Laporan Tugas Besar Milestone 3

IF2224 - Teori Bahasa Formal Otomata 2025/2026

Disusun oleh :

Kelompok AutoRejectAtas

Muhammad Syarafi Akmal	13522076
Kenneth Poenadi	13523040
Bob Kunanda	13523086
M. Zahran Ramadhan Ardiana	13523104

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

JL. GANESHA 10, BANDUNG 40132

2025

Daftar Isi

BAB 1
Landasan Teori

1.1. Pengertian Semantic Analysis

Semantic analysis merupakan tahap ketiga dalam proses kompilasi setelah lexical analysis dan syntax analysis. Jika syntax analysis hanya menjamin bahwa program sesuai dengan aturan grammar, maka semantic analysis memastikan bahwa program tersebut bermakna secara logis. Tahap ini memeriksa apakah setiap operasi, deklarasi, pemanggilan fungsi, penggunaan variabel, dan ekspresi dalam program sesuai dengan aturan semantik bahasa Pascal-S. Contoh kesalahan yang hanya dapat ditemukan pada tahap ini adalah penggunaan variabel sebelum deklarasi, operasi yang melibatkan tipe data yang tidak kompatibel, atau pemanggilan prosedur dengan jumlah parameter yang keliru. Dengan demikian, semantic analysis berfungsi sebagai penghubung antara struktur kode sumber dengan arti sebenarnya dari program, sehingga compiler dapat menghasilkan keluaran yang benar pada tahap berikutnya.

1.2. Attributed Grammar

Attributed Grammar (Tata Bahasa Beratribut) adalah landasan formal untuk Analisis Semantik. Ini adalah perpanjangan dari Context-Free Grammar (CFG) yang digunakan oleh parser dengan menambahkan dua komponen utama:

- **Atribut**: Misalnya, mencoba melakukan operasi aritmatika pada dua tipe yang berbeda atau menetapkan nilai boolean ke variabel integer.
- **Aturan Semantik**: Mencoba menggunakan variabel, fungsi, atau prosedur yang tidak ditemukan dalam Symbol Table pada scope yang berlaku.

Dalam konteks Analisis Semantik, Attributed Grammar digunakan untuk secara formal mendefinisikan dan memvalidasi makna dari sebuah program. Aturan semantik inilah yang diimplementasikan di dalam fungsi Visit pada AST.

Pada kode ini, ada Entry Point fungsi visit_BinOpNode(...):

```
def visit_BinOpNode(self, node: BinOpNode) -> DataType:
    # Hitung tipe operand kiri (T1)
    left_type = self.visit(node.left)

    # Hitung tipe operand kanan (T2)
    right_type = self.visit(node.right)

    # Terapkan aturan semantik untuk menghitung tipe hasil
    result_type = self._compute_binop_type(
        node.operator,
        left_type,
        right_type,
        node
    )

    # Simpan tipe hasil ke node
    node.computed_type = result_type
    return result_type
```

Lalu, terdapat aturan semantik penjumlahan pada fungsi _compute_binop_type(...):

```
def _compute_binop_type(self, operator, left_type, right_type,
node):
    if op in ['+', '-', '*']:
        # Cek apakah operand numerik
        if left_type in [INTEGER, REAL] and right_type in
[INTEGER, REAL]:

            # ATURAN: Jika salah satu REAL, hasil REAL
            if left_type == REAL or right_type == REAL:
                return DataType.REAL  # INTEGER + REAL → REAL

            # ATURAN: Jika keduanya INTEGER, hasil INTEGER
            return DataType.INTEGER  # INTEGER + INTEGER →
INTEGER

        # TYPE MISMATCH: operand bukan numerik
        self.add_error(f"Invalid operand types for
'{operator}'", node)
        return None  # ERROR
```

1.3. Abstract Syntax Tree (AST) Teranotasi

Abstract Syntax Tree (AST) merupakan representasi hierarkis program yang lebih ringkas dibandingkan Parse Tree. Pembentukannya menggunakan Syntax-Directed Translation Scheme (SDTS), di mana setiap production rule memiliki semantic action yang terkait. Tindakan semantik ini dijalankan pada node Parse Tree untuk membentuk node baru pada AST. AST fokus pada struktur logis program dan menghilangkan node yang bersifat seremonial, seperti tanda kurung dan titik koma, untuk menyederhanakan proses analisis semantik berikutnya.

Anotasi pada AST (Decorated AST)

Setelah AST terbentuk, proses Pengecekan Tipe & Lingkup dilakukan dengan menelusuri setiap node AST secara Depth First. Selama penelusuran ini, AST akan dianotasi atau didekorasi dengan informasi semantik yang relevan. Node pada Decorated AST minimal harus memiliki informasi penting berikut:

- Tipe Ekspresi (*e.g.*, *integer*, *boolean*)
- Referensi ke *Symbol Table* (indeks ke entri Symbol Table)
- Informasi Lingkup (*Scope*)

1.4. Symbol Table

Symbol Table adalah struktur data penting yang minimal menyimpan informasi nama identifier, tipe datanya, dan scope-nya. Dalam konteks kompilasi, Symbol Table menggunakan struktur data stack untuk manajemen lingkup.

Fungsi dan Mekanisme Lingkup (Scope)

Saat identifier digunakan dalam program (misalnya, dalam sebuah ekspresi), compiler akan melakukan lookup pada Symbol Table. Proses lookup ini akan memeriksa scope saat ini, kemudian mundur ke scope luar (berdasarkan lexical level) sampai identifier ditemukan.

Struktur Tabel dalam Pascal-S

Sesuai dengan spesifikasi Pascal-S yang digunakan, terdapat tiga tabel simbol yang saling terkait:

- **tab** (Identifier Table): Digunakan untuk menyimpan informasi mengenai setiap identifier (konstanta, variabel, prosedur, fungsi). Atribut utamanya meliputi identifiers, link (ke identifier sebelumnya dalam scope), obj (kelas objek: variabel, tipe, dll.), type (tipe dasar), lev (tingkat lexical level), dan adr (offset alamat).
- **btab** (Block Table): Menyimpan informasi blok prosedur, fungsi, atau definisi tipe record. Atributnya mencakup last (identifier terakhir dalam blok), lpar (parameter terakhir), psze (ukuran parameter), dan vsze (ukuran variabel lokal).
- **atab** (Array Table): Menyimpan informasi spesifik untuk tipe array, seperti xtyp (tipe indeks), etyp (tipe elemen), low dan high (batas indeks), dan size (total ukuran)

1.5. Visitor / Visit Functions

Untuk menelusuri AST dan menjalankan aturan semantik, digunakan pola desain Visitor.

Mekanisme Traversal

Setiap jenis node pada AST memiliki fungsi visit yang terkait:

```
def visit_ProgramNode(self, node: ProgramNode) -> None:
    """visit program node - entry point untuk analysis"""
    # Aksi 1: Akses Symbol Table - masukkan program name
    self.symbol_table.enter_program(node.name)  # Program
identifier

    # Aksi 2: Mengunjungi Anak - visit declarations
    if node.declarations:
        self.visit(node.declarations)

    # Aksi 3: Mengunjungi Anak - visit program body
    if node.body:
        self.visit(node.body)
```

Dan

```
def visit_AssignmentNode(self, node: AssignmentNode) -> None:
    """visit assignment statement - cek types match"""
    # Aksi 1: Mengunjungi Anak - get target type (bisa VarNode
atau ArrayAccessNode)
    target_type = self.visit(node.target)

    # Aksi 2: Mengunjungi Anak - get value type (expression)
    value_type = self.visit(node.value)

    if target_type is None:
        return  # Error already reported

    if value_type is None:
        return  # Error already reported

    # Aksi 3: Kalkulasi Atribut - cek type compatibility
    if not self._types_compatible(target_type, value_type):
        self.add_error(
            f"Type mismatch in assignment: cannot assign
{value_type.value} to {target_type.value}",
            node
        )

    # Aksi 4: Anotasi Node - decorate dengan computed type
    node.computed_type = target_type
```

Analisis Semantik dilakukan dengan menelusuri (traversing) AST secara rekursif, biasanya secara Top-Down dengan urutan Depth First.

```
ProgramNode (visit_ProgramNode)
    ↓
    ├─ DeclarationPartNode
    │    ├─ ConstDeclNode (visit children bottom-up)
    │    ├─ VarDeclNode
    │    └─ ...
    │
    └─ CompoundStatementNode (body)
         └─ AssignmentNode (visit_AssignmentNode)
              ├─ VarNode (target) ← visit dulu (depth-first)
              └─ BinOpNode (value) ← baru visit ini
                   ├─ left expression
                   └─ right expression
```

Semantic Actions

Di dalam setiap fungsi visit, dilakukan empat aksi utama:

1. Mengunjungi Anak: Fungsi visit memanggil fungsi visit untuk node anak-anaknya.

2. Akses Symbol Table: Melakukan lookup untuk identifier yang digunakan atau memasukkan identifier yang dideklarasikan ke dalam Symbol Table.
3. Kalkulasi Atribut: Menghitung atribut/tipe semantik dari node saat ini (misalnya, menentukan tipe ekspresi hasil operasi aritmatika).
4. Anotasi Node: Menambahkan informasi yang dikalkulasi ke node AST (dekorasi).

1.6. Error Handling Semantik

Meskipun Parser menjamin sintaks yang benar, Semantic Analyzer harus mampu mendeteksi dan melaporkan kesalahan semantik. Error handling yang baik akan menghasilkan pesan galat yang informatif.

- Type Mismatch Errors → Ini adalah error yang paling umum, terjadi ketika tipe data tidak cocok dalam operasi. Misalnya saat assignment, sistem akan memeriksa apakah tipe data yang di-assign kompatibel dengan tipe data variabel target. Begitu juga dengan operasi biner seperti penjumlahan atau perbandingan, sistem akan memastikan kedua operand memiliki tipe yang kompatibel. Ada juga pemeriksaan khusus untuk operasi integer seperti bagi dan mod yang hanya menerima operand integer, serta operasi boolean seperti dan dan atau yang hanya bisa digunakan dengan operand boolean.

  Contoh Type Mismatch

```
def visit_AssignmentNode(self, node: AssignmentNode) ->
None:
    target_type = self.visit(node.target)
    value_type = self.visit(node.value)

    # Error: tidak bisa assign tipe berbeda
    if not self._types_compatible(target_type,
value_type):
        self.add_error(
            f"Type mismatch in assignment: cannot assign
{value_type.value} to {target_type.value}",
            node
        )
```

```
def _compute_binop_type(self, operator, left_type,
right_type, node):
    if op in ['+', '-', '*']:
        if left_type in [INTEGER, REAL] and right_type in
[INTEGER, REAL]:
            # OK: numeric types
            return ...
        # Error: invalid operand types
        self.add_error(f"Invalid operand types for
'{operator}'", node)
        return None
```

- Undeclared Identifier Errors → Kategori ini menangkap semua kasus dimana kode mencoba menggunakan identifier yang belum dideklarasikan. Ini berlaku untuk variabel, array, fungsi, prosedur, bahkan tipe custom. Setiap kali ada referensi ke identifier, sistem akan melakukan lookup di symbol table. Kalau tidak ketemu, langsung dilaporkan sebagai error. Ini penting untuk mencegah penggunaan identifier yang salah atau typo.

  Contoh Undeclared Error:

```
def visit_VarNode(self, node: VarNode):
    entry, tab_index =
self.symbol_table.lookup_with_index(node.name)

    if not entry:
        self.add_error(f"Undeclared variable
'{node.name}'", node)
        return None
```

- Duplicate Declaration Errors → Error ini terjadi ketika mencoba mendeklarasikan identifier yang sudah ada di scope yang sama. Sistem memeriksa ini untuk semua jenis deklarasi: konstanta, tipe, variabel, parameter, prosedur, dan fungsi. Sebelum menambahkan entry baru ke symbol table, sistem selalu cek dulu apakah nama tersebut sudah ada di current scope. Ini memastikan setiap identifier di scope tertentu memiliki makna yang unik.

  Contoh Duplicate Error:

```
def visit_ConstDeclNode(self, node: ConstDeclNode):
    existing =
self.symbol_table.lookup_in_current_scope(node.name)
    if existing:
        self.add_error(f"Duplicate declaration of
constant '{node.name}'", node)
```

- Type Constraint Errors → Ini adalah error yang spesifik untuk constraint tipe tertentu dalam struktur bahasa Pascal-S. Contohnya, variabel loop dalam statement for harus bertipe integer, bound values di loop for juga harus integer, index array harus integer, dan range bounds harus integer. Constraint ini memastikan konstruksi bahasa digunakan sesuai dengan spesifikasi Pascal-S.

  Contoh Type Error:

```
def visit_ArrayAccessNode(self, node: ArrayAccessNode):
    if index_type and index_type != DataType.INTEGER:
        self.add_error("Array index must be integer",
node)
```

- Control Flow Errors → Error ini berkaitan dengan statement kontrol seperti if, while, dan repeat. Yang paling penting adalah memastikan kondisi di statement-statement ini selalu menghasilkan nilai boolean. Misalnya, kondisi di statement if atau while tidak boleh berupa ekspresi numerik atau tipe lainnya, harus boolean. Ini memastikan logika program berjalan dengan benar.

  Contoh Control Flow Error:

```
def visit_IfStatementNode(self, node: IfStatementNode):
    if condition_type and condition_type !=
DataType.BOOLEAN:
        self.add_error("If condition must be a boolean
expression", node)
```

- Operator-Specific Errors → Kategori terakhir ini menangani error yang spesifik untuk operator tertentu. Misalnya unary operator + dan - hanya bisa digunakan dengan operand numerik (integer atau real), operator tidak (not) hanya untuk boolean, dan

operator bagi serta mod hanya untuk integer. Setiap operator memiliki aturan tipenya sendiri yang harus dipatuhi.

Contoh Operator-Specific Error:

```
if op == '/':
    if left_type in [INTEGER, REAL] and right_type in
[INTEGER, REAL]:
        return DataType.REAL
    self.add_error(f"Invalid operand types for '/'",
node)
```

BAB 2
PERANCANGAN & IMPLEMENTASI

2.1. Alur Kerja Program

Program compiler ini bekerja dalam 3 tahap utama:

1. Input Processing

    - Program menerima input berupa file .pas (source code Pascal) atau .txt (token list).
    - Jika input .pas, file akan di-tokenize menggunakan lexer.
    - Jika input .txt, file langsung dibaca sebagai daftar token yang sudah ada.
    - Auto-detection encoding dilakukan untuk mendukung UTF-8 dan UTF-16-LE

2. Lexical Analysis (hanya untuk file .pas)

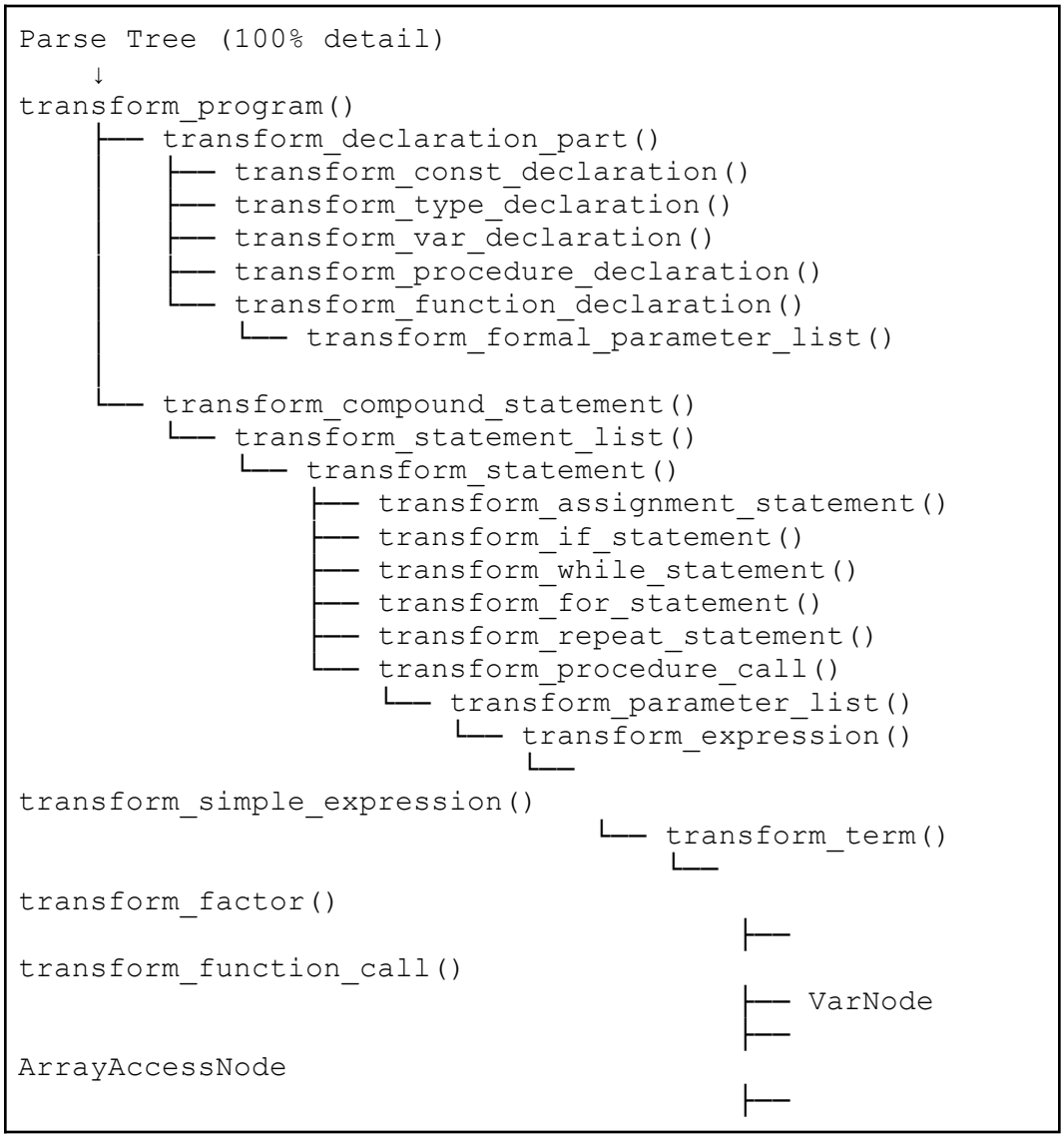
    - Lexer membaca source code karakter per karakter menggunakan DFA (Deterministic Finite Automaton).
    - Comment {...} dan (*...*) di-skip sebelum tokenizing.
    - Token dihasilkan dalam format tuple (type, value).
    - Compound keywords seperti "selain-itu" dan "turun-ke" digabungkan setelah tokenizing.
    - Hasilnya adalah list of tokens: [("KEYWORD", "program"), ("IDENTIFIER", "Hello"), ...].

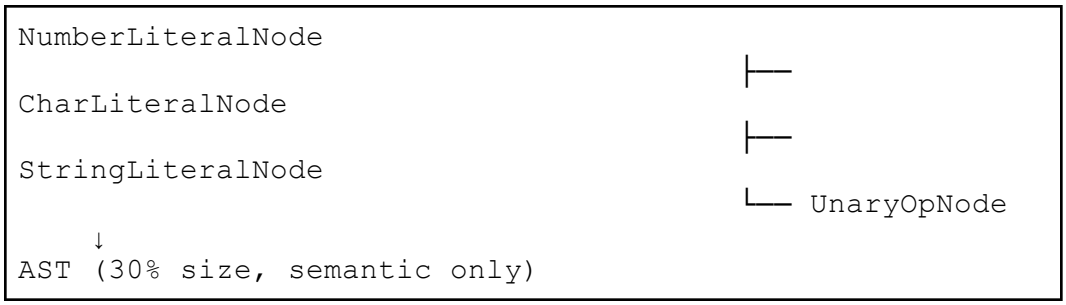
3. Syntax Analysis (Parsing)

    - Parser menerima list of tokens dari lexer.
    - Menggunakan algoritma Recursive Descent untuk membangun parse tree.
    - Setiap aturan grammar diimplementasikan sebagai fungsi terpisah (31 fungsi)
    - Parse tree direpresentasikan sebagai nested dictionary dengan struktur:

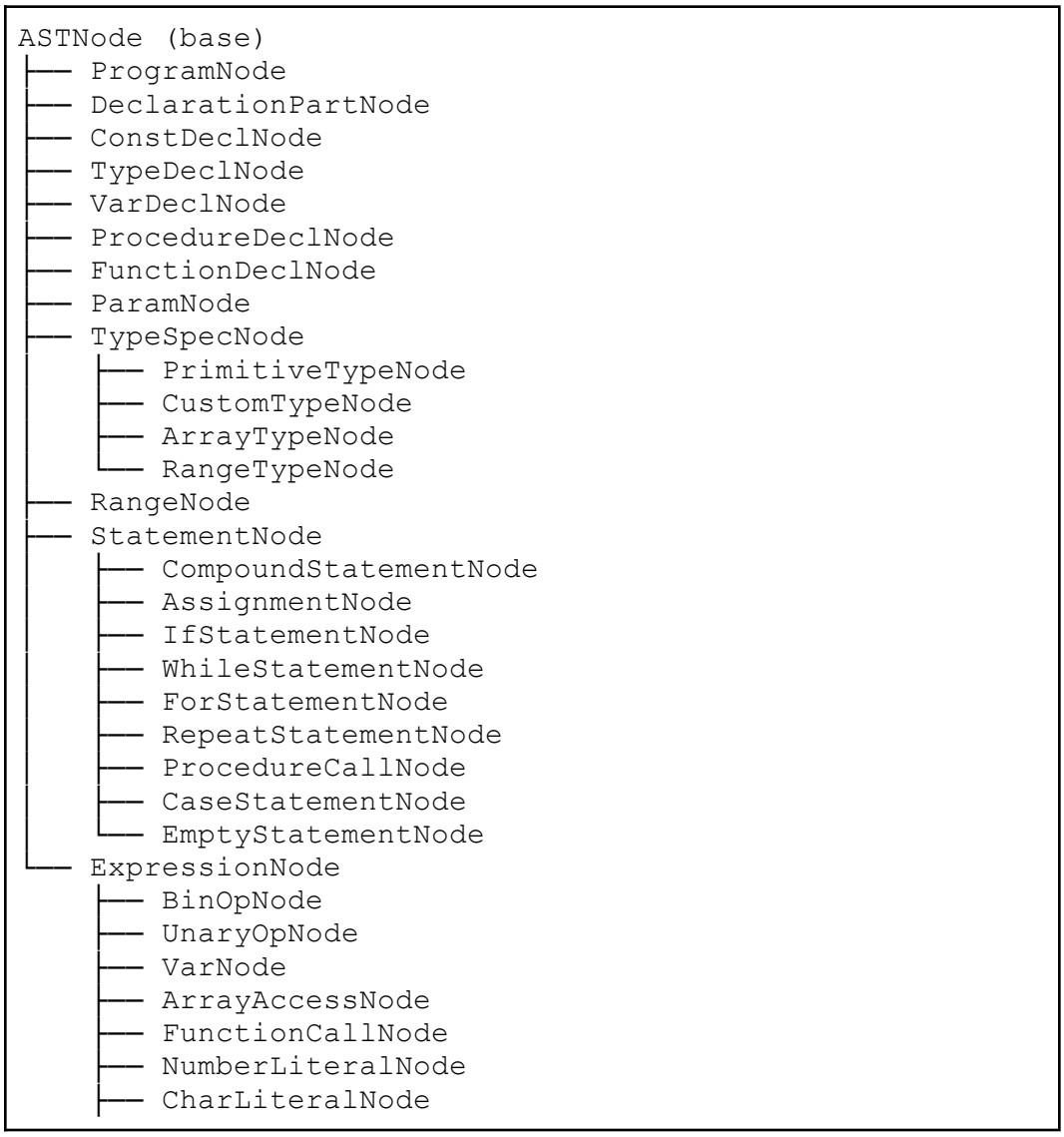
```
{
    "type": "<node-name>",
    "children": [child1, child2, ...]
}
```

4. AST Building (Abstract Syntax Tree)
    - ASTBuilder diinisialisasi
    - Setup error collection untuk semantic errors
    - Traverse parse tree recursively menggunakan transformer methods





- Membuang syntax noise seperti keyword, punctuation, dan structural token.
- Extract semantic information.
- Setiap semantic construct dibuat sebagai typed AST node.



```
    ├── StringLiteralNode
    ├── BooleanLiteralNode
    └── NilLiteralNode
```

5. Semantic Analysis
   - Inisialisasi symbol table
   - Pre-fill Reserved Entries (indices 0-31):

0	program
1	variabel
2	mulai
3	selesai
4	jika
5	maka
6	selain-itu
7	selama
8	lakukan
9	untuk
10	ke
11	turun-ke
12	integer
13	real
14	boolean
15	char
16	larik
17	dari
18	prosedur
19	fungsi

20	konstanta
21	tipe
22	string
23	kasus
24	ulangi
25	sampai
26	rekaman
27	dan
28	atau
29	tidak
30	bagi
31	mod

- Traverse AST menggunakan SemanticVisitor
- Cek tipe, deklarasi, dan scope

6. Output Generation

- AST Printer menerima decorated AST dan symbol table untuk generate output.
- Output ke Terminal.
- Output ke File .txt.

2.2. Struktur Data AST

Implementasi 34 tipe node AST yang merepresentasikan struktur semantik Pascal-S:

Base Class - ASTNode

```
class ASTNode:
    """Base class untuk semua AST nodes"""
    def __init__(self):
        # Attributes untuk decoration (diisi saat semantic
analysis)
```

```
        self.computed_type: Optional[DataType] = None  #
Tipe data hasil
        self.tab_index: Optional[int] = None           #
Index di symbol table
        self.scope_level: Optional[int] = None         #
Level scope
```

Program Structure (2 nodes)

```
ProgramNode(name: str, declarations: DeclarationPartNode,
            body: CompoundStatementNode)
    # Root node program Pascal-S
    # name: nama program
    # declarations: semua deklarasi (const, type, var,
procedure, function)
    # body: compound statement utama (mulai...selesai)

DeclarationPartNode(const_decls: List[ConstDeclNode],
                    type_decls: List[TypeDeclNode],
                    var_decls: List[VarDeclNode],
                    subprogram_decls:
List[Union[ProcedureDeclNode, FunctionDeclNode]])
    # Container untuk semua deklarasi dalam satu scope
```

Declaration Nodes (5 nodes)

```
ConstDeclNode(name: str, value: Any)
    # Deklarasi konstanta: konstanta PI = 3.14;

VarDeclNode(names: List[str], type_spec: TypeSpecNode)
    # Deklarasi variabel: variabel x, y: integer;

ProcedureDeclNode(name: str, params: List[ParamNode],
                  declarations: DeclarationPartNode,
                  body: CompoundStatementNode)
    # Deklarasi prosedur dengan parameter dan body

FunctionDeclNode(name: str, params: List[ParamNode],
                 return_type: TypeSpecNode,
                 declarations: DeclarationPartNode,
                 body: CompoundStatementNode)
    # Deklarasi fungsi dengan return type

ParamNode(names: List[str], type_spec: TypeSpecNode,
is_var: bool = False)
```

```
    # Parameter formal procedure/function
    # is_var: True untuk pass-by-reference
```

Type Specification Nodes (5 nodes)

```
TypeSpecNode
    # Base class untuk type specifications

PrimitiveTypeNode(type_name: str)
    # Tipe primitif: integer, real, boolean, char, string

CustomTypeNode(type_name: str)
    # Reference ke user-defined type

ArrayTypeNode(index_range: RangeNode, element_type:
TypeSpecNode)
    # larik [1..10] dari integer

RangeTypeNode(range_spec: RangeNode)
    # Subrange type untuk array index
```

Statement Nodes (9 nodes)

```
CompoundStatementNode(statements: List[StatementNode])
    # mulai...selesai block

AssignmentNode(target: Union[VarNode, ArrayAccessNode],
               value: ExpressionNode)
    # x := 5 atau arr[i] := 10

IfStatementNode(condition: ExpressionNode,
                then_stmt: StatementNode,
                else_stmt: Optional[StatementNode])
    # jika...maka...selain-itu

WhileStatementNode(condition: ExpressionNode, body:
StatementNode)
    # selama...lakukan

ForStatementNode(var_name: str, start: ExpressionNode,
                 end: ExpressionNode, body: StatementNode,
                 is_downto: bool)
    # untuk...ke/turun-ke...lakukan

RepeatStatementNode(body: List[StatementNode],
```

```
                    condition: ExpressionNode)
    # ulangi...sampai

ProcedureCallNode(name: str, args: List[ExpressionNode])
    # writeln(x, y)

CaseStatementNode(expression: ExpressionNode,
                  cases: List[Tuple[List[ExpressionNode],
StatementNode]],
                  else_stmt: Optional[StatementNode])
    # kasus...dari...selain-itu

EmptyStatementNode()
    # Statement kosong (;)
```

Expression Nodes (10 nodes)

```
BinOpNode(operator: str, left: ExpressionNode, right:
ExpressionNode)
    # Binary operations: +, -, *, /, bagi, mod, =, <>, <,
>, <=, >=, dan, atau

UnaryOpNode(operator: str, operand: ExpressionNode)
    # Unary operations: -, +, tidak

VarNode(name: str)
    # Variable reference

ArrayAccessNode(array_name: str, index: ExpressionNode)
    # arr[i+1]

FunctionCallNode(name: str, args: List[ExpressionNode])
    # kuadrat(x)

NumberLiteralNode(value: Union[int, float])
    # 123 atau 3.14

CharLiteralNode(value: str)
    # 'a'

StringLiteralNode(value: str)
    # 'hello'

BooleanLiteralNode(value: bool)
    # true/false
```

```
NilLiteralNode()
    # nil
```

2.3. AST Builder/Transformer

a. Transformation Methods

Implementasi 31 transformation methods untuk konversi parse tree ke AST

Category Distribution

Category	Count	Examples
Program Structure	3	transform_program(), transform_declaration_part(), transform_identifier_list()
Declarations	7	transform_const_declaration(), transform_var_declaration(), transform_procedure_declaration(), transform_function_declaration()
Types	4	transform_type(), transform_array_type(), transform_range()
Statements	9	transform_compound_statement(), transform_assignment_statement(), transform_if_statement(), transform_while_statement(), transform_for_statement()
Expressions	6	transform_expression(), transform_simple_expression(), transform_term(), transform_factor()
Helpers	2	extract_identifier(), parse_number()

b. Syntax Noise Removal

Syntax yang dibuang:

- Keyword (program, variabel, mulai, selesai, jika, maka, selain-itu, dll)
- Punctuation (;, :, ,, ., (,), [,])
- Operation (:=, =, <>, <=, >=)
- Structural (maka, lakukan, dari, sampai)

2.4. Symbol Table

a. Three-Table Architecture

- tab - Identifier Table

```
class SymbolTableEntry:
    id: str           # Identifier name ("x", "Hello",
"kuadrat")
    obj: ObjectType   # constant, variable, procedure,
function, type, program, parameter
    typ: DataType     # 0=void, 1=int, 2=real, 3=bool,
4=char, 5=array, 6=string/custom
    ref: int          # Reference ke btab/atab (-1 jika
tidak ada)
    nrm: bool         # True=value param,
False=reference param
    lev: int          # Lexical level (0=global, 1=first
nested, ...)
    adr: int          # Memory address/offset
    link: int         # Link ke previous symbol dalam
scope yang sama (-1 jika first)
```

- btab - Block Table

```
class BlockTableEntry:
    last: int      # Index of last identifier dalam
block ini
    lastpar: int   # Index of last parameter dalam block
ini
    psize: int     # Total size parameter area
    vsize: int     # Total size variable area
```

- atab - Array Table

```
class ArrayTableEntry:
    inxtyp: DataType   # Index type (biasanya INTEGER)
    eltyp: DataType    # Element type
    elref: int         # Element reference (untuk nested
array)
    low: int           # Lower bound
    high: int          # Upper bound
    elsize: int        # Element size in memory units
    size: int          # Total array size (elsize *
(high-low+1))
```

b. Reserved Words Management

Indices 0-31 Reserved untuk 32 Indonesian Keywords

```
RESERVED_WORDS = [
    # Keywords (27)
    "program", "variabel", "mulai", "selesai", "jika",
    "maka", "selain-itu", "selama", "lakukan", "untuk",
    "ke", "turun-ke", "integer", "real", "boolean",
    "char", "larik", "dari", "prosedur", "fungsi",
    "konstanta", "tipe", "string", "kasus", "ulangi",
    "sampai", "rekaman",

    # Logical operators (3)
    "dan", "atau", "tidak",

    # Arithmetic operators (2)
    "bagi", "mod"
]
```

c. Symbol Table Operations

- Enter Operations (7 methods)

```
enter_program(name: str) -> int
    # Masukkan program name ke symbol table

enter_variable(name: str, data_type: DataType,
array_ref: int) -> int
    # Masukkan variabel dengan type dan optional array
reference

enter_constant(name: str, data_type: DataType, value:
Any) -> int
    # Masukkan konstanta dengan value

enter_type(name: str, type_def: DataType, ref: int) ->
int
    # Masukkan type definition

enter_procedure(name: str, return_type: DataType, level:
int) -> int
    # Masukkan procedure/function (create new block)

enter_parameter(name: str, data_type: DataType,
by_reference: bool) -> int
    # Masukkan parameter dengan value/reference mode

enter_array(index_type: DataType, element_type:
DataType,
            low: int, high: int, element_ref: int) ->
int
```

```
    # Masukkan array info ke atab
```

- Lookup Operations (3 methods)

```
lookup(name: str) -> Optional[SymbolTableEntry]
    # Cari identifier dalam scope chain

lookup_with_index(name: str) -> tuple[SymbolTableEntry,
int]
    # Cari identifier dan return entry + index

lookup_in_current_scope(name: str) ->
Optional[SymbolTableEntry]
    # Cari hanya dalam scope sekarang (untuk duplicate
checking)
```

- Scope Operations (4 methods)

```
enter_block() -> int
    # Masuk ke block baru (create btab entry, update
display)

exit_block()
    # Keluar dari block (restore previous scope)

enter_scope() -> int  # Alias untuk enter_block()
exit_scope()          # Alias untuk exit_block()
```

2.5. Semantic Analyzer

a. Visitor Pattern Implementation

Base Visitor Class

```
class SemanticVisitor:
    def __init__(self):
        self.symbol_table = SymbolTable()
        self.errors: List[SemanticError] = []
        self.current_function = None  # Track current function
for return checking

    def visit(self, node: ASTNode) -> DataType:
        """Dispatch ke method yang sesuai berdasarkan node
type"""
        method_name = f'visit_{node.__class__.__name__}'
        visitor_method = getattr(self, method_name,
self.generic_visit)
        return visitor_method(node)
```

```
    def generic_visit(self, node: ASTNode) -> DataType:
        """Default visitor untuk node tanpa specific
handler"""
        return DataType.VOID
```

b. Attribute Attachment

- Attribute 1: tab_index

```
def visit_VarDeclNode(self, node: VarDeclNode):
    """Deklarasi variabel - enter ke symbol table"""
    # Convert type specification ke DataType
    data_type = self._get_data_type(node.type_spec)

    # Handle array type
    array_ref = -1
    if isinstance(node.type_spec, ArrayTypeNode):
        array_ref =
self._process_array_type(node.type_spec)

    # Enter each variable ke symbol table
    for name in node.names:
        # Check duplicate dalam scope sekarang
        if
self.symbol_table.lookup_in_current_scope(name):
            self.add_error(f"Duplicate declaration:
'{name}'", node)
            continue

        # Enter ke symbol table
        tab_idx = self.symbol_table.enter_variable(name,
data_type, array_ref)

        # ATTACH ATTRIBUTE 1: tab_index
        node.tab_index = tab_idx

        # ATTACH ATTRIBUTE 2: computed_type
        node.computed_type = data_type

        # ATTACH ATTRIBUTE 3: scope_level
        node.scope_level =
self.symbol_table.current_level

def visit_VarNode(self, node: VarNode) -> DataType:
    """Reference ke variabel - lookup dari symbol
table"""
    # Lookup dari symbol table
    entry, tab_idx =
self.symbol_table.lookup_with_index(node.name)
```

```
    if entry is None:
        self.add_error(f"Undeclared variable:
'{node.name}'", node)
        return DataType.VOID

    # ATTACH ATTRIBUTE 1: tab_index (reference ke entry)
    node.tab_index = tab_idx

    # ATTACH ATTRIBUTE 2: computed_type (dari
declaration)
    node.computed_type = entry.type

    # ATTACH ATTRIBUTE 3: scope_level (dari entry)
    node.scope_level = entry.lev

    return entry.type
```

- Attribute 2: computed_type

```
def visit_BinOpNode(self, node: BinOpNode) -> DataType:
    """Binary operation - type inference bottom-up"""
    # BOTTOM-UP: Visit children first
    left_type = self.visit(node.left)    # Get left
operand type
    right_type = self.visit(node.right)  # Get right
operand type

    # Type inference rules
    result_type =
self._compute_binop_type(node.operator, left_type,
right_type)

    # Type checking
    if result_type == DataType.VOID:
        self.add_error(
            f"Type mismatch: {node.operator} tidak bisa
diaplikasikan ke "
            f"{left_type.name} dan {right_type.name}",
node
        )

    # ATTACH ATTRIBUTE 2: computed_type (SYNTHESIZED)
    node.computed_type = result_type

    # ATTACH ATTRIBUTE 3: scope_level (INHERITED)
    node.scope_level = self.symbol_table.current_level

    return result_type
```

```
def _compute_binop_type(self, op: str, left: DataType,
right: DataType) -> DataType:
    """Type inference rules untuk binary operations"""
    # Arithmetic: +, -, *, /
    if op in ['+', '-', '*', '/']:
        if left == DataType.INTEGER and right ==
DataType.INTEGER:
            return DataType.INTEGER
        elif left == DataType.REAL and right ==
DataType.REAL:
            return DataType.REAL
        elif (left == DataType.INTEGER and right ==
DataType.REAL) or \
             (left == DataType.REAL and right ==
DataType.INTEGER):
            return DataType.REAL

    # Integer division: bagi, mod
    elif op in ['bagi', 'mod']:
        if left == DataType.INTEGER and right ==
DataType.INTEGER:
            return DataType.INTEGER

    # Comparison: =, <>, <, >, <=, >=
    elif op in ['=', '<>', '<', '>', '<=', '>=']:
        if left == right and left in [DataType.INTEGER,
DataType.REAL, DataType.CHAR]:
            return DataType.BOOLEAN

    # Logical: dan, atau
    elif op in ['dan', 'atau']:
        if left == DataType.BOOLEAN and right ==
DataType.BOOLEAN:
            return DataType.BOOLEAN

    # String concatenation
    elif op == '+' and left == DataType.STRING and right
== DataType.STRING:
        return DataType.STRING

    return DataType.VOID  # Type mismatch
```

- Attribute 3: scope_level

```
def visit_any_node(self, node: ASTNode):
    """Semua node mendapat scope_level dari context"""
    # INHERITED ATTRIBUTE
    node.scope_level = self.symbol_table.current_level
```

c. Validation & Error Detection

4 Categories of Semantic Errors:

- Undeclared Identifier

```
def visit_VarNode(self, node: VarNode) -> DataType:
    entry = self.symbol_table.lookup(node.name)
    if entry is None:
        self.add_error(
            f"Undeclared variable: '{node.name}'",
            node, ErrorType.UNDECLARED_VARIABLE
        )
        return DataType.VOID
```

- Type Mismatch

```
def visit_AssignmentNode(self, node: AssignmentNode):
    target_type = self.visit(node.target)
    value_type = self.visit(node.value)

    if not self._types_compatible(target_type,
value_type):
        self.add_error(
            f"Type mismatch: cannot assign
{value_type.name} to {target_type.name}",
            node, ErrorType.TYPE_MISMATCH
        )
```

- Duplicate Declaration

```
def visit_VarDeclNode(self, node: VarDeclNode):
    for name in node.names:
        if
self.symbol_table.lookup_in_current_scope(name):
            self.add_error(
                f"Duplicate declaration: '{name}'
already declared in this scope",
                node, ErrorType.DUPLICATE_DECLARATION
            )
```

- Invalid Array Bounds

```
def _process_array_type(self, array_node: ArrayTypeNode)
```

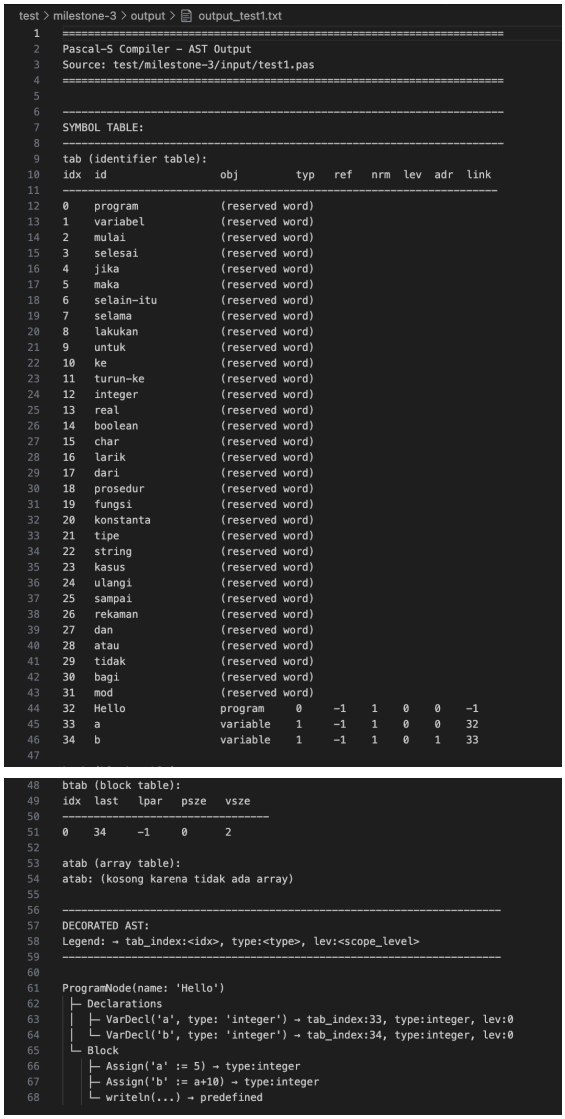
```
-> int:
    low =
self._eval_constant(array_node.index_range.start)
    high =
self._eval_constant(array_node.index_range.end)

    if low >= high:
        self.add_error(
            f"Invalid array bounds: lower bound ({low})
must be less than upper bound ({high})",
            array_node, ErrorType.INVALID_ARRAY_BOUNDS
        )
```

BAB 3

PENGUJIAN

3.1. Test Case 1 (test1.pas)

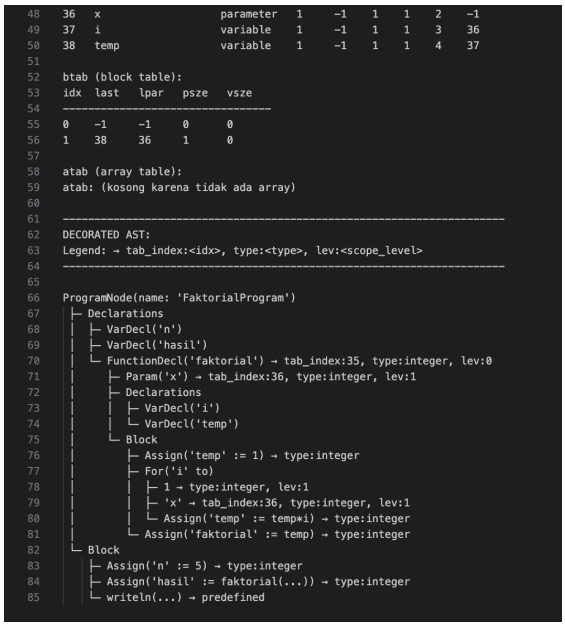
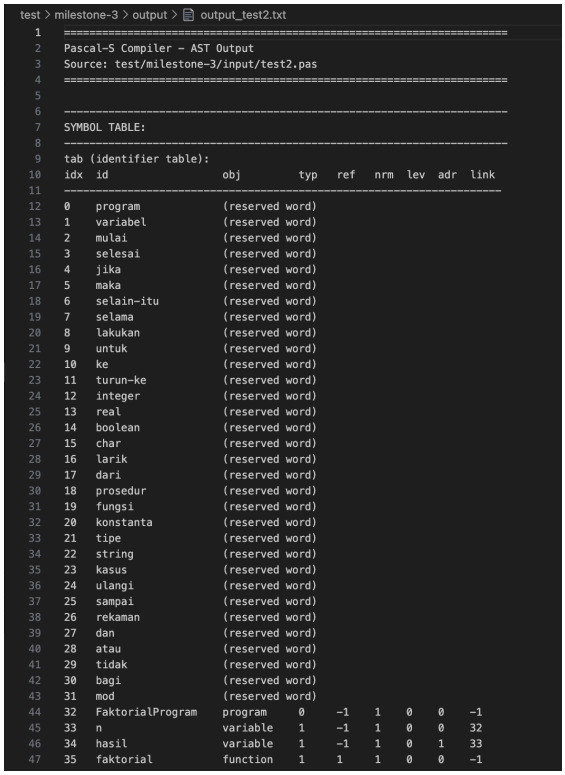
Input	Output

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

```
test > milestone-3 > output > output_test1.txt
1  ================================================================
2  Pascal-S Compiler - AST Output
3  Source: test/milestone-3/input/test1.pas
4  ================================================================
5
6  ----------------------------------------------------------------
7  SYMBOL TABLE:
8  ----------------------------------------------------------------
9  tab (identifier table):
10 idx  id                  obj         typ   ref   nrm  lev  adr  link
11 ------------------------------------------------------------------
12 0    program             (reserved word)
13 1    variabel            (reserved word)
14 2    mulai               (reserved word)
15 3    selesai             (reserved word)
16 4    jika                (reserved word)
17 5    maka                (reserved word)
18 6    selain-itu          (reserved word)
19 7    selama              (reserved word)
20 8    lakukan             (reserved word)
21 9    untuk               (reserved word)
22 10   ke                  (reserved word)
23 11   turun-ke            (reserved word)
24 12   integer             (reserved word)
25 13   real                (reserved word)
26 14   boolean             (reserved word)
27 15   char                (reserved word)
28 16   larik               (reserved word)
29 17   dari                (reserved word)
30 18   prosedur            (reserved word)
31 19   fungsi              (reserved word)
32 20   konstanta           (reserved word)
33 21   tipe                (reserved word)
34 22   string              (reserved word)
35 23   kasus               (reserved word)
36 24   ulangi              (reserved word)
37 25   sampai              (reserved word)
38 26   rekaman             (reserved word)
39 27   dan                 (reserved word)
40 28   atau                (reserved word)
41 29   tidak               (reserved word)
42 30   bagi                (reserved word)
43 31   mod                 (reserved word)
44 32   Hello               program     0     -1    1    0    0    -1
45 33   a                   variable    1     -1    1    0    0    32
46 34   b                   variable    1     -1    1    0    1    33
47
48 btab (block table):
49 idx  last   lpar   psze   vsze
50 ------------------------------
51 0    34     -1     0      2
52
53 atab (array table):
54 atab: (kosong karena tidak ada array)
55
56 ----------------------------------------------------------------
57 DECORATED AST:
58 Legend: → tab_index:<idx>, type:<type>, lev:<scope_level>
59 ----------------------------------------------------------------
60
61 ProgramNode(name: 'Hello')
62  ├─ Declarations
63  │  ├─ VarDecl('a', type: 'integer') → tab_index:33, type:integer, lev:0
64  │  └─ VarDecl('b', type: 'integer') → tab_index:34, type:integer, lev:0
65  └─ Block
66     ├─ Assign('a' := 5) → type:integer
67     ├─ Assign('b' := a+10) → type:integer
68     └─ writeln(...) → predefined
```

3.2. Test Case 2 (test2.pas)

Input	Output

`program FaktorialProgram;`
 	
`variabel`
`  n, hasil: integer;`
 	
`fungsi faktorial(x: integer): integer;`
`variabel`
`  i, temp: integer;`
`mulai`
`  temp := 1;`
`  untuk i := 1 ke x lakukan`
`  temp := temp * i;`
`  faktorial := temp;`
`selesai;`
 	
`mulai`
`  n := 5;`
`  hasil := faktorial(n);`
`  writeln('Faktorial dari ', n, ' adalah ', hasil);`
`selesai.`
	
	 

```
36  x                parameter  1    -1   1   1   2   -1
37  i                variable   1    -1   1   1   3   36
38  temp             variable   1    -1   1   1   4   37

btab (block table):
idx  last   lpar   psze   vsze
---------------------------------
0    -1     -1     0      0
1    38     36     1      0

atab (array table):
atab: (kosong karena tidak ada array)

----------------------------------------------------------------------
DECORATED AST:
Legend: → tab_index:<idx>, type:<type>, lev:<scope_level>
----------------------------------------------------------------------

ProgramNode(name: 'FaktorialProgram')
 ├─ Declarations
 │  ├─ VarDecl('n')
 │  ├─ VarDecl('hasil')
 │  └─ FunctionDecl('faktorial') → tab_index:35, type:integer, lev:0
 │     ├─ Param('x') → tab_index:36, type:integer, lev:1
 │     ├─ Declarations
 │     │  ├─ VarDecl('i')
 │     │  └─ VarDecl('temp')
 │     └─ Block
 │        ├─ Assign('temp' := 1) → type:integer
 │        ├─ For('i' to)
 │        │  ├─ 1 → type:integer, lev:1
 │        │  ├─ 'x' → tab_index:36, type:integer, lev:1
 │        │  └─ Assign('temp' := temp*i) → type:integer
 │        └─ Assign('faktorial' := temp) → type:integer
 └─ Block
    ├─ Assign('n' := 5) → type:integer
    ├─ Assign('hasil' := faktorial(...)) → type:integer
    └─ writeln(...) → predefined
```

test > milestone-3 > output > output_test2.txt

```
1   ======================================================================
2   Pascal-S Compiler - AST Output
3   Source: test/milestone-3/input/test2.pas
4   ======================================================================
5
6   ----------------------------------------------------------------------
7   SYMBOL TABLE:
8   ----------------------------------------------------------------------
9   tab (identifier table):
10  idx  id                obj         typ   ref   nrm  lev  adr  link
11  ----------------------------------------------------------------------
12  0    program            (reserved word)
13  1    variabel           (reserved word)
14  2    mulai              (reserved word)
15  3    selesai            (reserved word)
16  4    jika               (reserved word)
17  5    maka               (reserved word)
18  6    selain-itu         (reserved word)
19  7    selama             (reserved word)
20  8    lakukan            (reserved word)
21  9    untuk              (reserved word)
22  10   ke                 (reserved word)
23  11   turun-ke           (reserved word)
24  12   integer            (reserved word)
25  13   real               (reserved word)
26  14   boolean            (reserved word)
27  15   char               (reserved word)
28  16   larik              (reserved word)
29  17   dari               (reserved word)
30  18   prosedur           (reserved word)
31  19   fungsi             (reserved word)
32  20   konstanta          (reserved word)
33  21   tipe               (reserved word)
34  22   string             (reserved word)
35  23   kasus              (reserved word)
36  24   ulangi             (reserved word)
37  25   sampai             (reserved word)
38  26   rekaman            (reserved word)
39  27   dan                (reserved word)
40  28   atau               (reserved word)
41  29   tidak              (reserved word)
42  30   bagi               (reserved word)
43  31   mod                (reserved word)
44  32   FaktorialProgram   program     0     -1    1    0    0    -1
45  33   n                  variable    1     -1    1    0    0    32
46  34   hasil              variable    1     -1    1    0    1    33
47  35   faktorial          function    1     1     1    0    0    -1
```

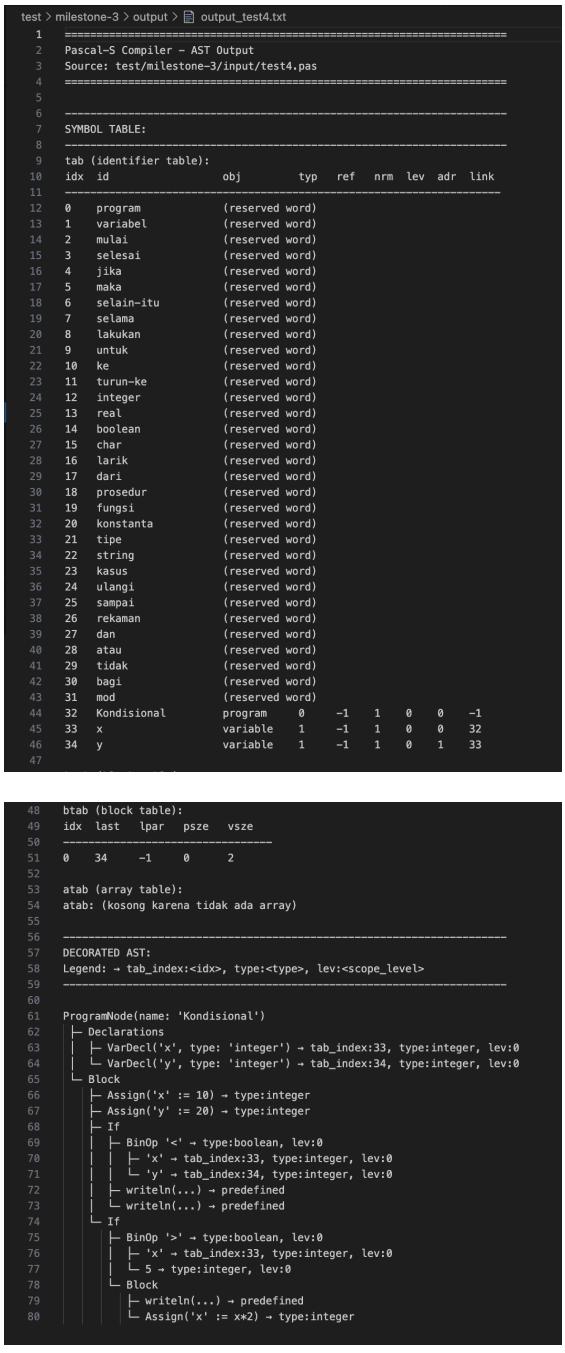
3.3. Test Case 3 (test3.pas)

Input	Output

`program ArrayTest;`
 	
`variabel`
`  arr: larik[1..5] dari integer;`
`  i: integer;`
 	
`mulai`
`  untuk i := 1 ke 5 lakukan`
`    arr[i] := i * 2;`
 	
`  i := 1;`
`  selama i <= 5 lakukan`
`  mulai`
`    writeln('arr[', i, '] = ', arr[i]);`
`    i := i + 1;`
`  selesai;`
`selesai.`
	test > milestone-3 > output > output_test3.txt
	`============================================================`
	`Pascal-S Compiler - AST Output`
	`Source: test/milestone-3/input/test3.pas`
	`============================================================`
	
	`------------------------------------------------------------`
	`SYMBOL TABLE:`
	`------------------------------------------------------------`
	`tab (identifier table):`
	`idx  id                  obj         typ   ref   nrm  lev  adr  link`
	`--------------------------------------------------------------------`
	`0    program             (reserved word)`
	`1    variabel            (reserved word)`
	`2    mulai               (reserved word)`
	`3    selesai             (reserved word)`
	`4    jika                (reserved word)`
	`5    maka                (reserved word)`
	`6    selain-itu          (reserved word)`
	`7    selama              (reserved word)`
	`8    lakukan             (reserved word)`
	`9    untuk               (reserved word)`
	`10   ke                  (reserved word)`
	`11   turun-ke            (reserved word)`
	`12   integer             (reserved word)`
	`13   real                (reserved word)`
	`14   boolean             (reserved word)`
	`15   char                (reserved word)`
	`16   larik               (reserved word)`
	`17   dari                (reserved word)`
	`18   prosedur            (reserved word)`
	`19   fungsi              (reserved word)`
	`20   konstanta           (reserved word)`
	`21   tipe                (reserved word)`
	`22   string              (reserved word)`
	`23   kasus               (reserved word)`
	`24   ulangi              (reserved word)`
	`25   sampai              (reserved word)`
	`26   rekaman             (reserved word)`
	`27   dan                 (reserved word)`
	`28   atau                (reserved word)`
	`29   tidak               (reserved word)`
	`30   bagi                (reserved word)`
	`31   mod                 (reserved word)`
	`32   ArrayTest           program     0     -1    1    0    0    -1`
	`33   arr                 variable    5     0     1    0    0    32`
	`34   i                   variable    1     -1    1    0    5    33`
	
	`btab (block table):`
	`idx  last   lpar   psze   vsze`
	`----------------------------------`
	`0    34     -1     0      6`
	
	`atab (array table):`
	`idx  xtyp  etyp  eref   low   high  elsz   size`
	`-------------------------------------------------`
	`0    1     1     -1     1     5     1      5`
	
	`------------------------------------------------------------------------`
	`DECORATED AST:`
	`Legend: → tab_index:<idx>, type:<type>, lev:<scope_level>`
	`------------------------------------------------------------------------`
	
	`ProgramNode(name: 'ArrayTest')`
	`  ├─ Declarations`
	`  │  ├─ VarDecl('arr', type: 'array of integer') → tab_index:33, type:array, lev:0`
	`  │  └─ VarDecl('i', type: 'integer') → tab_index:34, type:integer, lev:0`
	`  └─ Block`
	`     ├─ For('i' to)`
	`     │  ├─ 1 → type:integer, lev:0`
	`     │  ├─ 5 → type:integer, lev:0`
	`     │  └─ Assign('arr[i]' := i*2) → type:integer`
	`     ├─ Assign('i' := 1) → type:integer`
	`     └─ While`
	`        ├─ BinOp '<=' → type:boolean, lev:0`
	`        │  ├─ 'i' → tab_index:34, type:integer, lev:0`
	`        │  └─ 5 → type:integer, lev:0`
	`        └─ Block`
	`           ├─ writeln(...) → predefined`
	`           └─ Assign('i' := i+1) → type:integer`

3.4. Test Case 4 (test4.pas)

Input	Output

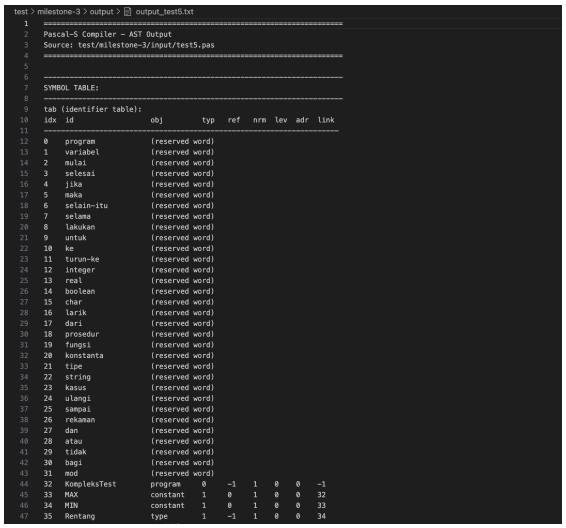
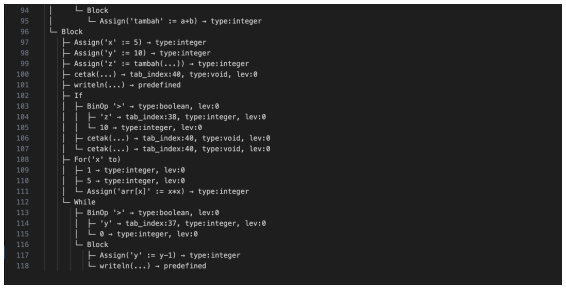
`program Kondisional;`
 	
`variabel`
`  x, y: integer;`
 	
`mulai`
`  x := 10;`
`  y := 20;`
 	
`  jika x < y maka`
`  writeln('x lebih kecil dari y')`
`  selain-itu`
`  writeln('x lebih besar atau sama dengan y');`
 	
`  jika x > 5 maka`
`  mulai`
`  writeln('x lebih besar dari 5');`
`  x := x * 2;`
`  selesai;`
`selesai.`
	

test > milestone-3 > output > output_test4.txt

```
1   ================================================================
2   Pascal-S Compiler - AST Output
3   Source: test/milestone-3/input/test4.pas
4   ================================================================
5
6   ----------------------------------------------------------------
7   SYMBOL TABLE:
8   ----------------------------------------------------------------
9   tab (identifier table):
10  idx  id                  obj         typ   ref   nrm  lev  adr  link
11  ------------------------------------------------------------------
12  0    program             (reserved word)
13  1    variabel            (reserved word)
14  2    mulai               (reserved word)
15  3    selesai             (reserved word)
16  4    jika                (reserved word)
17  5    maka                (reserved word)
18  6    selain-itu          (reserved word)
19  7    selama              (reserved word)
20  8    lakukan             (reserved word)
21  9    untuk               (reserved word)
22  10   ke                  (reserved word)
23  11   turun-ke            (reserved word)
24  12   integer             (reserved word)
25  13   real                (reserved word)
26  14   boolean             (reserved word)
27  15   char                (reserved word)
28  16   larik               (reserved word)
29  17   dari                (reserved word)
30  18   prosedur            (reserved word)
31  19   fungsi              (reserved word)
32  20   konstanta           (reserved word)
33  21   tipe                (reserved word)
34  22   string              (reserved word)
35  23   kasus               (reserved word)
36  24   ulangi              (reserved word)
37  25   sampai              (reserved word)
38  26   rekaman             (reserved word)
39  27   dan                 (reserved word)
40  28   atau                (reserved word)
41  29   tidak               (reserved word)
42  30   bagi                (reserved word)
43  31   mod                 (reserved word)
44  32   Kondisional         program     0     -1    1    0    0    -1
45  33   x                   variable    1     -1    1    0    0    32
46  34   y                   variable    1     -1    1    0    1    33
47
48  btab (block table):
49  idx  last   lpar   psze   vsze
50  ----------------------------------
51  0    34     -1     0      2
52
53  atab (array table):
54  atab: (kosong karena tidak ada array)
55
56  ----------------------------------------------------------------
57  DECORATED AST:
58  Legend: → tab_index:<idx>, type:<type>, lev:<scope_level>
59  ----------------------------------------------------------------
60
61  ProgramNode(name: 'Kondisional')
62    ├─ Declarations
63    │  ├─ VarDecl('x', type: 'integer') → tab_index:33, type:integer, lev:0
64    │  └─ VarDecl('y', type: 'integer') → tab_index:34, type:integer, lev:0
65    └─ Block
66       ├─ Assign('x' := 10) → type:integer
67       ├─ Assign('y' := 20) → type:integer
68       ├─ If
69       │  ├─ BinOp '<' → type:boolean, lev:0
70       │  │  ├─ 'x' → tab_index:33, type:integer, lev:0
71       │  │  └─ 'y' → tab_index:34, type:integer, lev:0
72       │  ├─ writeln(...) → predefined
73       │  └─ writeln(...) → predefined
74       └─ If
75          ├─ BinOp '>' → type:boolean, lev:0
76          │  ├─ 'x' → tab_index:33, type:integer, lev:0
77          │  └─ 5 → type:integer, lev:0
78          └─ Block
79             ├─ writeln(...) → predefined
80             └─ Assign('x' := x*2) → type:integer
```

3.5. Test Case 5 (test5.pas)

Input	Output

`program KompleksTest;`
 	
`konstanta`
`  MAX = 100;`
`  MIN = 0;`
 	
`tipe`
`  Rentang = 1..10;`
 	
`variabel`
`  x, y, z: integer;`
`  arr: larik[1..5] dari integer;`
 	
`prosedur cetak(msg: string);`
`mulai`
`  writeln(msg);`
`selesai;`
 	
`fungsi tambah(a, b: integer): integer;`
`mulai`
`  tambah := a + b;`
`selesai;`
 	
`mulai`
`  x := 5;`
`  y := 10;`
`  z := tambah(x, y);`
 	
`  cetak('Hasil penjumlahan:');`
`  writeln(z);`
 	
`  jika z > 10 maka`
`    cetak('z lebih besar dari 10')`
`  selain-itu`
`    cetak('z tidak lebih besar dari 10');`
 	
`  untuk x := 1 ke 5 lakukan`
`    arr[x] := x * x;`
 	
`  selama y > 0 lakukan`
`  mulai`
`    y := y - 1;`
	
	 
	 

`    writeln(y);`
`  selesai;`
`selesai.`	

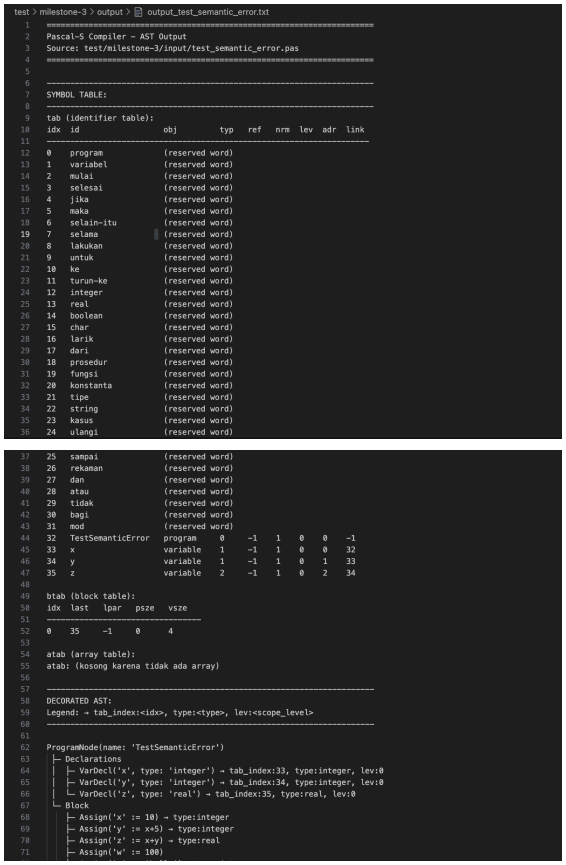
3.6. Test Case 6 (test_brutal.pas)

Test ini sangat panjang, kami hanya melampirkan link Google Drive dari test case ini yaitu:

GDrive Berikut (output_test_brutal.txt)

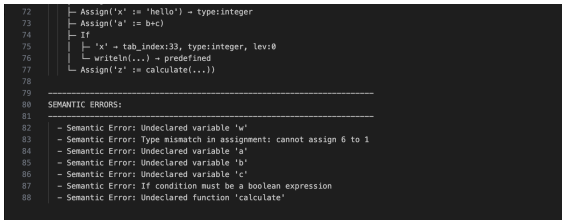
3.7. Test Case 7 (test_sematnic_error.pas)

Input	Output

`program TestSemanticError;`
 	
`variabel`
`  x, y: integer;`
`  z: real;`
 	
`mulai`
`  x := 10;`
`  y := x + 5;`
`  z := x + y;`
 	
`  { Error: undeclared variable }`
`  w := 100;`
 	
`  { Error: type mismatch - assigning string to integer }`
`  x := 'hello';`
 	
`  { Error: using variable before declaration }`
`  a := b + c;`
 	
`  { Error: wrong type in condition }`
`  jika x maka`
`    writeln('This is wrong');`
 	
`  { Error: undeclared function }`
`  z := calculate(x, y);`
	

```
test > milestone-3 > output > output_test_semantic_error.txt
1   ================================================================
2   Pascal-S Compiler - AST Output
3   Source: test/milestone-3/input/test_semantic_error.pas
4   ================================================================
5
6   ----------------------------------------------------------------
7   SYMBOL TABLE:
8   ----------------------------------------------------------------
9   tab (identifier table):
10  idx  id                obj        typ   ref   nrm  lev  adr  link
11  ------------------------------------------------------------------
12  0    program           (reserved word)
13  1    variabel          (reserved word)
14  2    mulai             (reserved word)
15  3    selesai           (reserved word)
16  4    jika              (reserved word)
17  5    maka              (reserved word)
18  6    selain-itu        (reserved word)
19  7    selama            (reserved word)
20  8    lakukan           (reserved word)
21  9    untuk             (reserved word)
22  10   ke                (reserved word)
23  11   turun-ke          (reserved word)
24  12   integer           (reserved word)
25  13   real              (reserved word)
26  14   boolean           (reserved word)
27  15   char              (reserved word)
28  16   larik             (reserved word)
29  17   dari              (reserved word)
30  18   prosedur          (reserved word)
31  19   fungsi            (reserved word)
32  20   konstanta         (reserved word)
33  21   tipe              (reserved word)
34  22   string            (reserved word)
35  23   kasus             (reserved word)
36  24   ulangi            (reserved word)
37  25   sampai            (reserved word)
38  26   rekaman           (reserved word)
39  27   dan               (reserved word)
40  28   atau              (reserved word)
41  29   tidak             (reserved word)
42  30   bagi              (reserved word)
43  31   mod               (reserved word)
44  32   TestSemanticError program    0    -1   1    0    0    -1
45  33   x                 variable   1    -1   1    0    0    32
46  34   y                 variable   1    -1   1    0    1    33
47  35   z                 variable   2    -1   1    0    2    34
48
49  btab (block table):
50  idx  last   lpar   psze   vsze
51  ------------------------------
52  0    35     -1     0      4
53
54  atab (array table):
55  atab: (kosong karena tidak ada array)
56
57  ----------------------------------------------------------------
58  DECORATED AST:
59  Legend: -> tab_index:<idx>, type:<type>, lev:<scope_level>
60  ----------------------------------------------------------------
61
62  ProgramNode(name: 'TestSemanticError')
63    ├─ Declarations
64    │  ├─ VarDecl('x', type: 'integer') -> tab_index:33, type:integer, lev:0
65    │  ├─ VarDecl('y', type: 'integer') -> tab_index:34, type:integer, lev:0
66    │  └─ VarDecl('z', type: 'real') -> tab_index:35, type:real, lev:0
67    └─ Block
68       ├─ Assign('x' := 10) -> type:integer
69       ├─ Assign('y' := x+5) -> type:integer
70       ├─ Assign('z' := x+y) -> type:real
71       ├─ Assign('w' := 100)
```

selesai.



```
    ├─ Assign('x' := 'hello') → type:integer
    ├─ Assign('a' := b+c)
    ├─ If
    │  ├─ 'x' → tab_index:33, type:integer, lev:0
    │  └─ writeln(...) → predefined
    └─ Assign('z' := calculate(...))

--------------------------------------------------------------------------------
SEMANTIC ERRORS:
--------------------------------------------------------------------------------
  - Semantic Error: Undeclared variable 'w'
  - Semantic Error: Type mismatch in assignment: cannot assign 6 to 1
  - Semantic Error: Undeclared variable 'a'
  - Semantic Error: Undeclared variable 'b'
  - Semantic Error: Undeclared variable 'c'
  - Semantic Error: If condition must be a boolean expression
  - Semantic Error: Undeclared function 'calculate'
```

BAB 4
Kesimpulan dan Saran

4.1. Kesimpulan

Pada pengerjaan Milestone 3, Semantic Analysis telah berhasil dibuat sebagai kelanjutan dari lexer dan parser yang dibangun pada milestone sebelumnya. Pada tahap ini, program yang telah dinyatakan benar secara sintaks kemudian direpresentasikan dalam bentuk Abstract Syntax Tree (AST) dan dianalisis menggunakan pendekatan visitor. Implementasi mencakup pembangunan Symbol Table bertingkat untuk menangani lingkup (scope) prosedur dan fungsi, anotasi tipe pada node AST, serta penerapan aturan-aturan semantik yang disesuaikan dengan spesifikasi Pascal-S. Pemeriksaan yang dilakukan meliputi pengecekan tipe data pada ekspresi aritmatika dan boolean, pemastian kesesuaian tipe pada assignment, validasi parameter fungsi dan prosedur, pemeriksaan indeks array, serta verifikasi kondisi pada struktur kontrol seperti if dan while. Dengan mekanisme traversal yang sistematis, Semantic Analyzer dapat memastikan bahwa setiap konstruksi program memiliki makna yang benar dan konsisten.

Selain itu, program juga telah dilengkapi dengan mekanisme pelaporan kesalahan semantik yang komprehensif. Sistem mampu mendeteksi berbagai kategori error, seperti undeclared identifier, duplicate declaration, type mismatch, penggunaan operator yang tidak valid, hingga kesalahan aturan khusus Pascal-S seperti tipe variabel loop atau batas array yang harus integer. Berdasarkan pengujian yang dilakukan terhadap sejumlah test case, termasuk kasus normal maupun kasus yang secara sengaja mengandung kesalahan, implementasi Semantic Analyzer telah mampu menghasilkan deteksi error yang tepat serta memvalidasi program yang bebas kesalahan semantik. Dengan demikian, milestone ini berhasil memenuhi spesifikasi tugas dan membentuk fondasi yang kuat untuk tahap kompilasi selanjutnya seperti pembuatan intermediate code maupun optimasi program.

4.2. Saran

Optimalisasi terhadap struktur AST yang telah teranotasi agar dapat digunakan secara lebih efisien untuk proses intermediate code generation ataupun tahapan optimasi. Sistem error handling juga dapat dikembangkan lebih lanjut, misalnya melalui penerapan error recovery yang memungkinkan analisis tetap dilanjutkan setelah menemukan kesalahan sehingga lebih banyak error dapat terdeteksi dalam satu kali kompilasi. Selain itu, perlu dipertimbangkan pengembangan mekanisme type inference sederhana untuk menangani ekspresi kompleks, serta penambahan dukungan terhadap lebih banyak fitur bahasa Pascal-S apabila dibutuhkan pada tahap backend. Dengan adanya perbaikan tersebut, compiler yang dikembangkan akan menjadi lebih robust, modular, dan siap untuk mendukung tahap kompilasi berikutnya.

LAMPIRAN

[1] I. S. Pratama, "Abstract Syntax Tree (AST)," Medium, 25 Feb 2024. [Online]. Available: https://medium.com/@ilham.suryapratama/abstract-syntax-tree-ast-91440fbca59f. [Accessed: 29-Nov-2025].

[2] "Semantic Analysis (ULiège)," [Online]. Available: https://people.montefiore.uliege.be/geurts/Cours/compil/2015/04-semantic-2015-2016.pdf [Accessed: 30-Nov-2025].

[3] "Recursive Descent Parser," GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/compiler-design/recursive-descent-parser/ [Accessed: 12-Nov-2025].

[4] TutorialsPoint, "Attributed Grammars," TutorialsPoint, [Online]. Terdapat di: https://www.tutorialspoint.com/compiler_design/compiler_design_attributed_grammars.htm. [Diakses: 29-Nov-2025].

Link Repository Github:

https://github.com/KennethhPoenadi/ARA-Tubes-IF2224.git

Pembagian Kerja:

NIM	Nama	Pembagian Kerja
13522076	Muhammad Syarafi Akmal	25%
13523040	Kenneth Poenadi	25%
13523086	Bob Kunanda	25%
13523104	Muhammad Zahran R.A.	25%