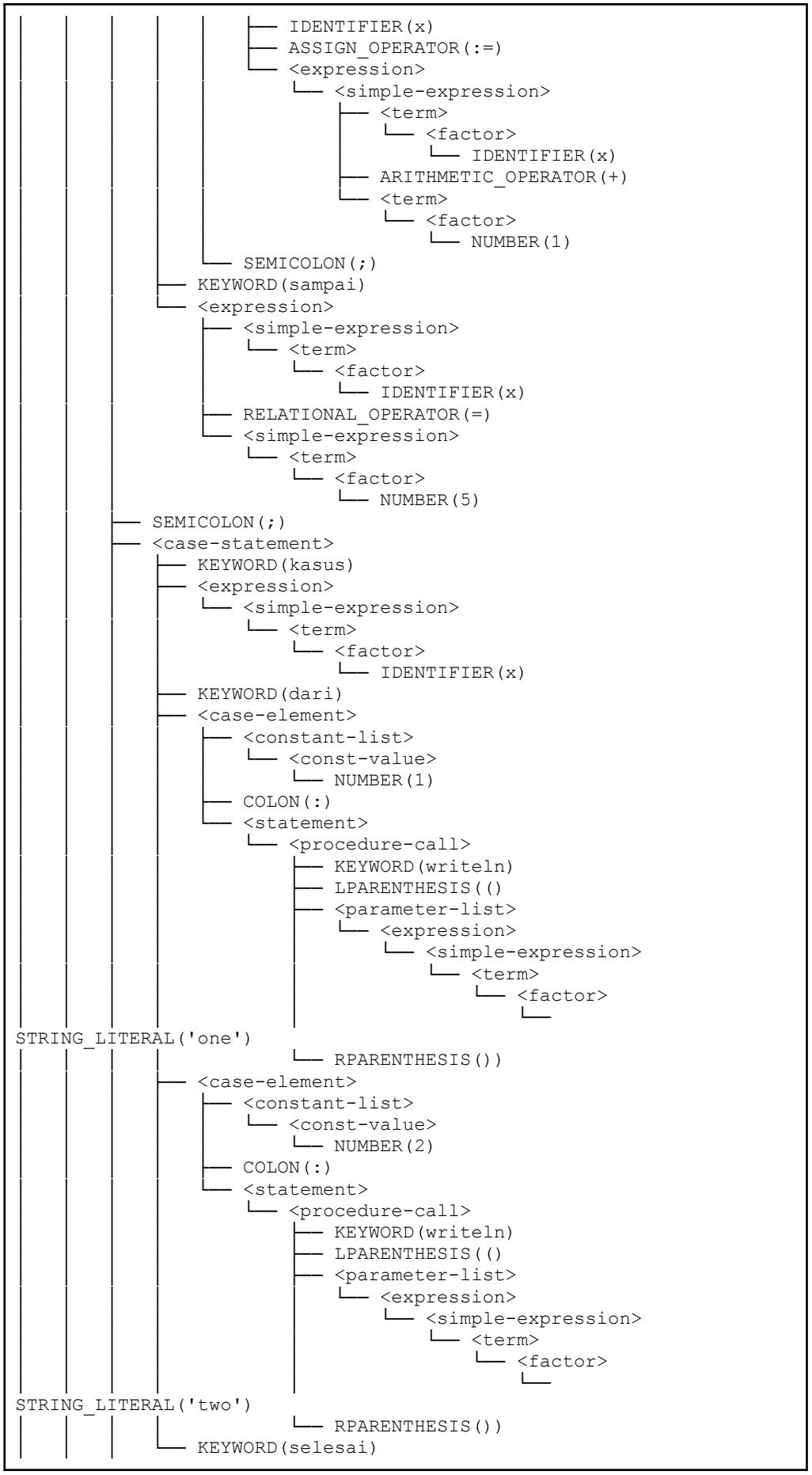
3.3 Kasus Uji 3

Input 3:

```
program Input12;
variabel x: integer;
mulai
  ulangi
    x := x + 1;
  sampai x = 5;
  kasus x dari
    1: writeln('one');
    2: writeln('two');
  selesai;
selesai.
```

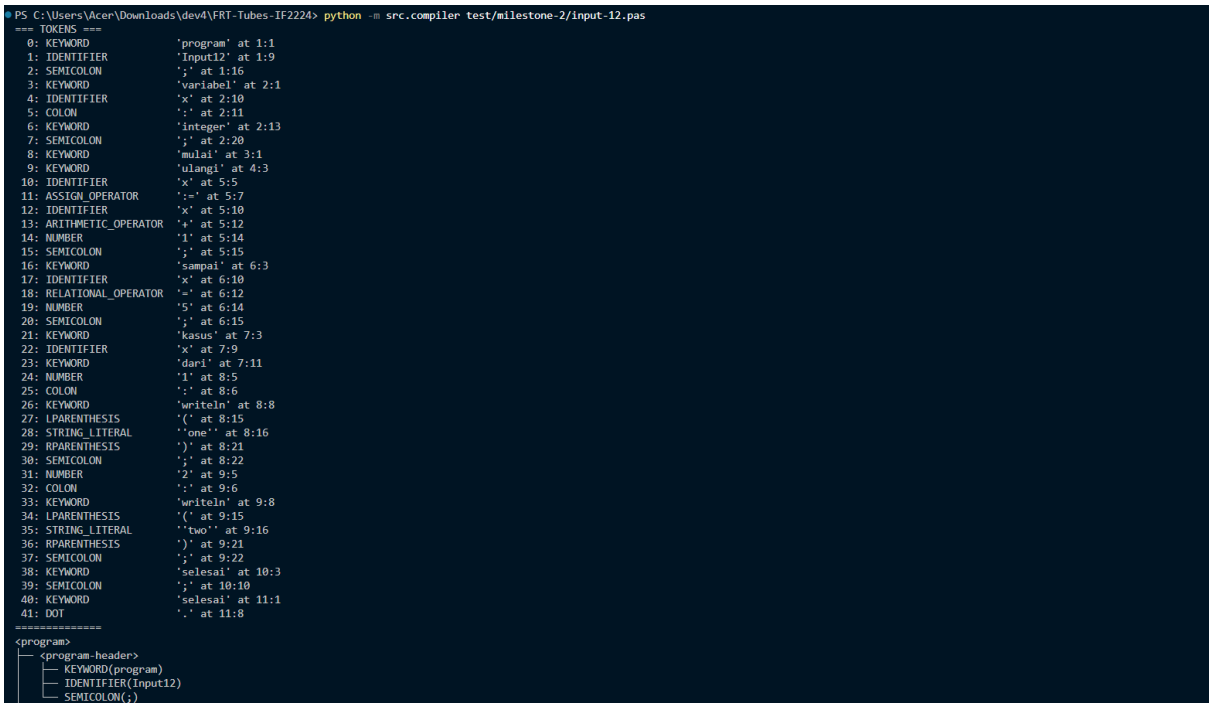
Output 3:

```
=== TOKENS ===
  0: KEYWORD              'program' at 1:1
  1: IDENTIFIER           'Input12' at 1:9
  2: SEMICOLON            ';' at 1:16
  3: KEYWORD              'variabel' at 2:1
  4: IDENTIFIER           'x' at 2:10
  5: COLON                ':' at 2:11
  6: KEYWORD              'integer' at 2:13
  7: SEMICOLON            ';' at 2:20
  8: KEYWORD              'mulai' at 3:1
  9: KEYWORD              'ulangi' at 4:3
 10: IDENTIFIER           'x' at 5:5
 11: ASSIGN_OPERATOR      ':=' at 5:7
 12: IDENTIFIER           'x' at 5:10
 13: ARITHMETIC_OPERATOR  '+' at 5:12
 14: NUMBER               '1' at 5:14
 15: SEMICOLON            ';' at 5:15
 16: KEYWORD              'sampai' at 6:3
 17: IDENTIFIER           'x' at 6:10
 18: RELATIONAL_OPERATOR  '=' at 6:12
 19: NUMBER               '5' at 6:14
 20: SEMICOLON            ';' at 6:15
 21: KEYWORD              'kasus' at 7:3
 22: IDENTIFIER           'x' at 7:9
 23: KEYWORD              'dari' at 7:11
 24: NUMBER               '1' at 8:5
 25: COLON                ':' at 8:6
 26: KEYWORD              'writeln' at 8:8
 27: LPARENTHESIS         '(' at 8:15
 28: STRING_LITERAL       ''one'' at 8:16
 29: RPARENTHESIS         ')' at 8:21
 30: SEMICOLON            ';' at 8:22
 31: NUMBER               '2' at 9:5
 32: COLON                ':' at 9:6
 33: KEYWORD              'writeln' at 9:8
 34: LPARENTHESIS         '(' at 9:15
 35: STRING_LITERAL       ''two'' at 9:16
 36: RPARENTHESIS         ')' at 9:21
 37: SEMICOLON            ';' at 9:22
 38: KEYWORD              'selesai' at 10:3
 39: SEMICOLON            ';' at 10:10
 40: KEYWORD              'selesai' at 11:1
 41: DOT                  '.' at 11:8
==============
<program>
├── <program-header>
│   ├── KEYWORD(program)
│   ├── IDENTIFIER(Input12)
│   └── SEMICOLON(;)
├── <declaration-part>
│   └── <var-declaration>
│       ├── KEYWORD(variabel)
│       ├── <identifier-list>
│       │   └── IDENTIFIER(x)
│       ├── COLON(:)
│       ├── <type>
│       │   └── KEYWORD(integer)
│       └── SEMICOLON(;)
├── <compound-statement>
│   ├── KEYWORD(mulai)
│   ├── <statement-list>
│   │   ├── <repeat-statement>
│   │   │   ├── KEYWORD(ulangi)
│   │   │   ├── <statement-list>
│   │   │   │   ├── <assignment-statement>
```

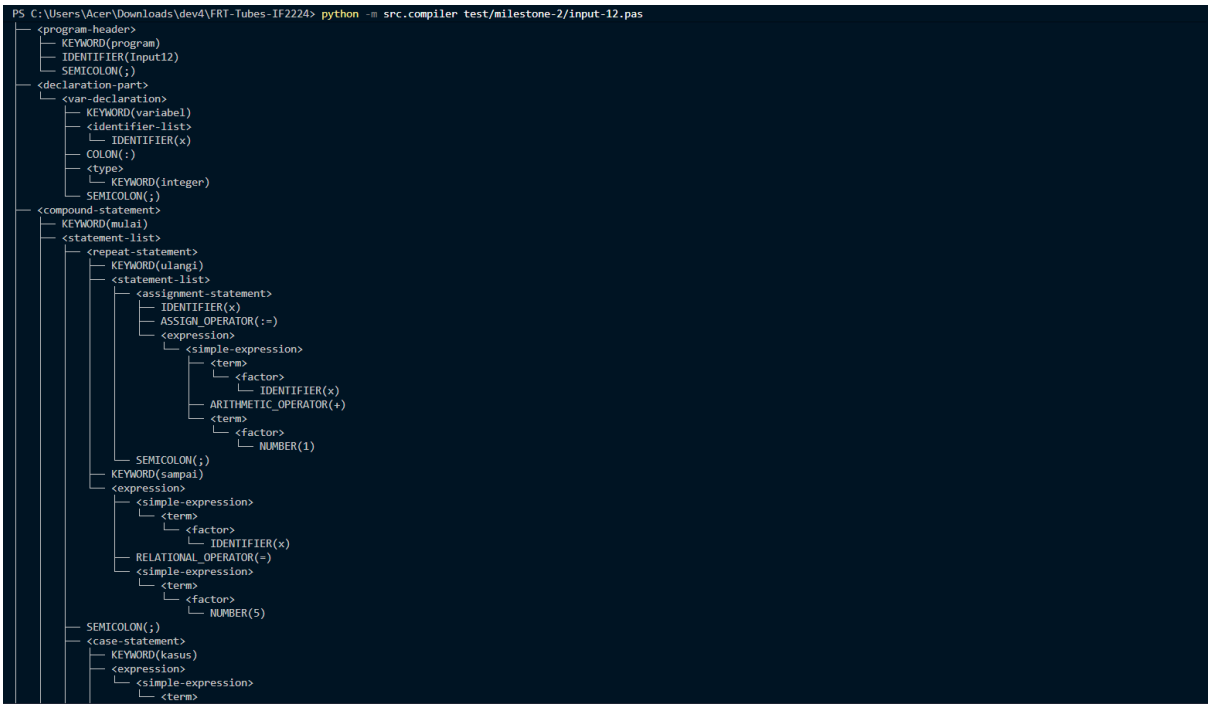

IDENTIFIER(x)
ASSIGN_OPERATOR(:=)
<expression>
<simple-expression>
<term>
<factor>
IDENTIFIER(x)
ARITHMETIC_OPERATOR(+)
<term>
<factor>
NUMBER(1)
SEMICOLON(;)
KEYWORD(sampai)
<expression>
<simple-expression>
<term>
<factor>
IDENTIFIER(x)
RELATIONAL_OPERATOR(=)
<simple-expression>
<term>
<factor>
NUMBER(5)
SEMICOLON(;)
<case-statement>
KEYWORD(kasus)
<expression>
<simple-expression>
<term>
<factor>
IDENTIFIER(x)
KEYWORD(dari)
<case-element>
<constant-list>
<const-value>
NUMBER(1)
COLON(:)
<statement>
<procedure-call>
KEYWORD(writeln)
LPARENTHESIS(()
<parameter-list>
<expression>
<simple-expression>
<term>
<factor>
STRING_LITERAL('one')
RPARENTHESIS())
<case-element>
<constant-list>
<const-value>
NUMBER(2)
COLON(:)
<statement>
<procedure-call>
KEYWORD(writeln)
LPARENTHESIS(()
<parameter-list>
<expression>
<simple-expression>
<term>
<factor>
STRING_LITERAL('two')
RPARENTHESIS())
KEYWORD(selesai)

```
│   │   └── SEMICOLON(;)
│   └── KEYWORD(selesai)
└── DOT(.)
```

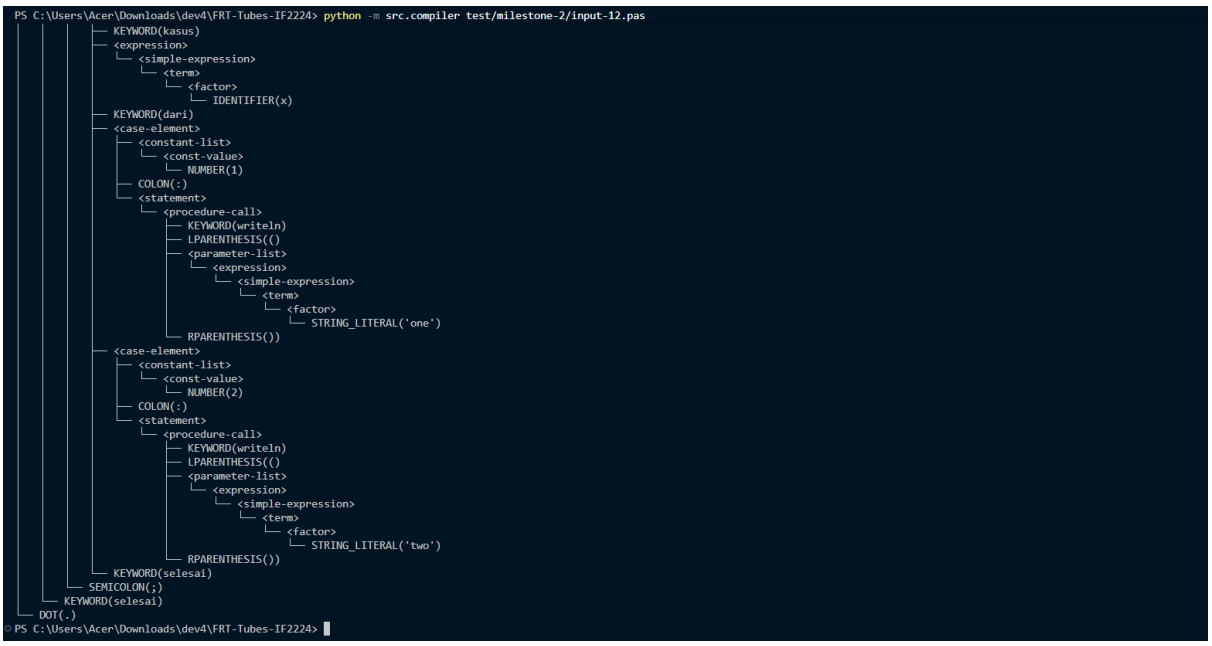
Screenshot 3:



```
PS C:\Users\Acer\Downloads\dev4\FRT-Tubes-IF2224> python -m src.compiler test/milestone-2/input-12.pas
=== TOKENS ===
  0: KEYWORD              'program' at 1:1
  1: IDENTIFIER           'Input12' at 1:9
  2: SEMICOLON            ';' at 1:16
  3: KEYWORD              'variabel' at 2:1
  4: IDENTIFIER           'x' at 2:10
  5: COLON                ':' at 2:11
  6: KEYWORD              'integer' at 2:13
  7: SEMICOLON            ';' at 2:20
  8: KEYWORD              'mulai' at 3:1
  9: KEYWORD              'ulangi' at 4:3
 10: IDENTIFIER           'x' at 5:5
 11: ASSIGN_OPERATOR      ':=' at 5:7
 12: IDENTIFIER           'x' at 5:10
 13: ARITHMETIC_OPERATOR  '+' at 5:12
 14: NUMBER               '1' at 5:14
 15: SEMICOLON            ';' at 5:15
 16: KEYWORD              'sampai' at 6:3
 17: IDENTIFIER           'x' at 6:10
 18: RELATIONAL_OPERATOR  '=' at 6:12
 19: NUMBER               '5' at 6:14
 20: SEMICOLON            ';' at 6:15
 21: KEYWORD              'kasus' at 7:3
 22: IDENTIFIER           'x' at 7:9
 23: KEYWORD              'dari' at 7:11
 24: NUMBER               '1' at 8:5
 25: COLON                ':' at 8:6
 26: KEYWORD              'writeln' at 8:8
 27: LPARENTHESIS         '(' at 8:15
 28: STRING_LITERAL       ''one'' at 8:16
 29: RPARENTHESIS         ')' at 8:21
 30: SEMICOLON            ';' at 8:22
 31: NUMBER               '2' at 9:5
 32: COLON                ':' at 9:6
 33: KEYWORD              'writeln' at 9:8
 34: LPARENTHESIS         '(' at 9:15
 35: STRING_LITERAL       ''two'' at 9:16
 36: RPARENTHESIS         ')' at 9:21
 37: SEMICOLON            ';' at 9:22
 38: KEYWORD              'selesai' at 10:3
 39: SEMICOLON            ';' at 10:10
 40: KEYWORD              'selesai' at 11:1
 41: DOT                  '.' at 11:8
==============
<program>
├── <program-header>
│   ├── KEYWORD(program)
│   ├── IDENTIFIER(Input12)
│   └── SEMICOLON(;)
```



```
PS C:\Users\Acer\Downloads\dev4\FRT-Tubes-IF2224> python -m src.compiler test/milestone-2/input-12.pas
├── <program-header>
│   ├── KEYWORD(program)
│   ├── IDENTIFIER(Input12)
│   └── SEMICOLON(;)
├── <declaration-part>
│   └── <var-declaration>
│       ├── KEYWORD(variabel)
│       ├── <identifier-list>
│       │   └── IDENTIFIER(x)
│       ├── COLON(:)
│       ├── <type>
│       │   └── KEYWORD(integer)
│       └── SEMICOLON(;)
├── <compound-statement>
│   ├── KEYWORD(mulai)
│   ├── <statement-list>
│   │   ├── <repeat-statement>
│   │   │   ├── KEYWORD(ulangi)
│   │   │   ├── <statement-list>
│   │   │   │   ├── <assignment-statement>
│   │   │   │   │   ├── IDENTIFIER(x)
│   │   │   │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │   │   │   └── <expression>
│   │   │   │   │       └── <simple-expression>
│   │   │   │   │           ├── <term>
│   │   │   │   │           │   └── <factor>
│   │   │   │   │           │       └── IDENTIFIER(x)
│   │   │   │   │           ├── ARITHMETIC_OPERATOR(+)
│   │   │   │   │           └── <term>
│   │   │   │   │               └── <factor>
│   │   │   │   │                   └── NUMBER(1)
│   │   │   │   └── SEMICOLON(;)
│   │   │   ├── KEYWORD(sampai)
│   │   │   └── <expression>
│   │   │       ├── <simple-expression>
│   │   │       │   └── <term>
│   │   │       │       └── <factor>
│   │   │       │           └── IDENTIFIER(x)
│   │   │       ├── RELATIONAL_OPERATOR(=)
│   │   │       └── <simple-expression>
│   │   │           └── <term>
│   │   │               └── <factor>
│   │   │                   └── NUMBER(5)
│   │   ├── SEMICOLON(;)
│   │   ├── <case-statement>
│   │   │   ├── KEYWORD(kasus)
│   │   │   ├── <expression>
│   │   │   │   └── <simple-expression>
│   │   │   │       └── <term>
```



3.4 Kasus Uji 4

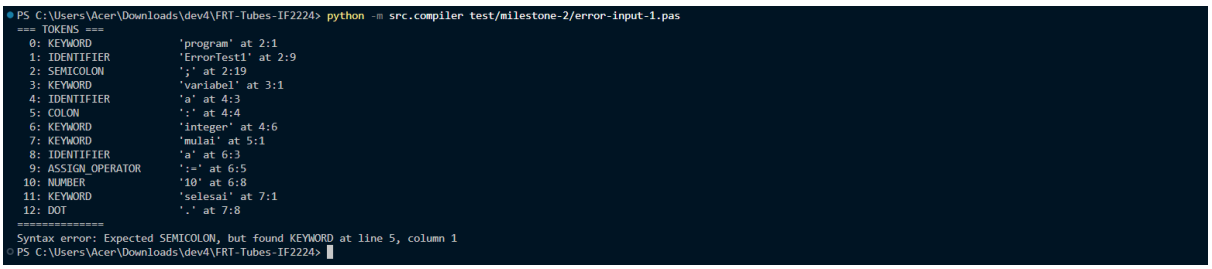
Input 4:

```
{Missing semicolon}
program ErrorTest1;
variabel
  a: integer
mulai
  a := 10
selesai.
```

Output 4:

```
=== TOKENS ===
  0: KEYWORD              'program' at 2:1
  1: IDENTIFIER           'ErrorTest1' at 2:9
  2: SEMICOLON            ';' at 2:19
  3: KEYWORD              'variabel' at 3:1
  4: IDENTIFIER           'a' at 4:3
  5: COLON                ':' at 4:4
  6: KEYWORD              'integer' at 4:6
  7: KEYWORD              'mulai' at 5:1
  8: IDENTIFIER           'a' at 6:3
  9: ASSIGN_OPERATOR      ':=' at 6:5
 10: NUMBER               '10' at 6:8
 11: KEYWORD              'selesai' at 7:1
 12: DOT                  '.' at 7:8
==============
Syntax error: Expected SEMICOLON, but found KEYWORD at line 5,
column 1
```

Screenshot 4:



3.5 Kasus Uji 5

Input 5:

```
{Invalid expression}
program ErrorTest6;
variabel
  a, b: integer;
mulai
  a := 10 + * 5;
  b := a;
selesai.
```

Output 5:

```
=== TOKENS ===
  0: KEYWORD              'program' at 2:1
  1: IDENTIFIER           'ErrorTest6' at 2:9
  2: SEMICOLON            ';' at 2:19
  3: KEYWORD              'variabel' at 3:1
  4: IDENTIFIER           'a' at 4:3
  5: COMMA                ',' at 4:4
  6: IDENTIFIER           'b' at 4:6
  7: COLON                ':' at 4:7
  8: KEYWORD              'integer' at 4:9
  9: SEMICOLON            ';' at 4:16
 10: KEYWORD              'mulai' at 5:1
 11: IDENTIFIER           'a' at 6:3
 12: ASSIGN_OPERATOR      ':=' at 6:5
 13: NUMBER               '10' at 6:8
 14: ARITHMETIC_OPERATOR  '+' at 6:11
 15: ARITHMETIC_OPERATOR  '*' at 6:13
 16: NUMBER               '5' at 6:15
 17: SEMICOLON            ';' at 6:16
 18: IDENTIFIER           'b' at 7:3
 19: ASSIGN_OPERATOR      ':=' at 7:5
 20: IDENTIFIER           'a' at 7:8
 21: SEMICOLON            ';' at 7:9
 22: KEYWORD              'selesai' at 8:1
 23: DOT                  '.' at 8:8
==============
Syntax error: Unexpected token ARITHMETIC_OPERATOR('*') in
<factor>
```

Screenshot 5:

```
PS C:\Users\Acer\Downloads\dev4\FRT-Tubes-IF2224> python -m src.compiler test/milestone-2/error-input-6.pas
=== TOKENS ===
  0: KEYWORD              'program' at 2:1
  1: IDENTIFIER           'ErrorTest6' at 2:9
  2: SEMICOLON            ';' at 2:19
  3: KEYWORD              'variabel' at 3:1
  4: IDENTIFIER           'a' at 4:3
  5: COMMA                ',' at 4:4
  6: IDENTIFIER           'b' at 4:6
  7: COLON                ':' at 4:7
  8: KEYWORD              'integer' at 4:9
  9: SEMICOLON            ';' at 4:16
 10: KEYWORD              'mulai' at 5:1
 11: IDENTIFIER           'a' at 6:3
 12: ASSIGN_OPERATOR      ':=' at 6:5
 13: NUMBER               '10' at 6:8
 14: ARITHMETIC_OPERATOR  '+' at 6:11
 15: ARITHMETIC_OPERATOR  '*' at 6:13
 16: NUMBER               '5' at 6:15
 17: SEMICOLON            ';' at 6:16
 18: IDENTIFIER           'b' at 7:3
 19: ASSIGN_OPERATOR      ':=' at 7:5
 20: IDENTIFIER           'a' at 7:8
 21: SEMICOLON            ';' at 7:9
 22: KEYWORD              'selesai' at 8:1
 23: DOT                  '.' at 8:8
==============
Syntax error: Unexpected token ARITHMETIC_OPERATOR('*') in <factor>
PS C:\Users\Acer\Downloads\dev4\FRT-Tubes-IF2224>
```

BAB 4
KESIMPULAN DAN SARAN

4.1 Kesimpulan

- Syntax Analysis (parser) merupakan tahap kedua proses kompilasi yang mengubah rangkaian token menjadi struktur parse tree menggunakan algoritma Recursive Descent Parsing.
- Desain grammar menjadi dasar utama dalam pembuatan parser karena menentukan bagaimana setiap token dikelompokkan, divalidasi, dan dibentuk menjadi struktur hierarkis program yang sesuai dengan aturan bahasa Pascal-S.
- Kemampuan syntax error detection sangat penting dalam parser untuk mengidentifikasi kesalahan struktural seperti token yang hilang, urutan token yang salah, atau pelanggaran aturan grammar.
- Parser membantu tahap semantic analysis dengan menyediakan parse tree yang terstruktur sehingga program Pascal-S dapat dianalisis lebih lanjut dan dijalankan dengan benar.

4.2 Saran

- Grammar dan aturan produksi harus didefinisikan dengan jelas dan konsisten agar proses parsing berjalan akurat.
- Error handling dan recovery mechanism harus ditangani dengan baik agar parser dapat memberikan pesan kesalahan yang informatif dan mudah dipahami.
- Integrasi antara lexer dan parser perlu diperkuat untuk memastikan aliran token yang baik dan deteksi kesalahan yang komprehensif.

LAMPIRAN

Link Release Repository Github:

https://github.com/JethroJNS/FRT-Tubes-IF2224.git

Pembagian Tugas:

NIM	Nama	Persentase Kontribusi
13521117	Maggie Zeta Rosida S	25%
13523037	Buege Mahara Putra	25%
13523041	Hanif Kalyana Aditya	25%
13523081	Jethro Jens Norbert Simatupang	25%

REFERENSI

[1] GeeksforGeeks. 2025. *Recursive Descent Parser*. (https://www.geeksforgeeks.org/compiler-design/recursive-descent-parser/, diakses : 15 November 2025)

[2] GeeksforGeeks. 2025. *Parse Tree in Compiler Design*. (https://www.geeksforgeeks.org/compiler-design/parse-tree-in-compiler-design/, diakses : 16 November 2025)