Laporan Tugas Besar
IF2224 Teori Bahasa Formal dan Otomata
Milestone 3 - Semantic Analysis

Disusun oleh :

Refki Alfarizi	13523002
Muhammad Aditya Rahmadeni	13523028
Aryo Bama Wiratama	13523088
Fityatul Haq Rosyidi	13523116

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

2025

Daftar Isi

Bab I Deskripsi Tugas

Pada milestone ini, kami diminta untuk membuat tahapan ketiga dari *compiler*, yaitu *semantic analysis*. Tahapan ini berfungsi melakukan analisis makna (*semantic analysis*) dengan menggunakan *Attributed Grammar*.

Parse tree yang sudah dihasilkan akan ditelusuri secara *top-down* menggunakan fungsi *visit* (*semantic visitor*) pada setiap *node-nya* yang dibuat berdasarkan *grammar* yang sebelumnya didefinisikan. Selama proses ini, *symbol table* digunakan untuk menyimpan dan mengambil informasi deklarasi simbol/*identifier*.

Masukan	Parse Tree
Keluaran	Decorated AST, Symbol Table
Algoritma	L-Attributed Grammar

Pada tahap ini akan dilakukan hal sebagai berikut:

1. Membentuk Abstract Syntax Tree, termasuk:
    a) Memanfaatkan *Syntax-Directed Translation Scheme*.
    b) Identifikasi *production rule* dan *semantic action* yang terkait.
    c) Menjalankan *semantic action* pada *node parser tree*, akan terbentuk *node* baru untuk AST.
2. Type & Scope Checking, meliputi:
    a) Mengunjungi setiap node AST secara *Depth First*.
    b) *Push scope* baru pada *stack symbol table* apabila memasuki *block* kode baru.
    c) Memasukan variabel/fungsi pada *symbol table* untuk setiap deklarasi.
    d) Setiap bertemu dengan *identifier*, *lookup* pada *symbol table*.
    d) Kalkulasi *type* data dari *node* yang dikunjungi
    e) Validasi apakah setiap *type* sesuai semantik, lalu "dekorasi" AST.
    f) Hasil dari tahap ini adalah program yang sudah tervalidasi secara semantik serta *Decorated AST*, dimana setiap node minimal memiliki informasi:
        i) Tipe ekspresi (lihat pada "Semantic Analysis (ULiège)" bagaimana tipe ekspresi diperoleh)
        ii) Referensi ke symbol table
        iii) Informasi scope

Sebelum memulai proses, inisiasi symbol table terlebih dahulu. Pada Pascal-S terdapat 3 jenis *symbol table*:

1. tab: digunakan untuk menyimpan informasi mengenai identifier termasuk konstanta, variabel, prosedur, atau fungsi.

2. btab: digunakan untuk menyimpan informasi blok prosedur dan definisi tipe rekord.
3. atab: digunakan untuk menyimpan informasi *array* seperti tipe *index*, batasan *index*, tipe elemen, dan ukuran array.

Berikut merupakan daftar atribut dari symbol table

tab	
atribut	deskripsi
identifiers	Nama identifier (misalnya nama variabel, konstanta, tipe, prosedur, fungsi). Indeks dimulai dari 29 karena ada reserved words.
link	Pointer/indeks ke identifier sebelumnya dalam scope yang sama. Digunakan untuk manajemen scope (linked list per blok).
obj	Kelas objek yang dienumerasi: konstanta, variabel, tipe, prosedur, fungsi, dll.
type	Tipe dasar dari identifier, misalnya: integer, boolean, char, real, array, record, dll. Biasanya berupa kode numerik.
ref	Pointer/indeks ke tabel lain jika tipe adalah komposit (array/record). Mengarah ke atab (array table) atau btab (record/procedure block).
nrm	Menandai apakah identifier adalah variabel normal (=1) atau parameter by-reference (var parameter) (=0).
lev	Tingkat *lexical level* tempat identifier dideklarasikan (0 = global, 1 = dalam prosedur, 2 = dalam prosedur di dalam prosedur, dst).
adr	Makna tergantung jenis objek: offset variabel di stack frame, nilai konstanta, offset field record, alamat prosedur, atau ukuran/penanda lain.

atab	
atribut	deskripsi
arrays	Indeks entri array
xtyp	Tipe indeks array (misalnya integer). Berupa kode tipe dari tabel tab.
etyp	Tipe elemen array (misalnya integer). Berupa kode tipe dari tabel tab.
eref	Pointer/indeks ke detail tipe elemen jika elemen adalah tipe komposit (misalnya array dalam array, atau record). Mengarah ke atab atau btab.
low	Batas bawah indeks array (misalnya 1 pada array[1..10] atau 0 pada array[0..15]).
high	Batas atas indeks array.

elsz	Ukuran satu elemen array (dalam byte/unit memori).
size	Total ukuran array

btab	
atribut	deskripsi
blocks	Indeks entri block (setiap block mewakili prosedur, fungsi, atau record type definition).
last	Pointer/indeks ke identifier terakhir yang dideklarasikan di dalam block tersebut (menghubungkan field record, parameter, atau variabel lokal).
lpar	Pointer/indeks ke parameter terakhir dari prosedur/fungsi pada block tersebut. Jika block adalah record, nilainya 0.
psze	Total ukuran parameter block (dalam byte/unit memori).
vsze	Total ukuran variabel lokal block (dalam byte/unit memori)

Dengan memiliki informasi pada symbol table, dapat dipastikan konsistensi tipe data serta memastikan variabel tidak bisa dioperasikan jika belum diinisialisasi dengan nilai.

Input Command (contoh yang digunakan menggunakan Python)
Format: python [Compiler] [Kode Pascal]

```
python compiler.py program.pas
```

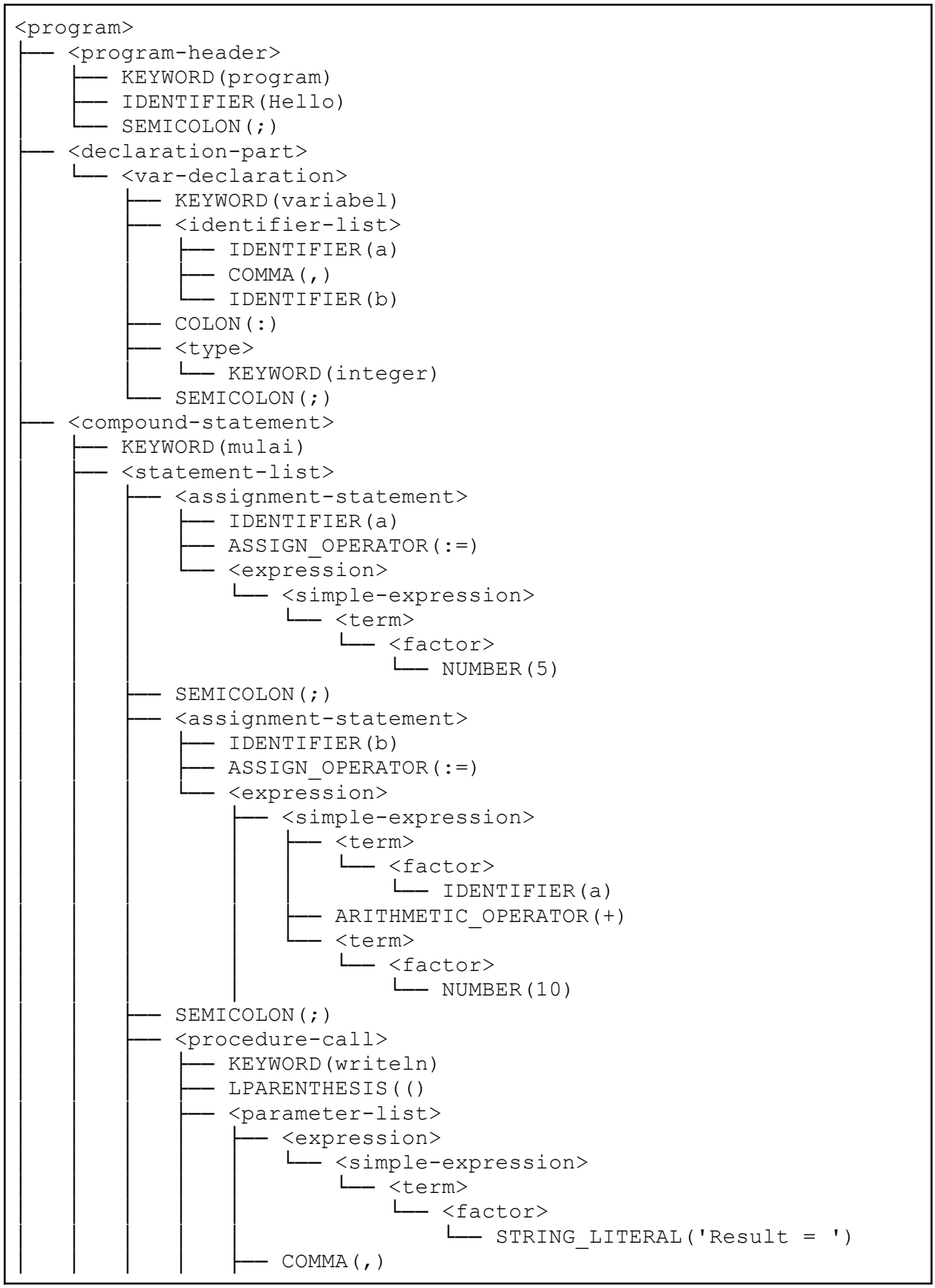
Isi program.pas

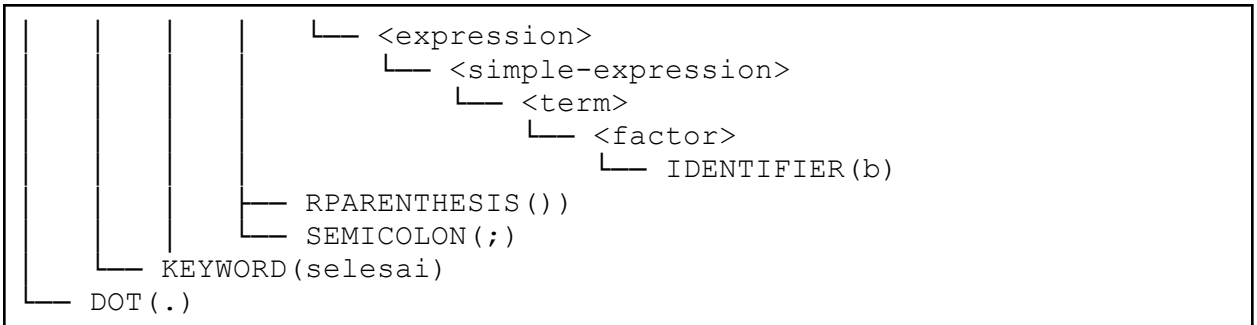
```
program Hello;

variabel
  a, b: integer;

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

Masukan (Input) untuk fase semantic analysis

```
Parse tree yang dihasilkan oleh parser
```

Isi Parse tree yang sebelumnya dihasilkan oleh parser


<program>
<program-header>
KEYWORD(program)
IDENTIFIER(Hello)
SEMICOLON(;)
<declaration-part>
<var-declaration>
KEYWORD(variabel)
<identifier-list>
IDENTIFIER(a)
COMMA(,)
IDENTIFIER(b)
COLON(:)
<type>
KEYWORD(integer)
SEMICOLON(;)
<compound-statement>
KEYWORD(mulai)
<statement-list>
<assignment-statement>
IDENTIFIER(a)
ASSIGN_OPERATOR(:=)
<expression>
<simple-expression>
<term>
<factor>
NUMBER(5)
SEMICOLON(;)
<assignment-statement>
IDENTIFIER(b)
ASSIGN_OPERATOR(:=)
<expression>
<simple-expression>
<term>
<factor>
IDENTIFIER(a)
ARITHMETIC_OPERATOR(+)
<term>
<factor>
NUMBER(10)
SEMICOLON(;)
<procedure-call>
KEYWORD(writeln)
LPARENTHESIS(()
<parameter-list>
<expression>
<simple-expression>
<term>
<factor>
STRING_LITERAL('Result = ')
COMMA(,)



Semantic Rule yang digunakan

Production Rule	**Semantic Rule**
<assignment-statement> → ID := <expr>	assignment_statement.node = new AssignNode(new VarNode(ID.lexeme), expr.node)
<expression> → <simple-expr> + <term>	expression.node = new BinOpNode('+', simple_expr.node, term.node)
<expression> → <simple-expr>	expression.node = simple_expr.node
<factor> → NUMBER	factor.node = new NumberNode(NUMBER.value)
<factor> → ID	factor.node = new VarNode(ID.lexeme)
<term> → <factor>	term.node = factor.node
<factor> → STRING_LITERAL	factor.node = new StringNode(STRING_LITERAL.value)
<procedure-call> → ID (<params>)	procedure_call.node = new ProcCallNode(ID.lexeme, params.nodes)

Proses Pembangunan AST

```
├── <assignment-statement>
│   │   │   ├── IDENTIFIER(a)
│   │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │   └── <expression>
│   │   │       └── <simple-expression>
│   │   │           └── <term>
│   │   │               └── <factor>
│   │   │                   └── NUMBER(5)
```

Misalkan Syntax Analyzer sedang memeriksa sub-tree diatas dan masuk sampai kedalam node “NUMBER(5)”

Production Rule yang digunakan adalah <factor> → ID dengan semantic rule factor.node = new VarNode(ID.lexeme)
Maka, aksi yang dilakukan adalah
factor.node = new NumberNode(5)
dan terbentuk node baru pada AST yaitu
NumberNode(5)

Kemudian naik pada node “<term>”
Production rule yang digunakan <term> → <factor>
Maka digunakan term.node = factor.node yang berakibat
term.node = NumberNode(5)
Atribut dinaikkan dalam tree (synthetized) sampai diperoleh bahwa expression.node = NumberNode(5)

Sekarang Syntax Analyzer naik sampai node <assignment-statement> dimana production rulenya
<assignment-statement> → ID := <expr>
Semantic rulenya adalah
assignment_statement.node = new AssignNode(new VarNode(ID.lexeme), expr.node)
Karena node ini punya leaf IDENTIFIER(a) dan attribut synthesized dari <expression> maka terjadi
assignment_statement.node = new AssignNode(new VarNode(‘a’), NumberNode(5))
Dan terbentuk node AST baru yaitu
AssignNode(target: VarNode('a'), value: NumberNode(5))

AST yang dihasilkan

```
ProgramNode(name: 'Hello')
  |
  +-- Declarations
  |     |
  |     +-- VarDecl(name: 'a', type: 'integer')
  |     \-- VarDecl(name: 'b', type: 'integer')
  |
  \-- Block
```

```
        |
        +-- Assign(target: Var('a'), value: Num(5))
        |
        +-- Assign(target: Var('b'),
        |          value: BinOp(op: '+',
        |                       left: Var('a'),
        |                       right: Num(10)))
        |
        \-- ProcedureCall(name: 'writeln',
                          args: [String('Result = '), Var('b')])
```

Proses Type & Scope Checking

```
Traversal dimulai dari root dengan fungsi visit(ProgramNode).
Sebelum memulai, inisialisasi ketiga tabel simbol sesuai
spesifikasi Pascal-S:

tab  = []   # identifier table (mulai dari indeks 29 untuk reserved
words)
btab = [    # block table - indeks 0 adalah program (global block)
    { "last":0, "lpar":0, "psze":0, "vsze":0 }   # btab[0] = global
block
]
atab = []   # array table (kosong awal)

display = [0]   # display[level] = indeks btab dari block pada
level tersebut
level   = 0     # lexical level (0 = global)

Contoh proses untuk program Hello yang sama:

1. visit(ProgramNode name='Hello')
   • Buat block baru untuk program (global block) → sudah ada di
btab[0]
   • Masukkan nama program ke tab:
     tab.append({
         "id": "Hello", "obj": "program", "type": 0, "ref": 0,
         "nrm": 1, "lev": 0, "adr": 0, "link": 0
     })
   • btab[0]["last"] = indeks terakhir di tab yang termasuk block
ini

2. visit(Declarations)
   • visit(VarDecl 'a', type='integer')
     • Cari tipe integer di tab (reserved, misal indeks 1 =
integer)
     • Masukkan ke tab:
       tab.append({
           "id": "a", "obj": "variable", "type": 1, "ref": 0,
           "nrm": 1, "lev": 0, "adr": 0, "link":
previous_var_in_block
```

```
        })
      • Update btab[0]["last"] = indeks baru ini
      • adr akan dihitung nanti saat code generation (ukuran tiap
variabel = 1 untuk integer)

   • visit(VarDecl 'b', type='integer')
     → sama seperti a, link menunjuk ke entri a (linked-list dalam
block)

3. visit(Block) – masuk ke compound statement (main block)
   • level += 1
   • Buat block baru di btab:
     btab.append({
         "last": 0, "lpar": 0, "psze": 0, "vsze": 0
     })
   • display.append(len(btab)-1)   # simpan indeks block level 1

4. visit(Assign target='a', value=5)
   • visit(Var 'a')
     • Lookup identifier 'a':
       mulai dari block saat ini (display[level]=1), ikuti link di
btab
       → tidak ada → turun ke level 0 → ketemu di tab[indeks_a]
     • Anotasi node: type = integer, tab_index = indeks_a, lev = 0
   • visit(Number 5) → type = integer
   • Cek kompatibilitas tipe → ok
   • Tandai variabel a sebagai initialized (bisa ditambah field
atau flag terpisah)

5. visit(Assign target='b', value=BinOp('+', Var('a'), 10))
   • Sama seperti di atas untuk 'b'
   • BinOp '+': kedua operand integer → result type integer
   • Assignment: target type = value type → ok

6. visit(ProcedureCall 'writeln')
   • Lookup 'writeln' → predefined procedure (sudah ada di tab
indeks kecil)
   • Cek jumlah dan tipe parameter (string + integer) → sesuai
```

Keluaran (output)

```
tab (hanya sebagian yang relevan):
idx  id        obj       type  ref  nrm  lev  adr  link
-------------------------------------------------------
...  (reserved words 0-28)
29   Hello     program   0     0    1    0    0    0
30   a         variable  1     0    1    0    0    31
31   b         variable  1     0    1    0    0    0
32   writeln   procedure ... (predefined)

btab:
```

```
idx  last  lpar  psze  vsze
---------------------------
0    31    0     0     2    ← global block (program), vsze=2 (a dan
b)
1    0     0     0     0    ← main block (kosong variabel lokal)

atab: (kosong karena tidak ada array)

Decorated AST (contoh anotasi minimal):
ProgramNode(name: 'Hello')
 ├─ Declarations
 │    ├─ VarDecl('a')      → tab_index:30, type:integer, lev:0
 │    └─ VarDecl('b')      → tab_index:31, type:integer, lev:0
 └─ Block                   → block_index:1, lev:1
      ├─ Assign('a' := 5)   → type:void
      │    ├─ target 'a'    → tab_index:30, type:integer
      │    └─ value 5       → type:integer
      ├─ Assign('b' := a+10)→ type:void
      │    ├─ target 'b'    → tab_index:31, type:integer
      │    └─ BinOp '+'     → type:integer
      │         ├─ 'a'      → tab_index:30, type:integer
      │         └─ 10       → type:integer
      └─ writeln(...)       → predefined, tab_index:32
```

Bab II Landasan Teori

1. Pascal-S

Pascal-S, merupakan varian subset dari bahasa pemrograman Pascal yang disederhanakan untuk tujuan pendidikan pembuatan sistem kompilasi. Dibuat oleh Niklaus Wirth pada 1970, Pascal menekankan sintaksis yang jelas, tipe data statis, dan pencegahan kesalahan saat kompilasi, terinspirasi dari ALGOL. Pascal-S menyederhanakan fitur-fitur yang dimiliki Pascal dengan mempertahankan elemen utama seperti struktur blok, deklarasi variabel eksplisit, serta dukungan untuk prosedur dan fungsi guna mendukung modularitas. Bahasa ini menggunakan tipe data statis untuk keamanan tipe dan mendukung kontrol alur terstruktur serta operator dasar.

Karena sifatnya yang *statically typed*, pemeriksaan semantik pada waktu kompilasi menjadi hal yang wajib dilakukan. Program yang benar secara sintaks bisa saja tidak bermakna secara semantik, misalkan penggunaan identifier tak terdefinisi, ketidakcocokan tipe pada ekspresi atau pemanggilan prosedur dengan parameter yang tidak sesuai.

2. Compiler

Compiler adalah perangkat lunak yang menerjemahkan kode sumber dari bahasa pemrograman tingkat tinggi ke bahasa dengan tingkatan yang lebih rendah sehingga dapat dieksekusi. Proses kompilasi biasanya terdiri dari fase-fase berikut: analisis *front-end* (leksikal, sintaksis, semantik) dan *back-end* (optimasi, generasi kode). *Front-end* fokus pada pemahaman struktur kode sumber, sementara *back-end* menangani transformasi ke representasi target.

Menurut prinsip desain *compiler*, seperti yang diuraikan dalam buku berjudul *Compilers: Principles, Techniques, and Tools* karya Aho, Sethi, dan Ullman, *compiler* harus menjamin kebenaran semantik, efisiensi eksekusi, dan penanganan kesalahan.

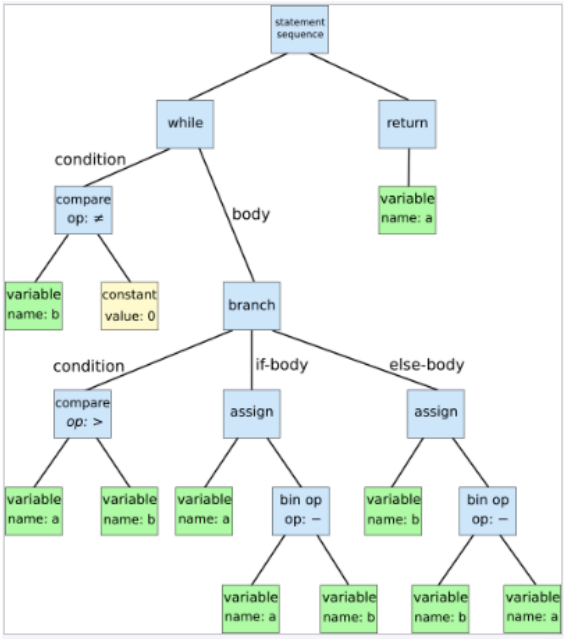
3. Semantic Analysis

Semantic Analysis biasanya merupakan tahap ketiga pada *front-end compiler* setelah *syntax analyzer* (*parser*). *Syntax Analysis* menerima *parser tree* yang telah dibangun oleh *parser*.

Semantic analyzer bertanggung jawab dalam memastikan konsistensi aturan bahasa di luar struktur sintaks seperti pemeriksaan deklarasi sebelum pemakaian, pemeriksaan tipe pada operasi dan *assignment*, validasi pemanggilan prosedur/fungsi (jumlah dan tipe parameter), serta manajemen *scope* (*lexical scoping*). *Semantic analyzer* akan melakukan anotasi terhadap *parse tree* yang telah dihasilkan. Anotasi berupa (dan tak terbatas kepada) penambahan informasi seperti tipe data dan referensi simbol (*identifier*) ke *symbol table*.

4. AST (Abstract Syntax Tree)

Abstract Syntax Tree (AST) adalah representasi pohon abstrak dari struktur program yang menekankan konstruksi semantik dan mengabaikan detail sintaksis yang tidak relevan, seperti tanda baca atau urutan aturan produksi spesifik. AST berbeda dari *parse tree* yang cenderung memuat semua simbol non-terminal dan terminal sesuai derivasi dari *grammar*; AST menyederhanakan dan mengkonsolidasikan *node* untuk merepresentasikan elemen penting seperti deklarasi, ekspresi, dan pernyataan.



An abstract syntax tree for the following code for the Euclidean algorithm:

```
while b != 0:
    if a > b:
        a := a - b
    else:
        b := b - a
return a
```

Gambar Contoh Abstract Syntax Tree
(Sumber: https://en.wikipedia.org/wiki/Abstract_syntax_tree)

Konsep *decorated* AST merujuk pada AST yang diperkaya dengan atribut semantik, misalnya tipe ekspresi, referensi ke entri tabel simbol, atau informasi scoping, yang membuat AST siap dipakai untuk validasi lanjut dan proses *back-end* seperti alokasi memori dan generasi kode.

5. Symbol Table

Symbol table adalah struktur data yang menyimpan informasi tentang *identifier* dalam suatu program seperti nama variabel, konstanta, tipe, prosedur/fungsi, beserta metadata yang relevan

seperti tipe, lokasi deklarasi (*scope/level*), dan informasi tambahan seperti ukuran, alamat/offset, atau pointer ke struktur tipe komposit.

Fungsi utama *symbol table* adalah untuk melakukan *lookup* saat *semantic analyzer* dan *generator kode* perlu mengakses properti *identifier*, serta membantu *manajemen scope* (misalkan *nested block*) melalui mekanisme seperti *stack scope*, *chaining per-block*, atau *display*. Desain *symbol table* dapat bervariasi (*array*, *hash table*, *linked lists per block*, atau kombinasi), dan untuk tipe komposit sering digunakan tabel pendamping yang menyimpan detail struktur seperti tabel tipe atau tabel *array*, sehingga *entry identifier* hanya perlu menyimpan referensi ke metadata tersebut.

Bab III Perancangan dan Implementasi

Untuk kode dari implementasi program bisa dilihat pada bagian lampiran.

Secara teknis, perancangan *compiler* ini menggunakan bahasa Python, karena memungkinkan kami untuk melakukan pengembangan (terutama *debugging*) lebih cepat serta relatif lebih sederhana dibanding bahasa lain.

Alur kerja program dimulai dengan membaca input program lalu akan melalui proses *lexical analysis* yang mengonversi aliran karakter menjadi token-token yang telah diklasifikasikan. Token-token ini kemudian diproses oleh *parser* menggunakan metode *recursive descent*, yang menghasilkan sebuah *parse tre*. Setelah AST terbentuk, tahap berikutnya adalah *semantic analysis*, yang dilakukan menggunakan pola Visitor untuk menelusuri pohon secara sistematis sekaligus membangung AST yang teranotasi. Visitor ini memeriksa aturan-aturan semantik seperti kompatibilitas tipe, validitas pemanggilan prosedur, pengecekan deklarasi variabel, dan pengisian informasi tabel simbol. Apabila ditemukan pelanggaran aturan semantik, program akan menghasilkan pesan kesalahan yang menjelaskan jenis dan lokasi kesalahan tersebut. Jika seluruh pemeriksaan berhasil dilalui, AST yang valid akan dihasilkan sebagai representasi struktural program yang siap digunakan untuk tahap berikutnya, seperti interpretasi atau kompilasi lanjutan.

1. Kelas Utama pada Semantic Analysis

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

1. **ASTNode**
   Berisi implementasi struktur data *abstract syntax tree*. Dari kelas ini akan diturunkan menjadi berbagai tipe *node* yang isinya menyesuaikan dengan karakteristik *node* tersebut. Untuk setiap *node*, pasti akan memiliki informasi tipe, index pada *symbol table*, level *scope*, beserta lokasi pada program (*line/col*).

2. **SemanticAnalyzer**
   Berisi implementasi utama *semantic analyzer* yang menyimpan *symbol table*, menelusuri *parse tree*, dan membangun AST. Pembangunan AST dilakukan bersamaan dengan analisis semantik karena pada spesifikasi tertulis “Menjalankan semantic action pada node parser tree, akan terbentuk node baru untuk AST.” Kelas ini juga berisi metode-metode *helper* untuk kebutuhan penulisan output.

2. Struktur Repositori Program

Berikut adalah struktur repositori proyek yang digunakan.

```
SMN-Tubes-IF222/
├── assets
├── doc
├── README.md
├── src
│   ├── cli.py
│   ├── errors.py
│   ├── lexer
│   │   ├── charstream.py
│   │   ├── __init__.py
│   │   ├── rules_loader.py
│   │   ├── scanner.py
│   │   └── token.py
│   ├── main.py
│   ├── parser
│   │   ├── __init__.py
│   │   ├── parser.py
│   │   └── parsertree.py
│   ├── rules
│   │   └── dfa.json
│   └── semantic
│       ├── analyzer.py
│       ├── ast_nodes.py
│       └── __init__.py
└── test
    ├── milestone-1
    ├── milestone-2
    └── milestone-3
```

3. Struktur AST

Berikut merupakan daftar tipe *node* yang digunakan untuk membangun AST lengkap dengan atribut, dekorasi. dan perannya.

No	Kelas AST	Atribut utama	Dekorasi	Peran

1	`ProgramNode`	`name: str`
		`declarations: list`
		`block: BlockNode`
			`tab_index`
			`type`
			`line/col`
				Root AST, mengelola *block* global

2	`BlockNode`	`statements: list[ASTNode]`	`block_index` `scope_level`	Representasi *compound statement*

				dan menandai *scope* baru

3	`VarDeclNode`	`name: str`
		`var_type_name: str`
			`tab_index`
			`type`
			`scope_level`
				Deklarasi variabel/parameter

4	`ConstDeclNode`	`name: str`
		`value`
		`const_type`
			`tab_index`
			`type`
				Deklarasi konstanta (value disimpan di *symbol table* pada kolom adr)

5	`TypeDeclNode`	`name: str`
		`type_def`
			`tab_index`
			`type`
			`ref`
				Definisi alias tipe

6	`ProcedureDeclNode`	`name: str`
		`params: list[VarDeclNode]`
		`block: BlockNode`
			`tab_index`
			`block_index`
			`type`
				Deklarasi prosedur dan menginisiasi block level baru

7	`FunctionDeclNode`	`name: str`
		`params`
		`return_type: str`
		`block`
			`tab_index`
			`type`
			`block_index`
				Deklarasi fungsi dengan tipe *return*

8	`AssignNode`	`target: VarNode | ArrayAccessNode` `value`	`type`	Assignment dan *semantic check* kompabilitas tipe

9	`BinOpNode`	`op: str`
		`left`
		`right`
			`type`	operasi aritmatik/logika

10	`UnaryOpNode`	`op` `operand`	`type`	unary minus, logical not, dsb.

11	`VarNode`	`name: str`	`tab_index`
			`type`
			`scope_level`
				referensi variable/parameter/konstanta

12	`NumberNode`	`value`	`type`	literal numeric
13	`StringNode`	`value`	`type`	literal
14	`CharNode`	`value`	`type`	literal

15	`BooleanNode`	`value`	`type`	literal
16	`ProcCallNode`	`name: str` `args: list`	`tab_index` `type`	pemeriksaan argumen dan *lookup reserved* atau *user procedure*

17	`IfNode`	`condition`
		`then_stmt`
		`else_stmt`
			`type`	kontrol alur

18	`WhileNode`	`condition` `body`	`type`	loop

19	`ForNode`	`iterator: VarNode`
		`start_expr`
		`end_expr`
		`direction`
		`body`
			`type`
			`iterator.tab_index`
				loop dengan iterator

20	`ArrayClassNode`	`name: str` `index_expr`	`tab_index` `type`	akses elemen *array*

4. Symbol Tables

Terdapat 29 *reserved* slot (dengan tambahan 1 slot kosong) pada tab yang kami tentukan sebagai berikut. Selain dari *reserved* slot tersebut, simbol akan ditambahkan pada tabel di paling akhir (konsep *stack*).

Indeks/*identifier*	**Nama**	**Tipe Objek**	**Kode Tipe**
1	integer	type	1
2	boolean	type	2
3	char	type	3
4	real	type	4
5	string	type	5
6	array	type	6
7	false	constant	2
8	true	constant	2

9	abs	function	0
10	sqr	function	0
11	odd	function	2
12	chr	function	3
13	ord	function	1
14	succ	function	0
15	pred	function	0
16	round	function	1
17	trunc	function	1
18	sin	function	4
19	cos	function	4
20	exp	function	4
21	ln	function	4
22	sqrt	function	4
23	arctan	function	4
24	eof	function	2
25	eoln	function	2
26	write	procedure	0
27	writeln	procedure	0
28	read	procedure	0
29	readln	procedure	0
30	<empty>	-	-

Untuk btab, implementasi berupa list blok, untuk setiap entry memiliki atribut last, lpar, psze, dan vsze. Indeks pertama sudah terisi untuk *global program*.

Untuk atab, implementasi berupa list untuk array dengan metadata xtyp, etyp, eref, low, high, elsz, dan size.

5. Visitor

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

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

Berikut merupakan daftar tipe *node* yang digunakan untuk membangun AST lengkap dengan atribut, dekorasi. dan perannya.

No	Visitor	Aksi	Output
1	`analyze visit_program`	Daftarkan program di tab; proses deklarasi; buka scope main; proses compound statement	ProgramNode (decorated). Error: invalid root
2	`visit_var_declaration`	Resolve tipe (literal/alias/array lalu dapatkan kode tipe atau buat atab); add_to_tab untuk tiap identifier	list VarDeclNode (tab_index terisi). Error: duplicate id / unknown type
3	`visit_const_declaration`	Infer tipe literal; tambah konstanta ke tab (adr = value)	list ConstDeclNode. Error: invalid literal
4	`visit_type_declaration`	Jika array, buat entri atab dan simpan ref; jika bukan, register alias tipe di tab	list TypeDeclNode. Error: malformed type
5	`visit_procedure_decl visit_function_decl`	add_to_tab proc/func; buka scope baru; tambahkan params ke tab; proses body; tutup scope	ProcDeclNode / FuncDeclNode (block_index). Error: duplicate, unknown param type
6	`visit_compound_statement visit_statement_list`	Kumpulkan statement dengan memanggil visit_statement	BlockNode (block_index set saat dibuat)
7	`visit_assignment`	lookup target; proses index bucket bila array; visit_expression untuk nilai; cek bukan konstanta; cek kompatibilitas tipe (izinkan int→real jika diinginkan)	AssignNode. Error: undeclared id, assign to constant, type mismatch
8	`visit_expression visit_simple_expression`	Depth-first untuk sub-ekspresi; buat BinOpNode/UnaryOpNode; tentukan type hasil (numeric/real/boolean)	Expression node (typed). Error: operator operand mismatch

	visit_term visit_factor		
9	visit_proc_all	lookup nama; proses arg list; cocokkan tipe/jumlah arg bila info tersedia	ProcCallNode (type jika function). Error: undeclared proc/func, param mismatch
10	visit_if visit_while visit_for	Evaluasi condition (harus boolean untuk if/while); proses body; pada for cek iterator dan bounds tipe	IfNode / WhileNode / ForNode. Error: condition not boolean, iterator undeclared
11	visit_identifier_list	Ambil daftar nama + posisi untuk dipakai deklarasi	list (name, line, col)

6. Semantic Rule

Berikut merupakan daftar aturan semantik yang kami gunakan.

No	Production Rule	Semantic Rule
1	<program> → program ID ; <declaration-part> <compound-statement> .	program.node = new ProgramNode(ID.lexeme, declarations.node, block.node); add_to_tab(ID.lexeme, "program", 0)
2	<declaration-part> → { <var_declaration> | <const_declaration> | <type_declaration> | <subprogram_declaration> }	declarations.node = concat(child.nodes) (visit tiap deklarasi)
3	<var_declaration> → variabel <identifier-list> : <type> ;	untuk tiap id ∈ identifier-list: idx = add_to_tab(id.lexeme, "variable", type.code, ...); var_decl.node = new VarDeclNode(id.lexeme, type.name); var_decl.node.tab_index = idx
4	<const_declaration> → konstanta ID = <const_value> ;	infer tipe/val; idx = add_to_tab(ID.lexeme, "constant", const_type, val=const_val); const_decl.node = new ConstDeclNode(ID.lexeme,

		const_val, const_type)
5	<type_declaration> → tipe ID = <type> ;	jika <type> adalah array: ref = new_atab(entry); idx = add_to_tab(ID.lexeme, "type", TYPE_CODE, ref=ref); type_decl.node = new TypeDeclNode(ID.lexeme, def)
6	<array_type> → larik [<range>] dari <type>	buat entry atab: atab_idx = atab.append({xtyp, etyp, low, high, elsz, size}); return type.code=array, ref=atab_idx
7	<subprogram_declaration> → <procedure_declaration> | <function_declaration>	dispatch ke masing-masing; return ProcDeclNode / FuncDeclNode
8	<procedure_declaration> → procedure ID (<formal_parameter_list>) ; <declaration-part> <compound_statement> ;	add_to_tab(ID, "procedure", 0); buka scope baru; tambah params ke tab; body.node = visit compound; proc.node = new ProcedureDeclNode(ID.lexeme, params, body); tutup scope
9	<function_declaration> → function ID (<formal_parameter_list>?) : IDtype ; <declaration-part> <compound_statement> ;	ret_code = resolve(type); add_to_tab(ID, "function", ret_code); buka scope; tambah params; body.node = visit compound; func.node = new FunctionDeclNode(ID.lexeme, params, ret_type, body); tutup scope
10	<formal_parameter_list> → <parameter_group> { ; <parameter_group> }	params.node = concat(each parameter_group.nodes); setiap param -> add_to_tab(name,"variable",type) (set nrm jika by-ref)
11	<compound_statement> → mulai <statement-list> selesai	block.node = new BlockNode(statement_list.nodes); set block.node.block_index = current_btab_index
12	<statement-list> → <statement> { ; <statement> }	stmts.node = list of visit_statement(children)
13	<assignment-statement> →	target = new

	ID := <expression>	VarNode(ID.lexeme); idx,info = lookup(ID); expr = visit_expression(...); cek info.obj != 'constant'; cek target.type kompatibel dengan expr.type (atau lakukan konversi yang diizinkan); assign.node = new AssignNode(target, expr)
14	<assignment-statement> → ID [<expression>] := <expression>	lookup ID → harus array; index = visit_expression(...) (cek integer); elem_type = atab[tab[ID].ref].etyp; target = new ArrayAccessNode(ID.lexeme, index); target.type = elem_type; selanjutnya sama seperti assignment biasa
15	<expression> → <simple-expression> [<relop> <simple-expression>]	jika ada relop: left = simple.node; right = simple.node2; node = new BinOpNode(relop.lexeme, left, right); node.type = boolean jika tidak node = simple.node
16	<simple-expression> → [+ | -] <term> { addop <term> }	jika unary sign: node = new UnaryOpNode(sign, term.node); untuk addops (+,-,or): node = new BinOpNode(op, left, right); tentukan numeric/boolean result type sesuai operand
17	<term> → <factor> { mulop <factor> }	untuk mulops (*, /, div, mod, and): node = new BinOpNode(op, left, right); tetapkan tipe hasil (int/real/boolean) sesuai operator dan operand
18	<factor> → NUMBER	factor.node = new NumberNode(NUMBER.value); set type = integer/real tergantung literal
19	<factor> → ID	idx,info = lookup(ID); jika info.obj == 'constant' -> buat literal node sesuai konstanta; jika tidak factor.node = new VarNode(ID.lexeme); factor.node.tab_index = idx;

		factor.node.type = info.type
20	<factor> → ID (<parameter-list>)	args = visit_param_list(...); cek lookup(ID) bahwa ID adalah proc/func; proc_call.node = new ProcCallNode(ID.lexeme, args); jika fungsi set proc_call.node.type = info.type
21	<factor> → '(' <expression> ')'	factor.node = expression.node
22	<procedure/function-call> → ID (<params>)	sama seperti factor dengan call: proc_call.node = new ProcCallNode(ID.lexeme, params.nodes); cek jumlah/tipe argumen bila tersedia
23	<if-statement> → jika '(' <expression> ')' <statement> [else <statement>]	cond = visit_expression(...) (harus boolean); then_n = visit_statement(...); else_n = visit_statement(...) (opsional); if.node = new IfNode(cond, then_n, else_n)
24	<while-statement> → selama '(' <expression> ')' <statement>	cond = visit_expression(...) (harus boolean); body = visit_statement(...); while.node = new WhileNode(cond, body)
25	<for-statement> → untuk ID := <expression> to/downto <expression> do <statement>	iter_idx = lookup(ID) (harus variable int); start = visit_expression(...); end = visit_expression(...); for.node = new ForNode(VarNode(ID), start, end, direction, body); cek tipe integer pada bounds
26	<parameter-list> → <expression> { , <expression> }	params.nodes = list of expression.nodes

Bab IV Pengujian

1. program.pas

Kasus uji ini mencakup kasus yang dicantumkan pada dokumen *Spesifikasi Milestone 3* sebagai salah satu contoh hasil yang diharapkan.

Input

```
program Hello;

variabel
  a, b: integer;

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

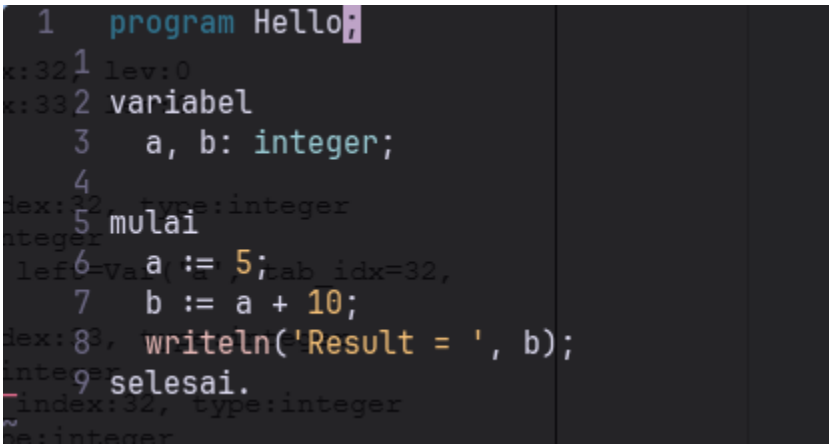
Output

```
tabs:
idx   id              obj           type    ref     nrm   lev
adr     link
-------------------------------------------------------------
-----------------
1     integer         type          1       0       1     0
0       0
2     boolean         type          2       0       1     0
0       0
3     char            type          3       0       1     0
0       0
4     real            type          4       0       1     0
0       0
5     string          type          5       0       1     0
0       0
6     array           type          6       0       1     0
0       0


7     false           constant      2       0       1     0
0       0
8     true            constant      2       0       1     0
1       0
9     abs             function      0       0       1     0
0       0
```

```
10    sqr               function      0       0       1      0
0       0
11    odd               function      2       0       1      0
0       0
12    chr               function      3       0       1      0
0       0
13    ord               function      1       0       1      0
0       0
14    succ              function      0       0       1      0
0       0
15    pred              function      0       0       1      0
0       0
16    round             function      1       0       1      0
0       0
17    trunc             function      1       0       1      0
0       0
18    sin               function      4       0       1      0
0       0
19    cos               function      4       0       1      0
0       0
20    exp               function      4       0       1      0
0       0
21    ln                function      4       0       1      0
0       0
22    sqrt              function      4       0       1      0
0       0
23    arctan            function      4       0       1      0
0       0
24    eof               function      2       0       1      0
0       0
25    eoln              function      2       0       1      0
0       0
26    write             procedure     0       0       1      0
0       0
27    writeln           procedure     0       0       1      0
0       0
28    read              procedure     0       0       1      0
0       0
29    readln            procedure     0       0       1      0
0       0
31    Hello             program       0       0       1      0
0       0
32    a                 variable      1       0       1      0
0       31
33    b                 variable      1       0       1      0
0       32

btab:
idx   last    lpar    psze    vsze
-----------------------------------
0     33      0       0       2
```

```
1     0       0       0       0

atab:
(empty because no array)

Decorated AST:
└─ ProgramNode('Hello')
   ├─ VarDeclNode('a') → tab_index:32, lev:0
   ├─ VarDeclNode('b') → tab_index:33, lev:0
   └─ Block → block_index:1
      ├─ AssignNode Num(5)
      │  ├─ VarNode('a') → tab_index:32, type:integer
      │  └─ NumberNode 5 → type:integer
      ├─ AssignNode BinOp(op='+', left=Var('a', tab_idx=32,
type=1), right=Num(10), type=1)
      │  ├─ VarNode('b') → tab_index:33, type:integer
      │  └─ BinOpNode '+' → type:integer
      │     ├─ VarNode('a') → tab_index:32, type:integer
      │     └─ NumberNode 10 → type:integer
      └─ ProcCallNode('writeln') → type:void/none
         ├─ StringNode Result =  → type:string
         └─ VarNode('b') → tab_index:33, type:integer
```

Bukti Input (Screenshot)



Bukti Output (Screenshot)

```
tabs:
idx   id              obj          type    ref     nrm   lev   adr     link
-------------------------------------------------------------------------------
1     integer         type         1       0       1     0     0       0
2     boolean         type         2       0       1     0     0       0
3     char            type         3       0       1     0     0       0
4     real            type         4       0       1     0     0       0
5     string          type         5       0       1     0     0       0
6     array           type         6       0       1     0     0       0
7     false           constant     2       0       1     0     0       0
8     true            constant     2       0       1     0     1       0
9     abs             function     0       0       1     0     0       0
10    sqr             function     0       0       1     0     0       0
11    odd             function     2       0       1     0     0       0
12    chr             function     3       0       1     0     0       0
13    ord             function     1       0       1     0     0       0
14    succ            function     0       0       1     0     0       0
15    pred            function     0       0       1     0     0       0
16    round           function     1       0       1     0     0       0
17    trunc           function     1       0       1     0     0       0
18    sin             function     4       0       1     0     0       0
19    cos             function     4       0       1     0     0       0
20    exp             function     4       0       1     0     0       0
21    ln              function     4       0       1     0     0       0
22    sqrt            function     4       0       1     0     0       0
23    arctan          function     4       0       1     0     0       0
24    eof             function     2       0       1     0     0       0
25    eoln            function     2       0       1     0     0       0
26    write           procedure    0       0       1     0     0       0
27    writeln         procedure    0       0       1     0     0       0
28    read            procedure    0       0       1     0     0       0
29    readln          procedure    0       0       1     0     0       0
31    Hello           program      0       0       1     0     0       0
32    a               variable     1       0       1     0     0       31
33    b               variable     1       0       1     0     0       32

btab:
idx   last    lpar    psze    vsze
-----------------------------------
0     33      0       0       2
1     0       0       0       0

atab:
(empty because no array)

Decorated AST:
└─ ProgramNode('Hello')
   ├─ VarDeclNode('a') → tab_index:32, lev:0
   ├─ VarDeclNode('b') → tab_index:33, lev:0
   └─ Block → block_index:1
      ├─ AssignNode Num(5)
      │  ├─ VarNode('a') → tab_index:32, type:integer
      │  └─ NumberNode 5 → type:integer
      ├─ AssignNode BinOp(op='+', left=Var('a', tab_idx=32, type=1), right=Num(10), type=1)
      │  ├─ VarNode('b') → tab_index:33, type:integer
      │  └─ BinOpNode '+' → type:integer
      │     ├─ VarNode('a') → tab_index:32, type:integer
      │     └─ NumberNode 10 → type:integer
      └─ ProcCallNode('writeln') → type:void/none
         ├─ StringNode Result =  → type:string
         └─ VarNode('b') → tab_index:33, type:integer
y4nked  tbfo3  feat/semantic ±
```

2. type_mismatch.pas

Kasus uji ini mencakup kasus yang menguji *analyzer* untuk menolak *assignment* pada variabel dengan tipe berbeda, pada kasus ini yaitu *assignment* string ke integer.

Input

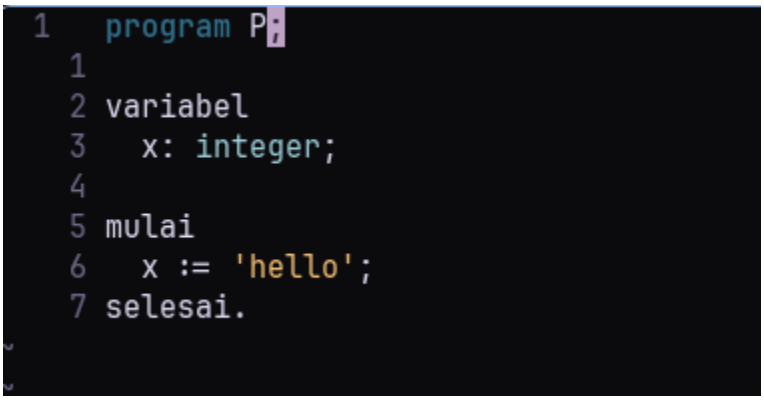
```
program P;

variabel
  x: integer;

mulai
```

```
  x := 'hello';
selesai.
```

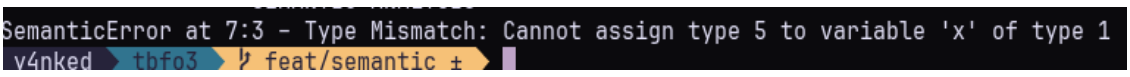
Output

```
SemanticError at 7:3 - Type Mismatch: Cannot assign type 5 to variable 'x' of type 1
```

Bukti Input (Screenshot)



Bukti Output (Screenshot)



3. array_access.pas

Kasus uji ini mencakup kasus yang menguji *symbol table* terutama atab dalam mengidentifikasi keberadaan *array* dengan tepat, juga *visitor* dalam mengidentifikasi akses *array*.

Input

```
program P;

variabel
  arr: larik[1..3] dari integer;

mulai
  arr[2] := 10;
selesai.
```

Output

```
tabs:
idx   id              obj           type    ref     nrm   lev
adr     link
-------------------------------------------------------------
-----------------
1     integer         type          1       0       1     0
0       0
2     boolean         type          2       0       1     0
0       0
3     char            type          3       0       1     0
0       0
4     real            type          4       0       1     0
0       0
5     string          type          5       0       1     0
0       0
6     array           type          6       0       1     0
0       0
7     false           constant      2       0       1     0
0       0
8     true            constant      2       0       1     0
1       0
9     abs             function      0       0       1     0
0       0
10    sqr             function      0       0       1     0
0       0
11    odd             function      2       0       1     0
0       0
12    chr             function      3       0       1     0
0       0
13    ord             function      1       0       1     0
0       0
14    succ            function      0       0       1     0
0       0
15    pred            function      0       0       1     0
0       0
16    round           function      1       0       1     0
0       0
17    trunc           function      1       0       1     0
0       0
18    sin             function      4       0       1     0
0       0
19    cos             function      4       0       1     0
0       0
20    exp             function      4       0       1     0
0       0
21    ln              function      4       0       1     0
0       0
22    sqrt            function      4       0       1     0
0       0
23    arctan          function      4       0       1     0
0       0
```

```
24    eof               function      2       0       1      0
0       0
25    eoln              function      2       0       1      0
0       0
26    write             procedure     0       0       1      0
0       0
27    writeln           procedure     0       0       1      0
0       0
28    read              procedure     0       0       1      0
0       0
29    readln            procedure     0       0       1      0
0       0
31    P                 program       0       0       1      0
0       0
32    arr               variable      6       0       1      0
0       31

btab:
idx   last    lpar    psze    vsze
-----------------------------------
0     32      0       0       1
1     0       0       0       0

atab:
idx   xtyp    etyp    low     high    elsz    size
-----------------------------------------------------
0     1       1       1       3       1       3

Decorated AST:
└─ ProgramNode('P')
   ├─ VarDeclNode('arr') → tab_index:32, lev:0
   └─ Block → block_index:1
      └─ AssignNode Num(10)
         ├─ ArrayAccessNode('arr') → tab_index:32,
type:integer
         │  └─ NumberNode 2 → type:integer
         └─ NumberNode 10 → type:integer
```

Bukti Input (Screenshot)

```
1   program P;
  1
  2 variabel
  3   arr: larik[1..3] dari integer;
  4
  5 mulai
  6   arr[2] := 10;
  7 selesai.
```

Bukti Output (Screenshot)

```
tabs:
idx   id              obj          type    ref     nrm   lev   adr     link
-------------------------------------------------------------------------------
1     integer         type         1       0       1     0     0       0
2     boolean         type         2       0       1     0     0       0
3     char            type         3       0       1     0     0       0
4     real            type         4       0       1     0     0       0
5     string          type         5       0       1     0     0       0
6     array           type         6       0       1     0     0       0
7     false           constant     2       0       1     0     0       0
8     true            constant     2       0       1     0     1       0
9     abs             function     0       0       1     0     0       0
10    sqr             function     0       0       1     0     0       0
11    odd             function     2       0       1     0     0       0
12    chr             function     3       0       1     0     0       0
13    ord             function     1       0       1     0     0       0
14    succ            function     0       0       1     0     0       0
15    pred            function     0       0       1     0     0       0
16    round           function     1       0       1     0     0       0
17    trunc           function     1       0       1     0     0       0
18    sin             function     4       0       1     0     0       0
19    cos             function     4       0       1     0     0       0
20    exp             function     4       0       1     0     0       0
21    ln              function     4       0       1     0     0       0
22    sqrt            function     4       0       1     0     0       0
23    arctan          function     4       0       1     0     0       0
24    eof             function     2       0       1     0     0       0
25    eoln            function     2       0       1     0     0       0
26    write           procedure    0       0       1     0     0       0
27    writeln         procedure    0       0       1     0     0       0
28    read            procedure    0       0       1     0     0       0
29    readln          procedure    0       0       1     0     0       0
31    P               program      0       0       1     0     0       0
32    arr             variable     6       0       1     0     0       31

btab:
idx   last    lpar    psze    vsze
-----------------------------------
0     32      0       0       1
1     0       0       0       0

atab:
idx   xtyp    etyp    low     high    elsz    size
-----------------------------------------------------
0     1       1       1       3       1       3

Decorated AST:
└─ ProgramNode('P')
   ├─ VarDeclNode('arr') → tab_index:32, lev:0
   └─ Block → block_index:1
      └─ AssignNode Num(10)
         ├─ ArrayAccessNode('arr') → tab_index:32, type:integer
         │  └─ NumberNode 2 → type:integer
         └─ NumberNode 10 → type:integer
```

4. procedure_call.pas

Kasus uji ini mencakup kasus yang menguji *analyzer* dalam mencari *reserved procedure* (writeln) dan menangani daftar argumen sebagai parameter..

Input

```
program P;

variabel
  i: integer;

mulai
  writeln('x = ', i);
selesai.
```

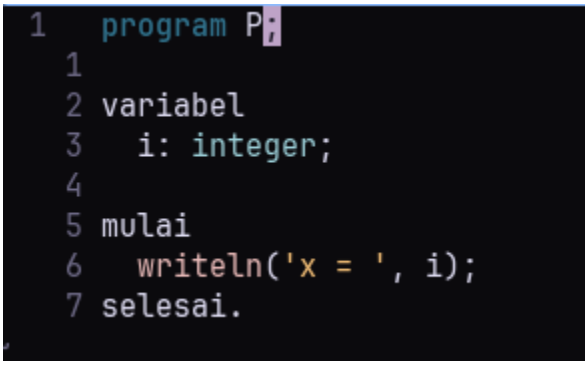
Output

```
tabs:
idx   id              obj           type    ref     nrm   lev
adr     link
-------------------------------------------------------------------
-----------------
1     integer         type          1       0       1     0
0       0
2     boolean         type          2       0       1     0
0       0
3     char            type          3       0       1     0
0       0
4     real            type          4       0       1     0
0       0
5     string          type          5       0       1     0
0       0
6     array           type          6       0       1     0
0       0
7     false           constant      2       0       1     0
0       0
8     true            constant      2       0       1     0
1       0
9     abs             function      0       0       1     0
0       0
10    sqr             function      0       0       1     0
0       0
11    odd             function      2       0       1     0
0       0
12    chr             function      3       0       1     0
0       0
13    ord             function      1       0       1     0
0       0
```

```
14    succ              function      0       0       1     0
0       0
15    pred              function      0       0       1     0
0       0
16    round             function      1       0       1     0
0       0
17    trunc             function      1       0       1     0
0       0
18    sin               function      4       0       1     0
0       0
19    cos               function      4       0       1     0
0       0
20    exp               function      4       0       1     0
0       0
21    ln                function      4       0       1     0
0       0
22    sqrt              function      4       0       1     0
0       0
23    arctan            function      4       0       1     0
0       0
24    eof               function      2       0       1     0
0       0
25    eoln              function      2       0       1     0
0       0
26    write             procedure     0       0       1     0
0       0
27    writeln           procedure     0       0       1     0
0       0
28    read              procedure     0       0       1     0
0       0
29    readln            procedure     0       0       1     0
0       0
31    P                 program       0       0       1     0
0       0
32    i                 variable      1       0       1     0
0       31

btab:
idx   last    lpar    psze    vsze
-----------------------------------
0     32      0       0       1
1     0       0       0       0

atab:
(empty because no array)

Decorated AST:
└─ ProgramNode('P')
   ├─ VarDeclNode('i') → tab_index:32, lev:0
   └─ Block → block_index:1
      └─ ProcCallNode('writeln') → type:void/none
```

```
        ├─ StringNode x =  → type:string
        └─ VarNode('i') → tab_index:32, type:integer
```

Bukti Input (Screenshot)



```
1  program P;
1
2  variabel
3    i: integer;
4
5  mulai
6    writeln('x = ', i);
7  selesai.
```

Bukti Output (Screenshot)

```
tabs:
idx   id              obj           type    ref     nrm   lev   adr     link
-------------------------------------------------------------------------------
1     integer         type          1       0       1     0     0       0
2     boolean         type          2       0       1     0     0       0
3     char            type          3       0       1     0     0       0
4     real            type          4       0       1     0     0       0
5     string          type          5       0       1     0     0       0
6     array           type          6       0       1     0     0       0
7     false           constant      2       0       1     0     0       0
8     true            constant      2       0       1     0     1       0
9     abs             function      0       0       1     0     0       0
10    sqr             function      0       0       1     0     0       0
11    odd             function      2       0       1     0     0       0
12    chr             function      3       0       1     0     0       0
13    ord             function      1       0       1     0     0       0
14    succ            function      0       0       1     0     0       0
15    pred            function      0       0       1     0     0       0
16    round           function      1       0       1     0     0       0
17    trunc           function      1       0       1     0     0       0
18    sin             function      4       0       1     0     0       0
19    cos             function      4       0       1     0     0       0
20    exp             function      4       0       1     0     0       0
21    ln              function      4       0       1     0     0       0
22    sqrt            function      4       0       1     0     0       0
23    arctan          function      4       0       1     0     0       0
24    eof             function      2       0       1     0     0       0
25    eoln            function      2       0       1     0     0       0
26    write           procedure     0       0       1     0     0       0
27    writeln         procedure     0       0       1     0     0       0
28    read            procedure     0       0       1     0     0       0
29    readln          procedure     0       0       1     0     0       0
31    P               program       0       0       1     0     0       0
32    i               variable      1       0       1     0     0       31

btab:
idx   last    lpar    psze    vsze
-----------------------------------
0     32      0       0       1
1     0       0       0       0

atab:
(empty because no array)

Decorated AST:
└─ ProgramNode('P')
   ├─ VarDeclNode('i') → tab_index:32, lev:0
   └─ Block → block_index:1
      └─ ProcCallNode('writeln') → type:void/none
         ├─ StringNode x =  → type:string
         └─ VarNode('i') → tab_index:32, type:integer
```

5. undeclared_identifier.pas

Kasus uji ini mencakup kasus yang menguji *analyzer* dalam melakukan *lookup identifier* dan melempar *error* yang informatif..

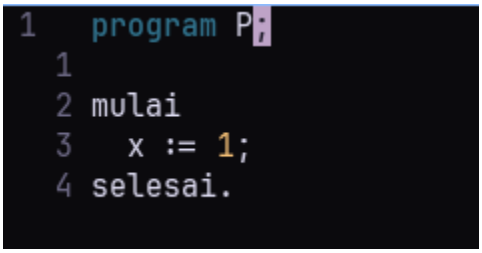
Input

```
program P;

mulai
  x := 1;
selesai.
```

Output

```
SemanticError at 4:3 – Identifier 'x' undeclared.
```

Bukti Input (Screenshot)



Bukti Output (Screenshot)

6. nested_procedure.pas

Kasus uji ini mencakup kasus yang menguji *symbol table* terutama btab dalam identifikasi *scope* pada prosedur bersarang..

Input

```
program test_proc_nested;

variabel
  g: integer;
  global_flag: boolean;

prosedur A(p: integer);
```

```
variabel
  x: integer;

  prosedur B(c: integer);
  variabel
    y: integer;
  mulai
    y := p + 1;
  selesai;

mulai
  x := p + g;
  B(x);
selesai;

mulai
  g := 10;
  A(5);
selesai.
```

Output

```
tabs:
idx   id              obj           type    ref     nrm   lev
adr     link
-------------------------------------------------------------
-----------------
1     integer         type          1       0       1     0
0       0
2     boolean         type          2       0       1     0
0       0
3     char            type          3       0       1     0
0       0
4     real            type          4       0       1     0
0       0
5     string          type          5       0       1     0
0       0
6     array           type          6       0       1     0
0       0
7     false           constant      2       0       1     0
0       0
8     true            constant      2       0       1     0
1       0
9     abs             function      0       0       1     0
0       0
10    sqr             function      0       0       1     0
0       0
```

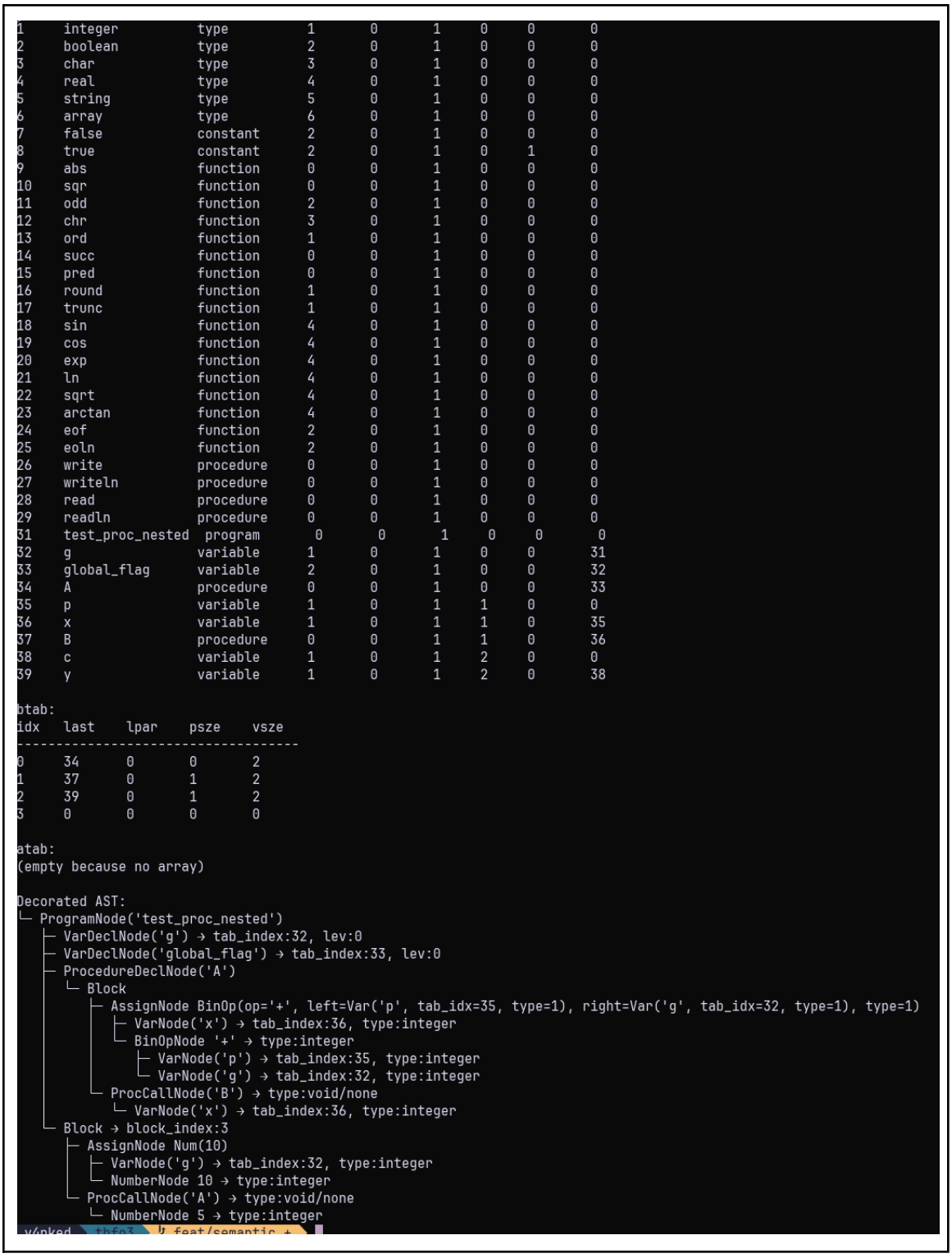
```
11    odd               function      2       0       1     0
0       0
12    chr               function      3       0       1     0
0       0
13    ord               function      1       0       1     0
0       0
14    succ              function      0       0       1     0
0       0
15    pred              function      0       0       1     0
0       0
16    round             function      1       0       1     0
0       0
17    trunc             function      1       0       1     0
0       0
18    sin               function      4       0       1     0
0       0
19    cos               function      4       0       1     0
0       0
20    exp               function      4       0       1     0
0       0
21    ln                function      4       0       1     0
0       0
22    sqrt              function      4       0       1     0
0       0
23    arctan            function      4       0       1     0
0       0
24    eof               function      2       0       1     0
0       0
25    eoln              function      2       0       1     0
0       0
26    write             procedure     0       0       1     0
0       0
27    writeln           procedure     0       0       1     0
0       0
28    read              procedure     0       0       1     0
0       0
29    readln            procedure     0       0       1     0
0       0
31    test_proc_nested  program       0       0       1     0
0       0
32    g                 variable      1       0       1     0
0       31
33    global_flag       variable      2       0       1     0
0       32
34    A                 procedure     0       0       1     0
0       33
35    p                 variable      1       0       1     1
0       0
36    x                 variable      1       0       1     1
0       35
37    B                 procedure     0       0       1     1
```

```
0       36
38    c                 variable      1       0       1     2
0       0
39    y                 variable      1       0       1     2
0       38

btab:
idx   last    lpar    psze    vsze
-----------------------------------
0     34      0       0       2
1     37      0       1       2
2     39      0       1       2
3     0       0       0       0

atab:
(empty because no array)

Decorated AST:
└─ ProgramNode('test_proc_nested')
   ├─ VarDeclNode('g') → tab_index:32, lev:0
   ├─ VarDeclNode('global_flag') → tab_index:33, lev:0
   ├─ ProcedureDeclNode('A')
   │  └─ Block
   │     ├─ AssignNode BinOp(op='+', left=Var('p', tab_idx=35,
type=1), right=Var('g', tab_idx=32, type=1), type=1)
   │     │  ├─ VarNode('x') → tab_index:36, type:integer
   │     │  └─ BinOpNode '+' → type:integer
   │     │     ├─ VarNode('p') → tab_index:35, type:integer
   │     │     └─ VarNode('g') → tab_index:32, type:integer
   │     └─ ProcCallNode('B') → type:void/none
   │        └─ VarNode('x') → tab_index:36, type:integer
   └─ Block → block_index:3
      ├─ AssignNode Num(10)
      │  ├─ VarNode('g') → tab_index:32, type:integer
      │  └─ NumberNode 10 → type:integer
      └─ ProcCallNode('A') → type:void/none
         └─ NumberNode 5 → type:integer
```

Bukti Input (Screenshot)

```
program test_proc_nested;

variabel
  g: integer;
  global_flag: boolean;

prosedur A(p: integer);
variabel
  x: integer;

  prosedur B(c: integer);
  variabel
    y: integer;
  mulai
    y := p + 1;
  selesai;

mulai
  x := p + g;
  B(x);
selesai;

mulai
  g := 10;
  A(5);
selesai.
```

Bukti Output (Screenshot)



```
1     integer          type          1       0       1     0     0       0
2     boolean          type          2       0       1     0     0       0
3     char             type          3       0       1     0     0       0
4     real             type          4       0       1     0     0       0
5     string           type          5       0       1     0     0       0
6     array            type          6       0       1     0     0       0
7     false            constant      2       0       1     0     0       0
8     true             constant      2       0       1     0     1       0
9     abs              function      0       0       1     0     0       0
10    sqr              function      0       0       1     0     0       0
11    odd              function      2       0       1     0     0       0
12    chr              function      3       0       1     0     0       0
13    ord              function      1       0       1     0     0       0
14    succ             function      0       0       1     0     0       0
15    pred             function      0       0       1     0     0       0
16    round            function      1       0       1     0     0       0
17    trunc            function      1       0       1     0     0       0
18    sin              function      4       0       1     0     0       0
19    cos              function      4       0       1     0     0       0
20    exp              function      4       0       1     0     0       0
21    ln               function      4       0       1     0     0       0
22    sqrt             function      4       0       1     0     0       0
23    arctan           function      4       0       1     0     0       0
24    eof              function      2       0       1     0     0       0
25    eoln             function      2       0       1     0     0       0
26    write            procedure     0       0       1     0     0       0
27    writeln          procedure     0       0       1     0     0       0
28    read             procedure     0       0       1     0     0       0
29    readln           procedure     0       0       1     0     0       0
31    test_proc_nested  program       0       0       1     0     0       0
32    g                variable      1       0       1     0     0       31
33    global_flag      variable      2       0       1     0     0       32
34    A                procedure     0       0       1     0     0       33
35    p                variable      1       0       1     1     0       0
36    x                variable      1       0       1     1     0       35
37    B                procedure     0       0       1     1     0       36
38    c                variable      1       0       1     2     0       0
39    y                variable      1       0       1     2     0       38

btab:
idx   last    lpar    psze    vsze
-----------------------------------
0     34      0       0       2
1     37      0       1       2
2     39      0       1       2
3     0       0       0       0

atab:
(empty because no array)

Decorated AST:
└─ ProgramNode('test_proc_nested')
   ├─ VarDeclNode('g') → tab_index:32, lev:0
   ├─ VarDeclNode('global_flag') → tab_index:33, lev:0
   ├─ ProcedureDeclNode('A')
   │  └─ Block
   │     ├─ AssignNode BinOp(op='+', left=Var('p', tab_idx=35, type=1), right=Var('g', tab_idx=32, type=1), type=1)
   │     │  ├─ VarNode('x') → tab_index:36, type:integer
   │     │  └─ BinOpNode '+' → type:integer
   │     │     ├─ VarNode('p') → tab_index:35, type:integer
   │     │     └─ VarNode('g') → tab_index:32, type:integer
   │     └─ ProcCallNode('B') → type:void/none
   │        └─ VarNode('x') → tab_index:36, type:integer
   └─ Block → block_index:3
      ├─ AssignNode Num(10)
      │  ├─ VarNode('g') → tab_index:32, type:integer
      │  └─ NumberNode 10 → type:integer
      └─ ProcCallNode('A') → type:void/none
         └─ NumberNode 5 → type:integer
```

7. all.pas

Kasus uji ini mencakup kasus yang menguji *analyzer* dalam menganalisis program yang kompleks.

Input

```
program test_full_tree;

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

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

variabel
  x, y, z: integer;
  sum: Real;
  avg: Real;
  count: integer;
  c: char;
  s: string;
  arr: larik[1..5] dari Real;
  done: boolean;

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

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

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

  done := false;

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

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

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

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

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

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

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

  writeln(greet);
  x := add(10, 20);
```

```
selesai.
```

Output

```
tabs:
idx   id              obj           type    ref     nrm   lev
adr     link
-------------------------------------------------------------
-----------------
1     integer         type          1       0       1     0
0       0
2     boolean         type          2       0       1     0
0       0
3     char            type          3       0       1     0
0       0
4     real            type          4       0       1     0
0       0
5     string          type          5       0       1     0
0       0
6     array           type          6       0       1     0
0       0
7     false           constant      2       0       1     0
0       0
8     true            constant      2       0       1     0
1       0
9     abs             function      0       0       1     0
0       0
10    sqr             function      0       0       1     0
0       0
11    odd             function      2       0       1     0
0       0
12    chr             function      3       0       1     0
0       0
13    ord             function      1       0       1     0
0       0
14    succ            function      0       0       1     0
0       0
15    pred            function      0       0       1     0
0       0
16    round           function      1       0       1     0
0       0
17    trunc           function      1       0       1     0
0       0
18    sin             function      4       0       1     0
0       0
19    cos             function      4       0       1     0
0       0
20    exp             function      4       0       1     0
0       0
```

```
21    ln              function      4       0       1       0
0       0
22    sqrt            function      4       0       1       0
0       0
23    arctan          function      4       0       1       0
0       0
24    eof             function      2       0       1       0
0       0
25    eoln            function      2       0       1       0
0       0
26    write           procedure     0       0       1       0
0       0
27    writeln         procedure     0       0       1       0
0       0
28    read            procedure     0       0       1       0
0       0
29    readln          procedure     0       0       1       0
0       0
31    test_full_tree  program       0       0       1       0
0       0
32    maxVal          constant      1       0       1       0
100     31
33    piVal           constant      4       0       1       0
3.14    32
34    greet           constant      5       0       1       0
0       33
35    index           type          1       0       1       0
0       34
36    realArray       type          6       0       1       0
0       35
37    letter          type          3       0       1       0
0       36
38    flag            type          2       0       1       0
0       37
39    koordinat       type          0       0       1       0
0       38
40    x               variable      1       0       1       0
0       39
41    y               variable      1       0       1       0
0       40
42    z               variable      1       0       1       0
0       41
43    sum             variable      4       0       1       0
0       42
44    avg             variable      4       0       1       0
0       43
45    count           variable      1       0       1       0
0       44
46    c               variable      3       0       1       0
0       45
47    s               variable      5       0       1       0
```

```
0       46
48    arr               variable      6       1       1     0
0       47
49    done              variable      2       0       1     0
0       48
50    ok                procedure     0       0       1     0
0       49
51    msg               variable      3       0       1     1
0       0
52    add               function      1       0       1     0
0       50
53    a                 variable      1       0       1     1
0       0
54    b                 variable      1       0       1     1
0       53

btab:
idx   last    lpar    psze    vsze
-----------------------------------
0     52      0       0       10
1     51      0       1       1
2     54      0       2       2
3     0       0       0       0

atab:
idx   xtyp    etyp    low     high    elsz    size
-----------------------------------------------------
0     1       4       1       5       1       5
1     1       4       1       5       1       5

Decorated AST:
└─ ProgramNode('test_full_tree')
   ├─ ConstDeclNode('maxVal')
   ├─ ConstDeclNode('piVal')
   ├─ ConstDeclNode('greet')
   ├─ TypeDeclNode('index')
   ├─ TypeDeclNode('realArray')
   ├─ TypeDeclNode('letter')
   ├─ TypeDeclNode('flag')
   ├─ TypeDeclNode('koordinat')
   ├─ VarDeclNode('x') → tab_index:40, lev:0
   ├─ VarDeclNode('y') → tab_index:41, lev:0
   ├─ VarDeclNode('z') → tab_index:42, lev:0
   ├─ VarDeclNode('sum') → tab_index:43, lev:0
   ├─ VarDeclNode('avg') → tab_index:44, lev:0
   ├─ VarDeclNode('count') → tab_index:45, lev:0
   ├─ VarDeclNode('c') → tab_index:46, lev:0
   ├─ VarDeclNode('s') → tab_index:47, lev:0
   ├─ VarDeclNode('arr') → tab_index:48, lev:0
   ├─ VarDeclNode('done') → tab_index:49, lev:0
   ├─ ProcedureDeclNode('ok')
```

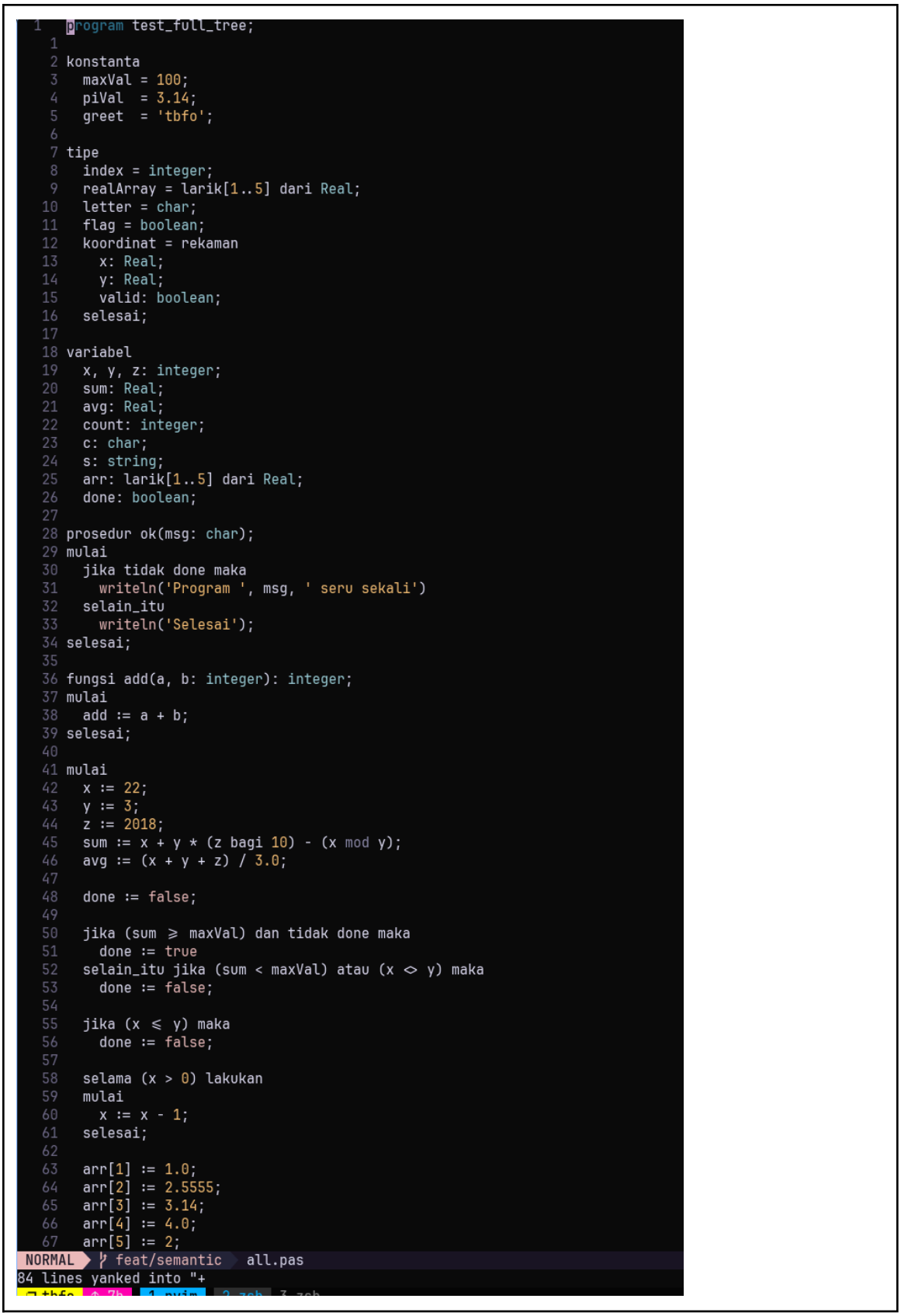
```
    │   └─ Block
    │      └─ IfNode
    │         ├─ UnaryOpNode 'tidak' → type:boolean
    │         │  └─ VarNode('done') → tab_index:49, type:boolean
    │         ├─ ProcCallNode('writeln') → type:void/none
    │         │  ├─ StringNode Program  → type:string
    │         │  ├─ VarNode('msg') → tab_index:51, type:char
    │         │  └─ StringNode  seru sekali → type:string
    │         └─ ProcCallNode('writeln') → type:void/none
    │            └─ StringNode Selesai → type:string
    ├─ FunctionDeclNode('add')
    │   └─ Block
    │      └─ AssignNode BinOp(op='+', left=Var('a', tab_idx=53,
type=1), right=Var('b', tab_idx=54, type=1), type=1)
    │         ├─ VarNode('add') → tab_index:52, type:integer
    │         └─ BinOpNode '+' → type:integer
    │            ├─ VarNode('a') → tab_index:53, type:integer
    │            └─ VarNode('b') → tab_index:54, type:integer
    │
    └─ Block → block_index:3
       ├─ AssignNode Num(22)
       │  ├─ VarNode('x') → tab_index:40, type:integer
       │  └─ NumberNode 22 → type:integer
       ├─ AssignNode Num(3)
       │  ├─ VarNode('y') → tab_index:41, type:integer
       │  └─ NumberNode 3 → type:integer
       ├─ AssignNode Num(2018)
       │  ├─ VarNode('z') → tab_index:42, type:integer
       │  └─ NumberNode 2018 → type:integer
       ├─ AssignNode BinOp(op='-', left=BinOp(op='+',
left=Var('x', tab_idx=40, type=1), right=BinOp(op='*',
left=Var('y', tab_idx=41, type=1), right=BinOp(op='bagi',
left=Var('z', tab_idx=42, type=1), right=Num(10), type=4),
type=4), type=4), right=BinOp(op='mod', left=Var('x',
tab_idx=40, type=1), right=Var('y', tab_idx=41, type=1),
type=1), type=4)
       │  ├─ VarNode('sum') → tab_index:43, type:real
       │  └─ BinOpNode '-' → type:real
       │     ├─ BinOpNode '+' → type:real
       │     │  ├─ VarNode('x') → tab_index:40, type:integer
       │     │  └─ BinOpNode '*' → type:real
       │     │     ├─ VarNode('y') → tab_index:41, type:integer
       │     │     └─ BinOpNode 'bagi' → type:real
       │     │        ├─ VarNode('z') → tab_index:42,
type:integer
       │     │        └─ NumberNode 10 → type:integer
       │     └─ BinOpNode 'mod' → type:integer
       │        ├─ VarNode('x') → tab_index:40, type:integer
       │        └─ VarNode('y') → tab_index:41, type:integer
       ├─ AssignNode BinOp(op='/', left=BinOp(op='+',
left=BinOp(op='+', left=Var('x', tab_idx=40, type=1),
right=Var('y', tab_idx=41, type=1), type=1), right=Var('z',
```

```
tab_idx=42, type=1), type=1), right=Num(3.0), type=4)
       │   ├─ VarNode('avg') → tab_index:44, type:real
       │   └─ BinOpNode '/' → type:real
       │      ├─ BinOpNode '+' → type:integer
       │      │   ├─ BinOpNode '+' → type:integer
       │      │   │   ├─ VarNode('x') → tab_index:40, type:integer
       │      │   │   └─ VarNode('y') → tab_index:41, type:integer
       │      │   └─ VarNode('z') → tab_index:42, type:integer
       │      └─ NumberNode 3.0 → type:real
       ├─ AssignNode Bool(False)
       │   ├─ VarNode('done') → tab_index:49, type:boolean
       │   └─ BooleanNode False → type:boolean
       ├─ IfNode
       │   ├─ BinOpNode 'dan' → type:boolean
       │   │   ├─ BinOpNode '>=' → type:boolean
       │   │   │   ├─ VarNode('sum') → tab_index:43, type:real
       │   │   │   └─ NumberNode 100 → type:integer
       │   │   └─ UnaryOpNode 'tidak' → type:boolean
       │   │      └─ VarNode('done') → tab_index:49, type:boolean
       │   ├─ AssignNode Bool(True)
       │   │   ├─ VarNode('done') → tab_index:49, type:boolean
       │   │   └─ BooleanNode True → type:boolean
       │   └─ IfNode
       │      ├─ BinOpNode 'atau' → type:boolean
       │      │   ├─ BinOpNode '<' → type:boolean
       │      │   │   ├─ VarNode('sum') → tab_index:43, type:real
       │      │   │   └─ NumberNode 100 → type:integer
       │      │   └─ BinOpNode '<>' → type:boolean
       │      │      ├─ VarNode('x') → tab_index:40, type:integer
       │      │      └─ VarNode('y') → tab_index:41, type:integer
       │      └─ AssignNode Bool(False)
       │         ├─ VarNode('done') → tab_index:49, type:boolean
       │         └─ BooleanNode False → type:boolean
       ├─ IfNode
       │   ├─ BinOpNode '<=' → type:boolean
       │   │   ├─ VarNode('x') → tab_index:40, type:integer
       │   │   └─ VarNode('y') → tab_index:41, type:integer
       │   └─ AssignNode Bool(False)
       │      ├─ VarNode('done') → tab_index:49, type:boolean
       │      └─ BooleanNode False → type:boolean
       ├─ WhileNode
       │   ├─ BinOpNode '>' → type:boolean
       │   │   ├─ VarNode('x') → tab_index:40, type:integer
       │   │   └─ NumberNode 0 → type:integer
       │   └─ Block
       │      └─ AssignNode BinOp(op='-', left=Var('x',
tab_idx=40, type=1), right=Num(1), type=1)
       │         ├─ VarNode('x') → tab_index:40, type:integer
       │         └─ BinOpNode '-' → type:integer
       │            ├─ VarNode('x') → tab_index:40, type:integer
       │            └─ NumberNode 1 → type:integer
       │
```

```
      ├─ AssignNode Num(1.0)
      │   ├─ ArrayAccessNode('arr') → tab_index:48, type:real
      │   │   └─ NumberNode 1 → type:integer
      │   └─ NumberNode 1.0 → type:real
      ├─ AssignNode Num(2.5555)
      │   ├─ ArrayAccessNode('arr') → tab_index:48, type:real
      │   │   └─ NumberNode 2 → type:integer
      │   └─ NumberNode 2.5555 → type:real
      ├─ AssignNode Num(3.14)
      │   ├─ ArrayAccessNode('arr') → tab_index:48, type:real
      │   │   └─ NumberNode 3 → type:integer
      │   └─ NumberNode 3.14 → type:real
      ├─ AssignNode Num(4.0)
      │   ├─ ArrayAccessNode('arr') → tab_index:48, type:real
      │   │   └─ NumberNode 4 → type:integer
      │   └─ NumberNode 4.0 → type:real
      ├─ AssignNode Num(2)
      │   ├─ ArrayAccessNode('arr') → tab_index:48, type:real
      │   │   └─ NumberNode 5 → type:integer
      │   └─ NumberNode 2 → type:integer
      ├─ AssignNode Char('a')
      │   ├─ VarNode('c') → tab_index:46, type:char
      │   └─ CharNode a → type:char
      ├─ AssignNode Char('b')
      │   ├─ VarNode('c') → tab_index:46, type:char
      │   └─ CharNode b → type:char
      ├─ AssignNode Char('c')
      │   ├─ VarNode('c') → tab_index:46, type:char
      │   └─ CharNode c → type:char
      ├─ AssignNode String('seru sekali')
      │   ├─ VarNode('s') → tab_index:47, type:string
      │   └─ StringNode seru sekali → type:string
      ├─ ForNode
      │   ├─ NumberNode 1 → type:integer
      │   ├─ NumberNode 5 → type:integer
      │   ├─ AssignNode BinOp(op='+', left=Var('sum',
tab_idx=43, type=4), right=Var('count', tab_idx=45, type=1),
type=4)
      │   │   ├─ VarNode('sum') → tab_index:43, type:real
      │   │   └─ BinOpNode '+' → type:real
      │   │      ├─ VarNode('sum') → tab_index:43, type:real
      │   │      └─ VarNode('count') → tab_index:45,
type:integer
      │   └─ VarNode('count') → tab_index:45
      ├─ ForNode
      │   ├─ NumberNode 5 → type:integer
      │   ├─ NumberNode 1 → type:integer
      │   ├─ AssignNode BinOp(op='+', left=Var('avg',
tab_idx=44, type=4), right=ArrayAccess(name='arr',
index=Var('count', tab_idx=45, type=1)), type=4)
      │   │   ├─ VarNode('avg') → tab_index:44, type:real
```

```
      │  │  └─ BinOpNode '+' → type:real
      │  │     ├─ VarNode('avg') → tab_index:44, type:real
      │  │     └─ ArrayAccessNode('arr') → tab_index:48,
type:real
      │  │        └─ VarNode('count') → tab_index:45,
type:integer
      │  └─ VarNode('count') → tab_index:45
      ├─ ProcCallNode('writeln') → type:void/none
      │  └─ VarNode('greet') → tab_index:34, type:string
      └─ AssignNode ProcCall(name='add', args=[Num(10),
Num(20)])
         ├─ VarNode('x') → tab_index:40, type:integer
         └─ ProcCallNode('add') → type:integer
            ├─ NumberNode 10 → type:integer
            └─ NumberNode 20 → type:integer
```

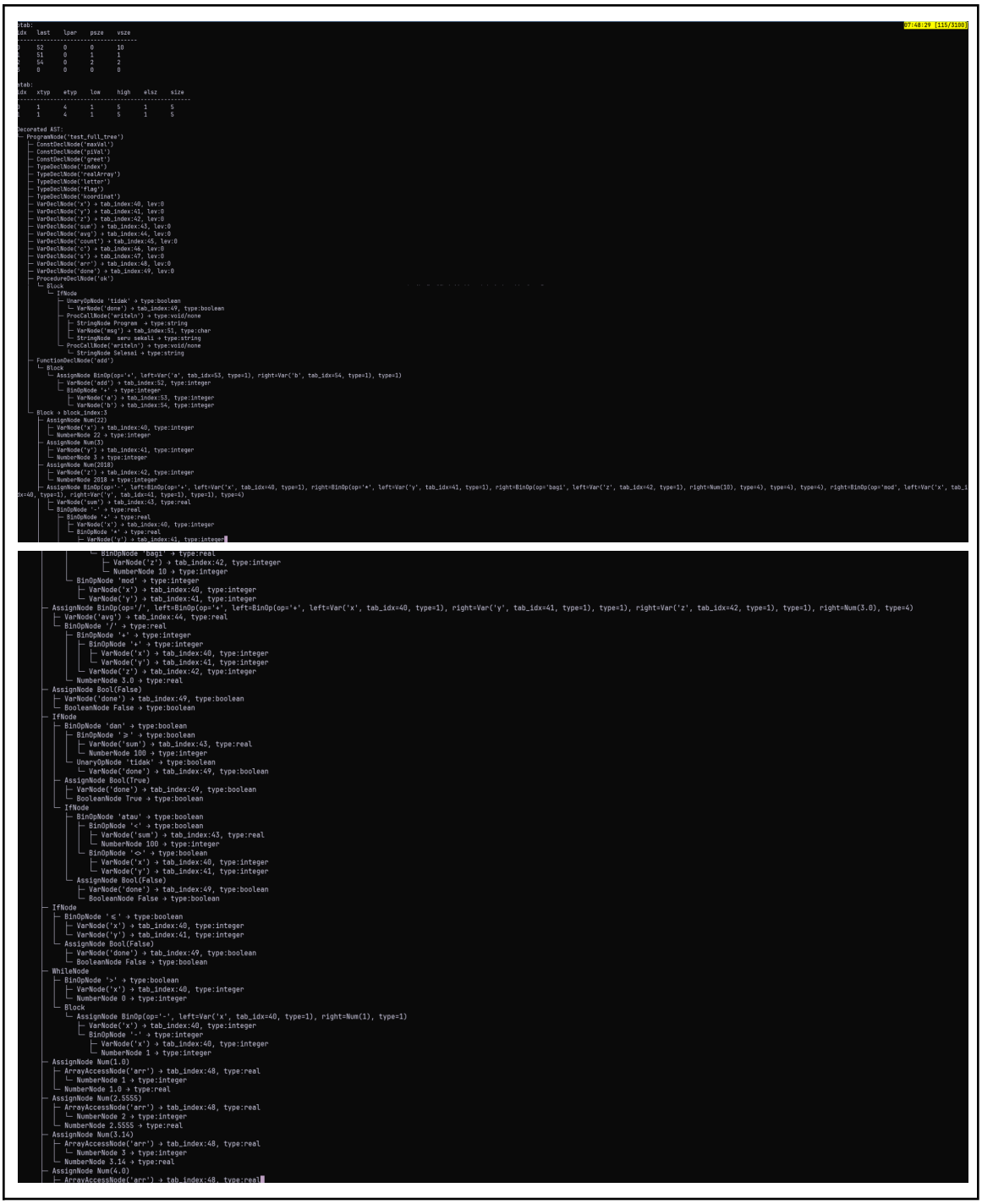
Bukti Input (Screenshot)

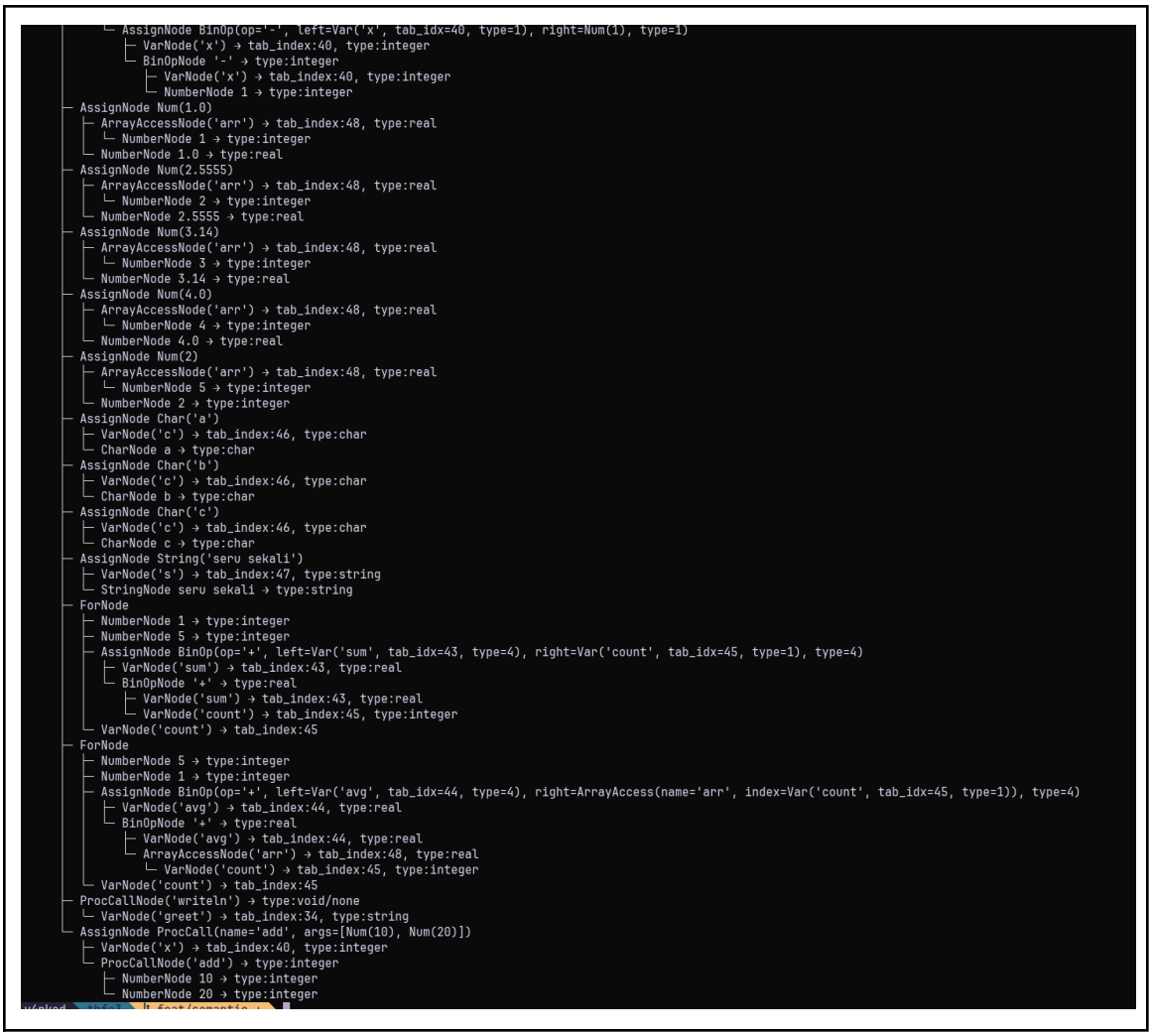


```
program test_full_tree;

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

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

variabel
  x, y, z: integer;
  sum: Real;
  avg: Real;
  count: integer;
  c: char;
  s: string;
  arr: larik[1..5] dari Real;
  done: boolean;

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

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

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

  done := false;

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

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

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

  arr[1] := 1.0;
  arr[2] := 2.5555;
  arr[3] := 3.14;
  arr[4] := 4.0;
  arr[5] := 2;
NORMAL  feat/semantic  all.pas
84 lines yanked into "+
```

Bukti Output (Screenshot)

```
tabs:
idx   id              obj           type    ref     nrm   lev   adr     link
-------------------------------------------------------------------------------
1     integer         type          1       0       1     0     0       0
2     boolean         type          2       0       1     0     0       0
3     char            type          3       0       1     0     0       0
4     real            type          4       0       1     0     0       0
5     string          type          5       0       1     0     0       0
6     array           type          6       0       1     0     0       0
7     false           constant      2       0       1     0     0       0
8     true            constant      2       0       1     0     1       0
9     abs             function      0       0       1     0     0       0
10    sqr             function      0       0       1     0     0       0
11    odd             function      2       0       1     0     0       0
12    chr             function      3       0       1     0     0       0
13    ord             function      1       0       1     0     0       0
14    succ            function      0       0       1     0     0       0
15    pred            function      0       0       1     0     0       0
16    round           function      1       0       1     0     0       0
17    trunc           function      1       0       1     0     0       0
18    sin             function      4       0       1     0     0       0
19    cos             function      4       0       1     0     0       0
20    exp             function      4       0       1     0     0       0
21    ln              function      4       0       1     0     0       0
22    sqrt            function      4       0       1     0     0       0
23    arctan          function      4       0       1     0     0       0
24    eof             function      2       0       1     0     0       0
25    eoln            function      2       0       1     0     0       0
26    write           procedure     0       0       1     0     0       0
27    writeln         procedure     0       0       1     0     0       0
28    read            procedure     0       0       1     0     0       0
29    readln          procedure     0       0       1     0     0       0
31    test_full_tree  program       0       0       1     0     0       0
32    maxVal          constant      1       0       1     0     100     31
33    piVal           constant      4       0       1     0     3.14    32
34    greet           constant      5       0       1     0     0       33
35    index           type          1       0       1     0     0       34
36    realArray       type          6       0       1     0     0       35
37    letter          type          3       0       1     0     0       36
38    flag            type          2       0       1     0     0       37
39    koordinat       type          0       0       1     0     0       38
40    x               variable      1       0       1     0     0       39
41    y               variable      1       0       1     0     0       40
42    z               variable      1       0       1     0     0       41
43    sum             variable      4       0       1     0     0       42
44    avg             variable      4       0       1     0     0       43
45    count           variable      1       0       1     0     0       44
46    c               variable      3       0       1     0     0       45
47    s               variable      5       0       1     0     0       46
48    arr             variable      6       1       1     0     0       47
49    done            variable      2       0       1     0     0       48
50    ok              procedure     0       0       1     0     0       49
51    msg             variable      3       0       1     1     0       0
52    add             function      1       0       1     0     0       50
53    a               variable      1       0       1     1     0       0
54    b               variable      1       0       1     1     0       53

btab:
idx   last    lpar    psze    vsze
-----------------------------------
0     52      0       0       10
1     51      0       1       1
2     54      0       2       2
3     0       0       0       0

atab:
idx   xtyp    etyp    low     high    elsz    size
```


atab:
07:48:29 [115/3100]
idx last lpar psze vsze
52 0 0 10
51 0 1 1
54 0 2 2
0 0 0 0
utab:
idx xtyp etyp low high elsz size
1 4 1 5 1 5
1 4 1 5 1 5
Decorated AST:
ProgramNode('test_full_tree')
ConstDeclNode('maxVal')
ConstDeclNode('piVal')
ConstDeclNode('greet')
TypeDeclNode('index')
TypeDeclNode('realArray')
TypeDeclNode('letter')
TypeDeclNode('flag')
TypeDeclNode('koordinat')
VarDeclNode('x') → tab_index:40, lev:0
VarDeclNode('y') → tab_index:41, lev:0
VarDeclNode('z') → tab_index:42, lev:0
VarDeclNode('sum') → tab_index:43, lev:0
VarDeclNode('avg') → tab_index:44, lev:0
VarDeclNode('count') → tab_index:45, lev:0
VarDeclNode('c') → tab_index:46, lev:0
VarDeclNode('c') → tab_index:47, lev:0
VarDeclNode('arr') → tab_index:48, lev:0
VarDeclNode('done') → tab_index:49, lev:0
ProcedureDeclNode('ok')
Block
IfNode
UnaryOpNode 'tidak' → type:boolean
VarNode('done') → tab_index:49, type:boolean
ProcCallNode('writeln') → type:void/none
StringNode Program → type:string
VarNode('msg') → tab_index:51, type:char
StringNode saru sakali → type:string
ProcCallNode('writeln') → type:void/none
StringNode Selesai → type:string
FunctionDeclNode('add')
Block
AssignNode BinOp(op='+', left=Var('a', tab_idx=53, type=1), right=Var('b', tab_idx=54, type=1), type=1)
VarNode('add') → tab_index:52, type:integer
BinOpNode '+' → type:integer
VarNode('a') → tab_index:53, type:integer
VarNode('b') → tab_index:54, type:integer
Block → block_index:3
AssignNode Num(22)
VarNode('x') → tab_index:40, type:integer
NumberNode 22 → type:integer
AssignNode Num(3)
VarNode('y') → tab_index:41, type:integer
NumberNode 3 → type:integer
AssignNode Num(2018)
VarNode('z') → tab_index:42, type:integer
NumberNode 2018 → type:integer
AssignNode BinOp(op='-', left=BinOp(op='+', left=Var('x', tab_idx=40, type=1), right=BinOp(op='*', left=Var('y', tab_idx=41, type=1), right=BinOp(op='bagi', left=Var('z', tab_idx=42, type=1), right=Num(10), type=4), type=4), type=4), right=BinOp(op='mod', left=Var('x', tab_idx=40, type=1), right=Var('y', tab_idx=41, type=1), type=1), type=4)
VarNode('sum') → tab_index:43, type:real
BinOpNode '-' → type:real
BinOpNode '+' → type:real
VarNode('x') → tab_index:40, type:integer
BinOpNode '*' → type:real
VarNode('y') → tab_index:41, type:integer
BinOpNode 'bagi' → type:real
VarNode('z') → tab_index:42, type:integer
NumberNode 10 → type:integer
BinOpNode 'mod' → type:integer
VarNode('x') → tab_index:40, type:integer
VarNode('y') → tab_index:41, type:integer
AssignNode BinOp(op='/', left=BinOp(op='+', left=BinOp(op='+', left=Var('x', tab_idx=40, type=1), right=Var('y', tab_idx=41, type=1), type=1), right=Var('z', tab_idx=42, type=1), type=1), right=Num(3.0), type=4)
VarNode('avg') → tab_index:44, type:real
BinOpNode '/' → type:real
BinOpNode '+' → type:integer
BinOpNode '+' → type:integer
VarNode('x') → tab_index:40, type:integer
VarNode('y') → tab_index:41, type:integer
VarNode('z') → tab_index:42, type:integer
NumberNode 3.0 → type:real
AssignNode Bool(False)
VarNode('done') → tab_index:49, type:boolean
BooleanNode False → type:boolean
IfNode
BinOpNode 'dan' → type:boolean
BinOpNode '≥' → type:boolean
VarNode('sum') → tab_index:43, type:real
NumberNode 100 → type:integer
UnaryOpNode 'tidak' → type:boolean
VarNode('done') → tab_index:49, type:boolean
AssignNode Bool(True)
VarNode('done') → tab_index:49, type:boolean
BooleanNode True → type:boolean
IfNode
BinOpNode 'atau' → type:boolean
BinOpNode '<' → type:boolean
VarNode('sum') → tab_index:43, type:real
NumberNode 100 → type:integer
BinOpNode '<>' → type:boolean
VarNode('x') → tab_index:40, type:integer
VarNode('y') → tab_index:41, type:integer
AssignNode Bool(False)
VarNode('done') → tab_index:49, type:boolean
BooleanNode False → type:boolean
IfNode
BinOpNode '≤' → type:boolean
VarNode('x') → tab_index:40, type:integer
VarNode('y') → tab_index:41, type:integer
AssignNode Bool(False)
VarNode('done') → tab_index:49, type:boolean
BooleanNode False → type:boolean
WhileNode
BinOpNode '>' → type:boolean
VarNode('x') → tab_index:40, type:integer
NumberNode 0 → type:integer
Block
AssignNode BinOp(op='-', left=Var('x', tab_idx=40, type=1), right=Num(1), type=1)
VarNode('x') → tab_index:40, type:integer
BinOpNode '-' → type:integer
VarNode('x') → tab_index:40, type:integer
NumberNode 1 → type:integer
AssignNode Num(1.0)
ArrayAccessNode('arr') → tab_index:48, type:real
NumberNode 1 → type:integer
NumberNode 1.0 → type:real
AssignNode Num(2.5555)
ArrayAccessNode('arr') → tab_index:48, type:real
NumberNode 2 → type:integer
NumberNode 2.5555 → type:real
AssignNode Num(3.14)
ArrayAccessNode('arr') → tab_index:48, type:real
NumberNode 3 → type:integer
NumberNode 3.14 → type:real
AssignNode Num(4.0)
ArrayAccessNode('arr') → tab_index:48, type:real



8. not_supported.pas

Kasus uji ini mencakup kasus yang menunjukkan salah satu fitur atau karakteristik pada program Pascal-S yang seharusnya bisa di-*compile* namun pada *compiler* yang kami buat tidak bisa. Hal tersebut dikarenakan *compiler* Pascal-S yang kompleks dan spesifikasi tugas yang tidak menjelaskan secara detail fitur apa saja yang perlu di-*cover*.

Pada kasus ini, kasus menunjukkan ketidakmampuan *parser* dalam menggunakan tipe data buatan sebagai tipe suatu variabel

Input

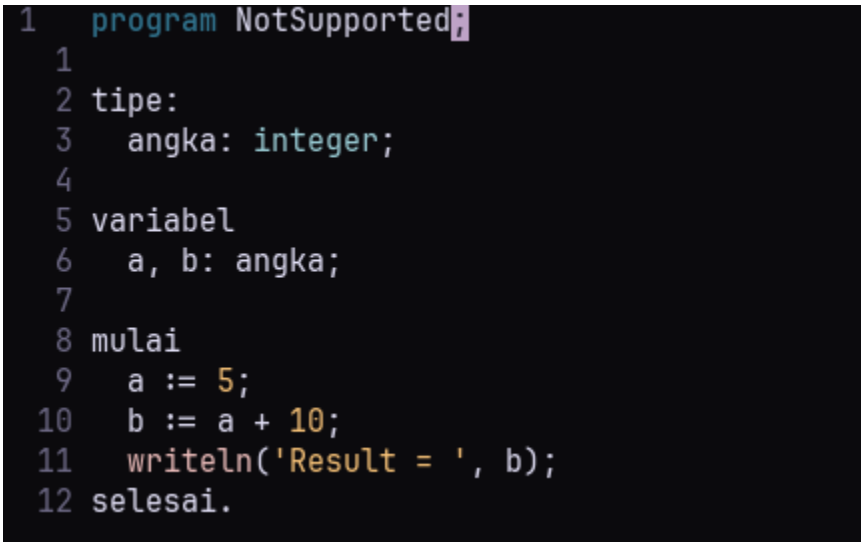
```
program NotSupported;

tipe:
  angka: integer;
```

```
variabel
  a, b: angka;

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

Output

```
SyntaxError at 3:5 - Expected at least one identifier at type declaration
```

Bukti Input (Screenshot)



Bukti Output (Screenshot)

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

Bab V Kesimpulan dan Saran

1. Kesimpulan

Pada milestone ketiga tugas besar ini, *semantic analysis* telah berhasil diimplementasikan yang memproses *parse tree* menjadi *decorated AST* dan mengisi struktur simbol yang diperlukan (tab, btab, atab). Hasil implementasi yang sudah diuji menunjukkan bahwa *analyzer* telah berhasil melakukan *traversal visitor* pada *parse tree* untuk melakukan deklarasi identifier, pembuatan entri tabel simbol, pemeriksaan *scope* leksikal, inferensi dan verifikasi tipe pada ekspresi, serta validasi operasi semantik dasar (*assignment*, pemanggilan prosedur/fungsi, akses *array*, kontrol alur). Hasil keluaran berupa *decorated AST* dan tabel simbol memungkinkan program untuk mendeteksi kesalahan semantik seperti *identifier undeclared*, *type mismatch*, *assign to constant* dengan pelaporan posisi (*line/col*), sehingga *pipeline front-end compiler* menjadi lebih kuat dan siap untuk tahap *back-end* berikutnya.

2. Saran

Untuk milestone berikutnya, disarankan untuk mengembangkan *intermediate code generation* dengan memanfaatkan AST serta *symbol table* yang telah dikembangkan pada milestone ini. Selain itu, perlu dilakukan konsiderasi batasan fitur yang diimplementasikan agar tidak terjadi perubahan kode pada milestone-milestone sebelumnya. Untuk asisten, bisa menyalakan fitur notifikasi perubahan pada *sheets* QnA agar pertanyaan dapat direspons dengan cepat.

Lampiran

1. Pranala Repositori Github

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

Pranala Rilis Milestone 3: Rilis Milestone 3

2. Pembagian Tugas

NIM	Nama	Tugas	Persentase
13523002	Refki Alfarizi	- Definisi AST dan Semantic Analyzer - Laporan	25%
13523028	Muhammad Aditya Rahmadeni	- Menyesuaikan DFA - Error handling - Laporan	25%
13523088	Aryo Bama Wiratama	- Merancang AST Node - *Testing*	25%
13523116	Fityatul Haq Rosyidi	- Menyesuaikan Parser - *Testing*	25%

Referensi

[1] Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compilers: Principles, Techniques*, and Tools. Pearson Education. Diakses pada 26 November 2025 dari https://repository.unikom.ac.id/48769/1/Compilers%20-%20Principles,%20Techniques,%20and%20Tools%20(2006).pdf

[2] Wirth, N. (1976). *PASCAL-S: A Subset and its Implementation*. ETH Zürich. Diakses pada 26 November 2025 dari http://pascal.hansotten.com/uploads/pascals/PASCAL-S%20A%20subset%20and%20its%20Implementation%20012.pdf

[3] Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation (3rd ed.)*. Pearson Education. Diakses pada 27 November 2025 dari https://cdn-edunex.itb.ac.id/29161-Formal-Language-Theory-and-Automata/1629640939613_Introduction-to-Automata-Theory,-Languages,-and-Computation-Edition-3.pdf

[4] *Semantic Analysis.* (2015). University of Liège. Diakses pada 28 November 2025 dari https://people.montefiore.uliege.be/geurts/Cours/compil/2015/04-semantic-2015-2016.pdf

[5] *Semantic analysis (linguistics)*. (2025). Wikipedia, Ensiklopedia Bebas. Diakses pada 27 November 2025 dari https://en.wikipedia.org/wiki/Semantic_analysis_(linguistics)