

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

Daftar Isi.....	2
BAB 1	
Landasan Teori.....	3
1.1. Pengertian Semantic Analysis.....	3
1.2. Attributed Grammar.....	3
1.3. Abstract Syntax Tree (AST) Teranotasi.....	5
1.4. Symbol Table.....	5
1.5. Visitor / Visit Functions.....	6
1.6. Error Handling Semantik.....	8
BAB 2	
PERANCANGAN & IMPLEMENTASI.....	12
2.1. Alur Kerja Program.....	12
2.2. Struktur Data AST.....	16
2.3. AST Builder/Transformer.....	20
2.4. Symbol Table.....	21
2.5. Semantic Analyzer.....	23
BAB 3	
PENGUJIAN.....	29
3.1. Test Case 1 (test1.pas).....	29
3.2. Test Case 2 (test2.pas).....	30
3.3. Test Case 3 (test3.pas).....	31
3.4. Test Case 4 (test4.pas).....	32
3.5. Test Case 5 (test5.pas).....	33
3.6. Test Case 6 (test_brutal.pas).....	34
3.7. Test Case 7 (test_sematnic_error.pas).....	34
BAB 4.....	36
Kesimpulan dan Saran.....	36
4.1. Kesimpulan.....	36
4.2. Saran.....	36
LAMPIRAN.....	37
Link Repository Github:.....	37
Pembagian Kerja:.....	37

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

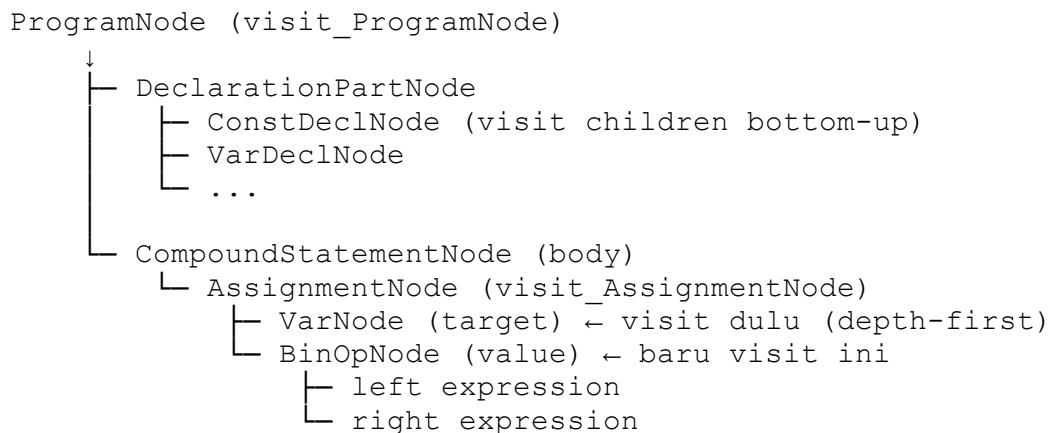
    if value_type is None:
        return # Error already reported

    # 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.



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"), ...].

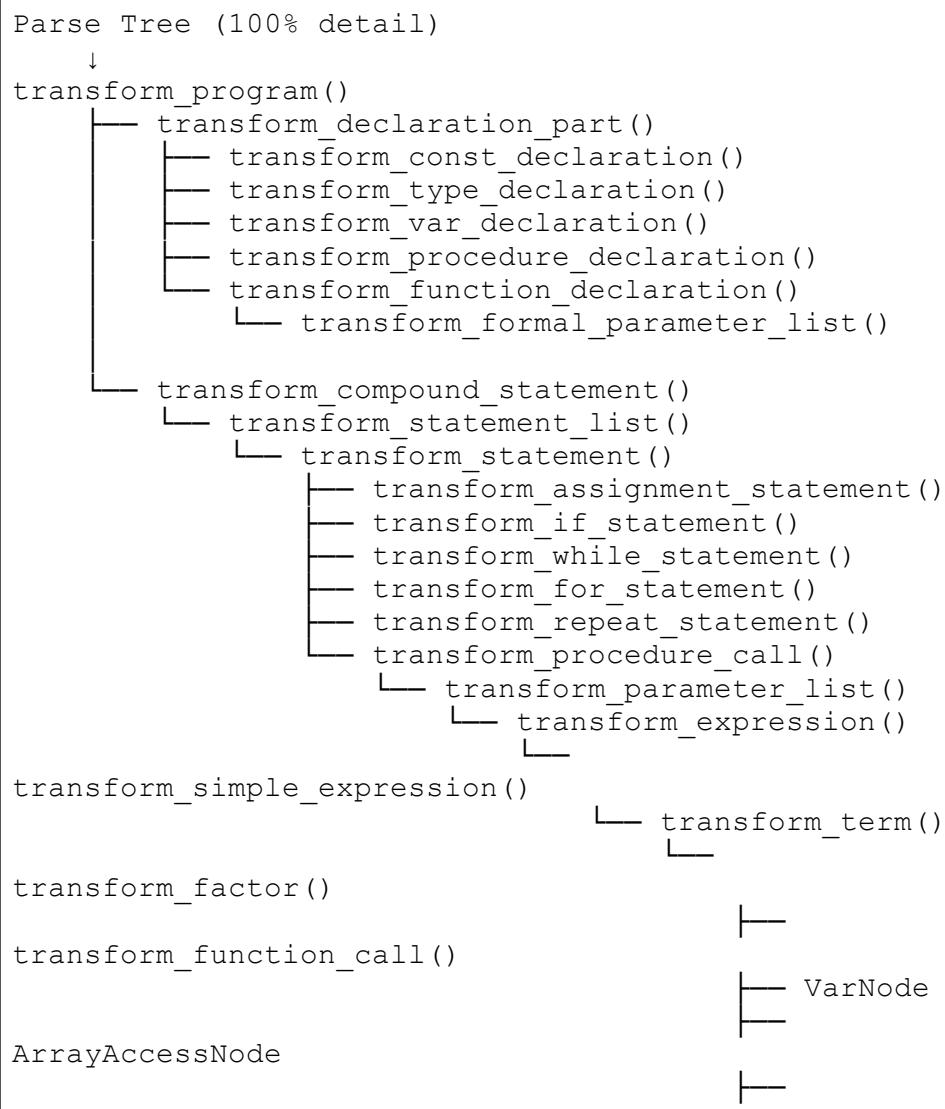
3. Syntax Analysis (Parsing)

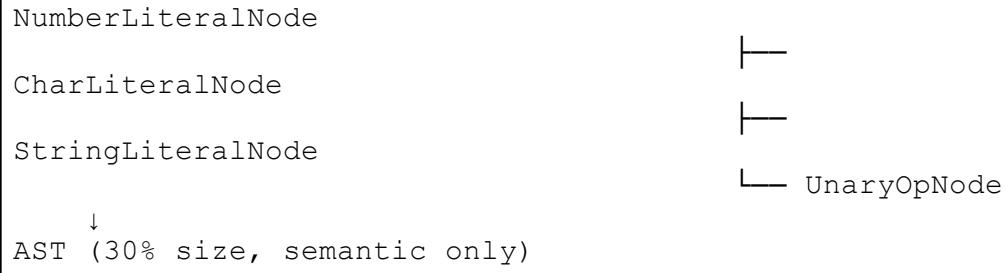
- Parser menerima list of tokens dari lexer.
- Menggunakan algoritma Recursive Descent untuk membangun parse tree.
- Setiap aturan grammar diimplementasikan sebagai fungsi terpisah (31 fungsi)
- Parse tree direpresentasikan sebagai nested dictionary dengan struktur:

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

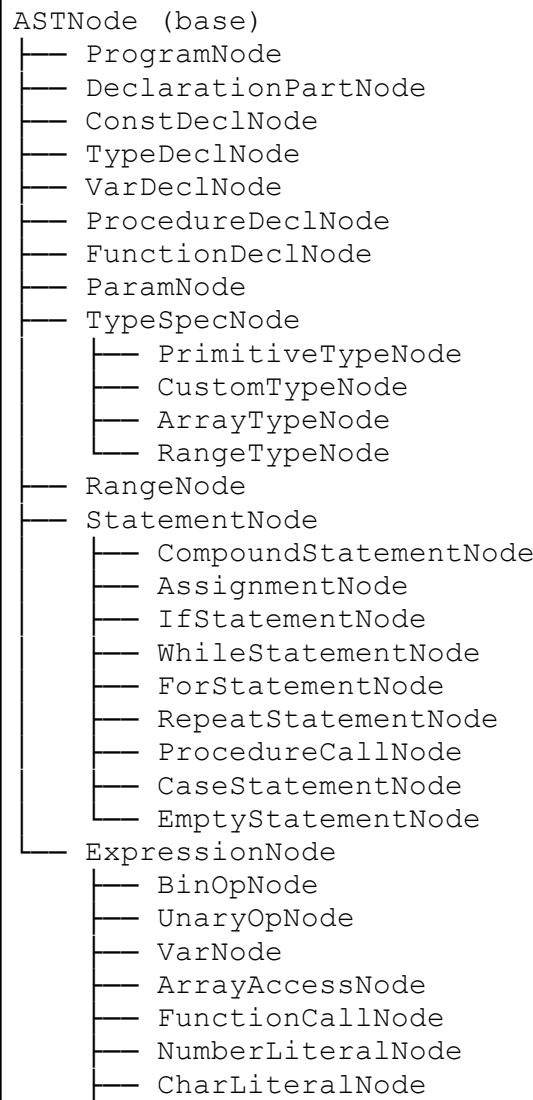
4. AST Building (Abstract Syntax Tree)

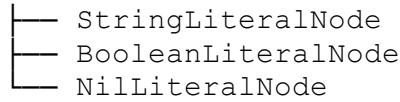
- ASTBuilder diinisialisasi
- Setup error collection untuk semantic errors
- Traverse parse tree recursively menggunakan transformer methods





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





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_declarator_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
<pre>program Hello; variabel a, b: integer; mulai a := 5; b := a + 10; writeln('Result = ', b); selesai.</pre>	<pre>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 0 program (reserved word) 12 1 variabel (reserved word) 13 2 mulai (reserved word) 14 3 selesai (reserved word) 15 4 jika (reserved word) 16 5 maka (reserved word) 17 6 selain-itu (reserved word) 18 7 selama (reserved word) 19 8 lakukan (reserved word) 20 9 untuk (reserved word) 21 10 ke (reserved word) 22 11 turun-ke (reserved word) 23 12 integer (reserved word) 24 13 real (reserved word) 25 14 boolean (reserved word) 26 15 char (reserved word) 27 16 larik (reserved word) 28 17 dari (reserved word) 29 18 prosedur (reserved word) 30 19 fungsi (reserved word) 31 20 konstanta (reserved word) 32 21 tipe (reserved word) 33 22 string (reserved word) 34 23 kasus (reserved word) 35 24 ulangi (reserved word) 36 25 sampai (reserved word) 37 26 rekaman (reserved word) 38 27 dan (reserved word) 39 28 atau (reserved word) 40 29 tidak (reserved word) 41 30 bagi (reserved word) 42 31 mod (reserved word) 43 32 Hello program 0 -1 1 0 0 -1 44 33 a variable 1 -1 1 0 0 32 45 34 b variable 1 -1 1 0 1 33 46 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</pre>

3.2. Test Case 2 (test2.pas)

Input	Output
<pre> 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. </pre>	<pre> 48 36 x parameter 1 -1 1 1 2 -1 49 37 i variable 1 -1 1 1 3 36 50 38 temp variable 1 -1 1 1 4 37 51 52 btab (block table): 53 idx last lpar psze vsze 54 ----- 55 0 -1 -1 0 0 56 1 38 36 1 0 57 58 atab (array table): 59 atab: (kosong karena tidak ada array) 60 61 ----- 62 DECORATED AST: 63 Legend: -> tab_index:<id>, type:<type>, lev:<scope_level> 64 65 66 ProgramNode(name: 'FaktorialProgram') 67 --- Declarations 68 --- VarDecl('n') 69 --- VarDecl('hasil') 70 --- FunctionDecl('faktorial') -> tab_index:35, type:integer, lev:0 71 --- Param('x') -> tab_index:36, type:integer, lev:1 72 --- Declarations 73 --- VarDecl('i') 74 --- VarDecl('temp') 75 --- Block 76 --- Assign('temp' := 1) -> type:integer 77 --- For('i' to) 78 --- 1 -> type:integer, lev:1 79 --- 'x' -> tab_index:36, type:integer, lev:1 80 --- Assign('temp' := temp*i) -> type:integer 81 --- Assign('faktorial' := temp) -> type:integer 82 --- Block 83 --- Assign('n' := 5) -> type:integer 84 --- Assign('hasil' := faktorial(...)) -> type:integer 85 --- 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 SYMBOL TABLE: 7 8 tab (identifier table): 9 id id obj typ ref nrm lev adr link 10 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 1 0 0 32 46 34 hasil variable 1 -1 1 0 1 33 47 35 faktorial function 1 1 1 0 0 -1 </pre>

3.3. Test Case 3 (test3.pas)

Input	Output
<pre> 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. </pre>	<pre> test > milestone-3 > output > output_test3.txt ===== 1 ===== 2 Pascal-S Compiler - AST Output 3 Source: test/milestone-3/input/test3.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 ArrayTest program 0 -1 1 0 0 -1 45 33 arr variable 5 0 1 0 0 32 46 34 i variable 1 -1 1 0 5 33 47 48 btab (block table): 49 idx last lpar psze vsze 50 ----- 51 0 34 -1 0 6 52 53 atab (array table): 54 idx xtyp etyp eref low high elsz size 55 ----- 56 0 1 1 -1 1 5 1 5 57 58 59 DECORATED AST: 60 Legend: -> tab_index:<idx>, type:<type>, lev:<scope_level> 61 62 63 ProgramNode(name: 'ArrayTest') 64 - Declarations 65 - VarDecl('arr', type: 'array of integer') -> tab_index:33, type:array, lev:0 66 - VarDecl('i', type: 'integer') -> tab_index:34, type:integer, lev:0 67 Block 68 - For('i' to) 69 - 1 -> type:integer, lev:0 70 - 5 -> type:integer, lev:0 71 - Assign('arr[i]' := ix2) -> type:integer 72 - Assign('i' := i) -> type:integer 73 While 74 - BinOp '<=' -> type:boolean, lev:0 75 - 'i' -> tab_index:34, type:integer, lev:0 76 - 5 -> type:integer, lev:0 77 Block 78 - writeln(...) -> predefined 79 - Assign('i' := i+1) -> type:integer </pre>

3.4. Test Case 4 (test4.pas)

Input	Output
<pre> 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. </pre>	<pre> 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 </pre> <pre> 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 </pre>

3.5. Test Case 5 (test5.pas)

Input	Output
<pre> 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; </pre>	<pre> test > milestone-3 > output > output_test5.txt ===== 1 Pascal-S Compiler - AST Output 2 Source: test/milestone-3/input/test5.pas 3 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 nulis (reserved word) 15 3 selesai (reserved word) 16 4 jika (reserved word) 17 5 nasa (reserved word) 18 6 selain-itu (reserved word) 19 7 selama (reserved word) 20 8 lakukan (reserved word) 21 9 untukuk (reserved word) 22 10 x (reserved word) 23 11 turun-ke (reserved word) 24 12 integer (reserved word) 25 13 real (reserved word) 26 14 ketulan (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 tiving (reserved word) 35 23 kasus (reserved word) 36 24 ulangi (reserved word) 37 25 sampai (reserved word) 38 26 relawan (reserved word) 39 27 dan (reserved word) 40 28 atau (reserved word) 41 29 tidak (reserved word) 42 30 bagi (reserved word) 43 31 buat (reserved word) 44 32 KompleksTest procedure @ 1 1 0 0 -1 45 33 MAX constant 1 0 1 0 0 32 46 34 MIN constant 1 0 1 0 0 33 47 35 Rentang type 1 -3 1 0 0 34 48 49 36 x variable 1 -1 1 0 0 35 50 37 y variable 1 -1 1 0 1 36 51 38 z variable 1 -1 1 0 3 37 52 39 arr variable 5 1 0 1 0 38 53 40 cetak procedure @ 1 1 0 0 -1 54 41 msg parameter 6 -1 1 1 0 -1 55 42 tambah function 1 2 1 0 0 41 56 43 a parameter 1 -1 1 1 12 41 57 44 b parameter 1 -1 1 1 13 43 58 59 tab [block table]: 60 idx last lpar psze vsze 61 0 39 -1 0 8 62 1 41 41 4 8 63 2 44 44 2 8 64 65 tab [array table]: 66 idx xtyp etyp eref low high elsz size 67 68 0 1 1 -1 1 5 1 5 69 70 71 DECORATED AST: 72 Legend: = tab_index<>idx, type<>ttype, lev<>scope_level 73 74 75 ProgramNode(name: 'KompleksTest') 76 - Declarations 77 - ConstDecl('MAX') = tab_index@33, type:integer, lev:@0 78 - ConstDecl('MIN') = tab_index@34, type:integer, lev:@0 79 - TypDecl('Rentang') = tab_index@35, type:array, lev:@0 80 - VarDecl('x') = tab_index@36, type:integer, lev:@0 81 - VarDecl('y') = tab_index@37, type:integer, lev:@0 82 - VarDecl('z') = tab_index@38, type:integer, lev:@0 83 - VarDecl('arr') = tab_index@39, type:array, lev:@0 84 - ProcedureDecl('cetak') = tab_index@40, type:void, lev:@0 85 - Param('msg') = tab_index@41, type:string, lev:@0 86 - Block 87 - If 88 - BinOp('z > 10') = type:boolean, lev:@0 89 - 'z' = tab_index@38, type:integer, lev:@0 90 - 10 = type:integer, lev:@0 91 - cetak(.,.) = tab_index@40, type:void, lev:@0 92 - 'tambah' = tab_index@42, type:integer, lev:@0 93 - Param('a', 'b') = tab_index@43, type:integer, lev:@0 94 95 - Block 96 - Assign('tambah' := a+b) = type:integer 97 - Block 98 - Assign('x := 5') = type:integer 99 - Assign('y := 10') = type:integer 100 - Assign('z := tambah(x, y)') = type:integer 101 - cetak(.,.) = tab_index@40, type:void, lev:@0 102 - writeln(., .) = predefined 103 - If 104 - BinOp('z > 10') = type:boolean, lev:@0 105 - 'z' = tab_index@38, type:integer, lev:@0 106 - 10 = type:integer, lev:@0 107 - cetak(.,.) = tab_index@40, type:void, lev:@0 108 - writeln(., .) = predefined 109 - For('x' to 1) = type:loop, lev:@0 110 - 1 = type:integer, lev:@0 111 - 5 = type:integer, lev:@0 112 - writeln('x:=', xx) = type:integer 113 - While 114 - BinOp('y > 0') = type:boolean, lev:@0 115 - 'y' = tab_index@37, type:integer, lev:@0 116 - 0 = type:integer, lev:@0 117 - Block 118 - Assign('y := y-1') = type:integer 119 - writeln(., .) = predefined 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 589 590 591 592 593 594 595 596 597 598 599 599 600 601 602 603 604 605 606 607 608 609 609 610 611 612 613 614 615 616 617 618 619 619 620 621 622 623 624 625 626 627 627 628 629 629 630 631 632 633 634 634 635 635 636 636 637 637 638 638 639 639 640 640 641 641 642 642 643 643 644 644 645 645 646 646 647 647 648 648 649 649 650 650 651 651 652 652 653 653 654 654 655 655 656 656 657 657 658 658 659 659 660 660 661 661 662 662 663 663 664 664 665 665 666 666 667 667 668 668 669 669 670 670 671 671 672 672 673 673 674 674 675 675 676 676 677 677 678 678 679 679 680 680 681 681 682 682 683 683 684 684 685 685 686 686 687 687 688 688 689 689 690 690 691 691 692 692 693 693 694 694 695 695 696 696 697 697 698 698 699 699 700 700 701 701 702 702 703 703 704 704 705 705 706 706 707 707 708 708 709 709 710 710 711 711 712 712 713 713 714 714 715 715 716 716 717 717 718 718 719 719 720 720 721 721 722 722 723 723 724 724 725 725 726 726 727 727 728 728 729 729 730 730 731 731 732 732 733 733 734 734 735 735 736 736 737 737 738 738 739 739 740 740 741 741 742 742 743 743 744 744 745 745 746 746 747 747 748 748 749 749 750 750 751 751 752 752 753 753 754 754 755 755 756 756 757 757 758 758 759 759 760 760 761 761 762 762 763 763 764 764 765 765 766 766 767 767 768 768 769 769 770 770 771 771 772 772 773 773 774 774 775 775 776 776 777 777 778 778 779 779 780 780 781 781 782 782 783 783 784 784 785 785 786 786 787 787 788 788 789 789 790 790 791 791 792 792 793 793 794 794 795 795 796 796 797 797 798 798 799 799 800 800 801 801 802 802 803 803 804 804 805 805 806 806 807 807 808 808 809 809 810 810 811 811 812 812 813 813 814 814 815 815 816 816 817 817 818 818 819 819 820 820 821 821 822 822 823 823 824 824 825 825 826 826 827 827 828 828 829 829 830 830 831 831 832 832 833 833 834 834 835 835 836 836 837 837 838 838 839 839 840 840 841 841 842 842 843 843 844 844 845 845 846 846 847 847 848 848 849 849 850 850 851 851 852 852 853 853 854 854 855 855 856 856 857 857 858 858 859 859 860 860 861 861 862 862 863 863 864 864 865 865 866 866 867 867 868 868 869 869 870 870 871 871 872 872 873 873 874 874 875 875 876 876 877 877 878 878 879 879 880 880 881 881 882 882 883 883 884 884 885 885 886 886 887 887 888 888 889 889 890 890 891 891 892 892 893 893 894 894 895 895 896 896 897 897 898 898 899 899 900 900 901 901 902 902 903 903 904 904 905 905 906 906 907 907 908 908 909 909 910 910 911 911 912 912 913 913 914 914 915 915 916 916 917 917 918 918 919 919 920 920 921 921 922 922 923 923 924 924 925 925 926 926 927 927 928 928 929 929 930 930 931 931 932 932 933 933 934 934 935 935 936 936 937 937 938 938 939 939 940 940 941 941 942 942 943 943 944 944 945 945 946 946 947 947 948 948 949 949 950 950 951 951 952 952 953 953 954 954 955 955 956 956 957 957 958 958 959 959 960 960 961 961 962 962 963 963 964 964 965 965 966 966 967 967 968 968 969 969 970 970 971 971 972 972 973 973 974 974 975 975 976 976 977 977 978 978 979 979 980 980 981 981 982 982 983 983 984 984 985 985 986 986 987 987 988 988 989 989 990 990 991 991 992 992 993 993 994 994 995 995 996 996 997 997 998 998 999 999 1000 1000 1001 1001 1002 1002 1003 1003 1004 1004 1005 1005 1006 1006 1007 1007 1008 1008 1009 1009 1010 1010 1011 1011 1012 1012 1013 1013 1014 1014 1015 1015 1016 1016 1017 1017 1018 1018 1019 1019 1020 1020 1021 1021 1022 1022 1023 1023 1024 1024 1025 1025 1026 1026 1027 1027 1028 1028 1029 1029 1030 1030 1031 1031 1032 1032 1033 1033 1034 1034 1035 1035 1036 1036 1037 1037 1038 1038 1039 1039 1040 1040 1041 1041 1042 1042 1043 1043 1044 1044 1045 1045 1046 1046 1047 1047 1048 1048 1049 1049 1050 1050 1051 1051 1052 1052 1053 1053 1054 1054 1055 1055 1056 1056 1057 1057 1058 1058 1059 1059 1060 1060 1061 1061 1062 1062 1063 1063 1064 1064 1065 1065 1066 1066 1067 1067 1068 1068 1069 1069 1070 1070 1071 1071 1072 1072 1073 1073 1074 1074 1075 1075 1076 1076 1077 1077 1078 1078 1079 1079 1080 1080 1081 1081 1082 1082 1083 1083 1084 1084 1085 1085 1086 1086 1087 1087 1088 1088 1089 1089 1090 1090 1091 1091 1092 1092 1093 1093 1094 1094 1095 1095 1096 1096 1097 1097 1098 1098 1099 1099 1100 1100 1101 1101 1102 1102 1103 1103 1104 1104 1105 1105 1106 1106 1107 1107 1108 1108 1109 1109 1110 1110 1111 1111 1112 1112 1113 1113 1114 1114 1115 1115 1116 1116 1117 1117 1118 1118 1119 1119 1120 1120 1121 1121 1122 1122 1123 1123 1124 1124 1125 1125 1126 1126 1127 1127 1128 1128 1129 1129 1130 1130 1131 1131 1132 1132 1133 1133 1134 1134 1135 1135 1136 1136 1137 1137 1138 1138 1139 1139 1140 1140 1141 1141 1142 1142 1143 1143 1144 1144 1145 1145 1146 1146 1147 1147 1148 1148 1149 1149 1150 1150 1151 1151 1152 1152 1153 1153 1154 1154 1155 1155 1156 1156 1157 1157 1158 1158 1159 1159 1160 1160 1161 1161 1162 1162 1163 1163 1164 1164 1165 1165 1166 1166 1167 1167 1168 1168 1169 1169 1170 1170 1171 1171 1172 1172 1173 1173 1174 1174 1175 1175 1176 1176 1177 1177 1178 1178 1179 1179 1180 1180 1181 1181 1182 1182 1183 1183 1184 1184 1185 1185 1186 1186 1187 1187 1188 1188 1189 1189 1190 1190 1191 1191 1192 1192 1193 1193 1194 1194 1195 1195 1196 1196 1197 1197 1198 1198 1199 1199 1200 1200 1201 1201 1202 1202 1203 1203 1204 1204 1205 1205 1206 1206 1207 1207 1208 1208 1209 1209 1210 1210 1211 1211 1212 1212 1213 1213 1214 1214 1215 1215 1216 1216 1217 1217 1218 1218 1219 1219 1220 1220 1221 1221 1222 1222 1223 1223 1224 1224 1225 1225 1226 1226 1227 1227 1228 1228 1229 1229 1230 1230 1231 1231 1232 1232 1233 1233 1234 1234 1235 1235 1236 1236 1237 1237 1238 1238 1239 1239 1240 1240 1241 1241 1242 1242 1243 1243 1244 1244 1245 1245 1246 1246 1247 1247 1248 1248 1249 1249 1250 1250 1251 1251 1252 1252 1253 1253 1254 1254 1255 1255 1256 1256 1257 1257 1258 1258 1259 1259 1260 1260 1261 1261 1262 1262 1263 1263 1264 1264 1265 1265 1266 1266 1267 1267 1268 1268 1269 1269 1270 1270 1271 1271 1272 1272 1273 1273 1274 1274 1275 1275 1276 1276 1277 1277 1278 1278 1279 1279 1280 1280 1281 1281 1282 1282 1283 1283 1284 1284 1285 1285 1286 1286 1287 1287 1288 1288 1289 1289 1290 1290 1291 1291 1292 1292 1293 1293 1294 1294 1295 1295 1296 1296 1297 1297 1298 1298 1299 1299 1300 1300 1301 1301 1302 1302 1303 1303 1304 1304 1305 1305 1306 1306 1307 1307 1308 1308 1309 1309 1310 1310 1311 1311 1312 1312 1313 1313 1314 1314 1315 1315 1316 1316 1317 1317 1318 1318 1319 1319 1320 1320 1321 1321 1322 1322 1323 1323 1324 1324 1325 1325 1326 1326 1327 1327 1328 1328 1329 1329 1330 1330 1331 1331 1332 1332 1333 1333 1334 1334 1335 1335 1336 1336 1337 1337 1338 1338 1339 1339 1340 1340 1341 1341 1342 1342 1343 1343 1344 1344 1345 1345 1346 1346 1347 1347 1348 1348 1349 1349 1350 1350 1351 1351 1352 1352 1353 1353 1354 1354 1355 1355 1356 1356 1357 1357 1358 1358 1359 1359 1360 1360 1361 1361 1362 1362 1363 1363 1364 1364 1365 1365 1366 1366 1367 1367 1368 1368 1369 1369 1370 1370 1371 1371 1372 1372 1373 1373 1374 1374 1375</pre>

<pre>writeln(y); selesai; selesai.</pre>	
--	--

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
<pre>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);</pre>	<pre>test > milestone-3 > output > output_test_semantic_error.txt ===== 1 Pascal-S Compiler - AST Output 2 Sources: test/milestone-3/input/test_semantic_error.pas 3 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 alihuan (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 kuantifier (reserved word) 33 21 tipe (reserved word) 34 22 string (reserved word) 35 23 kasus (reserved word) 36 24 ulangi (reserved word) 37 25 sempli (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 id (reserved word) 44 32 TestSemanticError program 0 -1 1 0 0 -1 45 33 x variable 1 -1 1 0 0 1 32 46 34 y variable 1 -1 1 0 1 33 47 35 z variable 2 -1 1 0 2 34 48 49 tabb (block table), 50 idx last lpar psze vsze 51 0 35 -1 0 4 52 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+y) = type:integer 70 - Assign('z' := x+y) = type:real 71 - Assign('w' := 100) ...</pre>

selesai.

```
72 |   └ Assign('x' := "hello") - type:integer
73 |   └ Assign('a' := b+c)
74 |   └ If
75 |   |   └ 'x' = tab_index33, type:integer, lev:0
76 |   |   └ writeln(..., ...) - predefined
77 |   |   └ Assign('z' := calculate(...))
78 |
79 | SEMANTIC ERRORS:
80 |
81 |
82 | - Semantic Error: Undeclared variable 'w'
83 | - Semantic Error: Type mismatch in assignment: cannot assign 6 to 1
84 | - Semantic Error: Undeclared variable 'a'
85 | - Semantic Error: Undeclared variable 'b'
86 | - Semantic Error: Undeclared variable 'c'
87 | - Semantic Error: If condition must be a boolean expression
88 | - Semantic Error: Undeclared function 'calculate'
```

BAB 4

Kesimpulan dan Saran

4.1. Kesimpulan

Pada pelaksanaan 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.ulg.ac.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%