

**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

Daftar Isi.....	2
Bab I Deskripsi Tugas.....	3
Bab II Landasan Teori.....	12
1. Pascal-S.....	12
2. Compiler.....	12
3. Semantic Analysis.....	12
4. AST (Abstract Syntax Tree).....	13
5. Symbol Table.....	13
Bab III Perancangan dan Implementasi.....	15
1. Kelas Utama pada Semantic Analysis.....	15
2. Struktur Repozitori Program.....	15
3. Struktur AST.....	16
4. Symbol Tables.....	18
5. Visitor.....	20
6. Semantic Rule.....	21
Bab IV Pengujian.....	26
1. program.pas.....	26
2. type_mismatch.pas.....	29
3. array_access.pas.....	30
4. procedure_call.pas.....	35
5. undeclared_identifier.pas.....	39
6. nested_procedure.pas.....	39
7. all.pas.....	44
8. not_supported.pas.....	57
Bab V Kesimpulan dan Saran.....	60
1. Kesimpulan.....	60
2. Saran.....	60
Lampiran.....	61
1. Pranala Repozitori Github.....	61
2. Pembagian Tugas.....	61
Referensi.....	62

# 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*.
  - 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 <i>lexical level</i> 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)

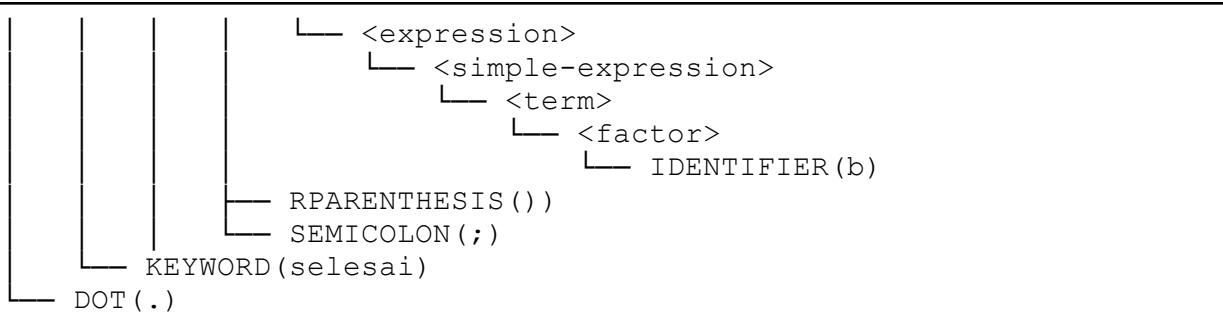
Dengan memiliki informasi pada symbol table, dapat dipastikan konsistensi tipe data serta memastikan variabel tidak bisa dioperasikan jika belum diinisialisasi dengan nilai.

Input Command (contoh yang digunakan menggunakan Python) Format: python [Compiler] [Kode Pascal]
python compiler.py program.pas
Isi program.pas
<pre>program Hello;  variabel   a, b: integer;  mulai   a := 5;   b := a + 10;   writeln('Result = ', b); selesai.</pre>
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
$\langle \text{assignment-statement} \rangle \rightarrow \text{ID} := \langle \text{expr} \rangle$	assignment_statement.node = new AssignNode(new VarNode(ID.lexeme), expr.node)
$\langle \text{expression} \rangle \rightarrow \langle \text{simple-expr} \rangle + \langle \text{term} \rangle$	expression.node = new BinOpNode('+', simple_expr.node, term.node)
$\langle \text{expression} \rangle \rightarrow \langle \text{simple-expr} \rangle$	expression.node = simple_expr.node
$\langle \text{factor} \rangle \rightarrow \text{NUMBER}$	factor.node = new NumberNode(NUMBER.value)
$\langle \text{factor} \rangle \rightarrow \text{ID}$	factor.node = new VarNode(ID.lexeme)
$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle$	term.node = factor.node
$\langle \text{factor} \rangle \rightarrow \text{STRING_LITERAL}$	factor.node = new StringNode(STRING_LITERAL.value)
$\langle \text{procedure-call} \rangle \rightarrow \text{ID} (\langle \text{params} \rangle )$	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 (synthesized) 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
    └─ 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.

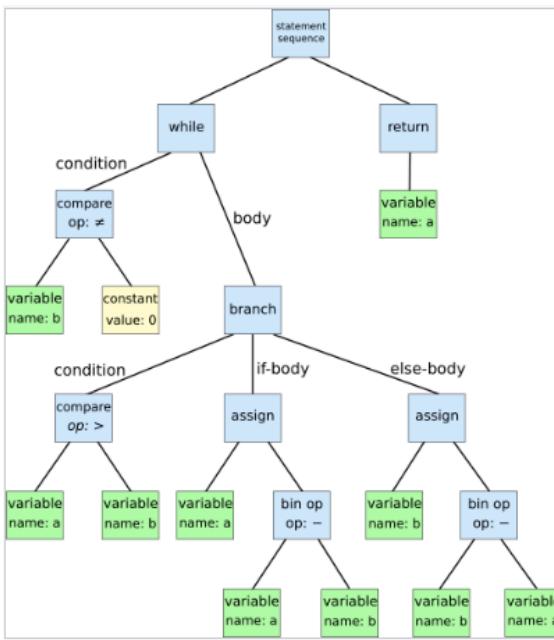
## 3. Semantic Analysis

*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](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 tree*. Setelah AST terbentuk, tahap berikutnya adalah *semantic analysis*, yang dilakukan menggunakan pola *Visitor* untuk menelusuri pohon secara sistematis sekaligus membangun 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 Repozitori 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 <i>scope</i> 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 <i>symbol table</i> 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 <i>return</i>
8	AssignNode	target: VarNode   ArrayAccessNode value	type	Assignment dan <i>semantic check</i> kompatibilitas 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/ko nstanta
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 <i>lookup reserved</i> atau <i>user procedure</i>
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	ArrayClass Node	name: str index_expr	tab_index type	akses elemen <i>array</i>

## 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_declarat	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_declarat	Infer tipe literal; tambah konstanta ke tab (adr = value)	list ConstDeclNode. Error: invalid literal
4	visit_type_declarat	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	$\langle \text{program} \rangle \rightarrow \text{program ID ;}$ $\langle \text{declaration-part} \rangle$ $\langle \text{compound-statement} \rangle .$	program.node = new ProgramNode(ID.lexeme, declarations.node, block.node); add_to_tab(ID.lexeme, "program", 0)
2	$\langle \text{declaration-part} \rangle \rightarrow \{$ $\langle \text{var\_declaration} \rangle \mid$ $\langle \text{const\_declaration} \rangle \mid$ $\langle \text{type\_declaration} \rangle \mid$ $\langle \text{subprogram\_declaration} \rangle$ }	declarations.node = concat(child.nodes) (visit tiap deklarasi)
3	$\langle \text{var\_declaration} \rangle \rightarrow \text{variabel}$ $\langle \text{identifier-list} \rangle :$ $\langle \text{type} \rangle ;$	untuk tiap id $\in$ 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	$\langle \text{const\_declaration} \rangle \rightarrow \text{konstanta ID =}$ $\langle \text{const\_value} \rangle ;$	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", typ e) (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                           type                       2               0               1               0 2  boolean                   type                       3               0               1               0 0  0                           type                       4               0               1               0 3  char                       type                       5               0               1               0 0  0                           type                       6               0               1               0 4  real                       type                       7               0               1               0 0  0                           type                       8               0               1               0 5  string                    type                       9               0               1               0 0  0                           array                       10              0               1               0 6  array                       type                       11              0               1               0 0  0                           type                       12              0               1               0 7  false                      constant               13              0               1               0 0  0                           constant               14              0               1               0 8  true                       constant               15              0               1               0 1  0                           function               16              0               1               0 9  abs                       function               17              0               1               0 0  0                           function               18              0               1               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					
btabs:						
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)

```

1 program Hello;
1 lev:0
2 variabel
3   a, b: integer;
4
5 mulai
6   a := 5; tab_idx=32,
7   b := a + 10;
8   writeln('Result = ', b);
9 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 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

stab:
(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)

```
1 program P;
1
2 variabel
3   x: integer;
4
5 mulai
6   x := 'hello';
7 selesai.
```

#### Bukti Output (Screenshot)

```
SemanticError at 7:3 - Type Mismatch: Cannot assign type 5 to variable 'x' of type 1
yanked ➤ tbfo3 ➤ ↵feat/semantic + ➤ ↵
```

### 3. array\_access.pas

Kasus uji ini mencakup kasus yang menguji *symbol table* terutama atas 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

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

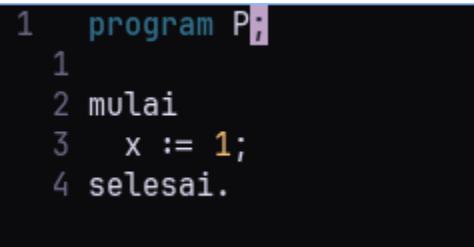
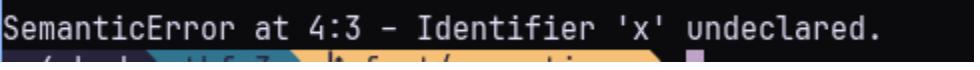
atab:
(empty because no array)

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)
 A screenshot of a code editor window. The code is as follows: 1 program P; 1 2 mulai 3 x := 1; 4 selesai. The identifier 'x' is highlighted in orange, indicating it is undeclared.
Bukti Output (Screenshot)
 A screenshot of a terminal window. The output is: SemanticError at 4:3 - Identifier 'x' undeclared.

## 6. nested\_procedure.pas

Kasus uji ini mencakup kasus yang menguji *symbol table* terutama berdasarkan 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					
<hr/>						
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)

```
1  program test_proc_nested;
2 variabel
3   g: integer;
4   global_flag: boolean;
5
6 prosedur A(p: integer);
7 variabel
8   x: integer;
9
10 prosedur B(c: integer);
11 variabel
12   y: integer;
13 mulai
14   y := p + 1;
15 selesai;
16
17 mulai
18   x := p + g;
19   B(x);
20 selesai;
21
22 mulai
23   g := 10;
24   A(5);
25 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
30 test_proc_nested program  0   0   1   0   0   0
31 g               variable 1   0   1   0   0   31
32 global_flag     variable 2   0   1   0   0   32
33 A               procedure 0   0   1   0   0   33
34 p               variable 1   0   1   1   0   0
35 x               variable 1   0   1   1   0   35
36 B               procedure 0   0   1   1   0   36
37 c               variable 1   0   1   2   0   0
38 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					
<hr/>						
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=Var('x', tab_idx=40, type=1), right=Var('y', tab_idx=41, type=1), type=1), right=Var('z', type=1))

```

```

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)**

```

1 program test_full_tree;
2
3 konstanta
4 maxVal = 100;
5 piVal = 3.14;
6 greet = 'tbfo';
7
8 tipe
9 index = integer;
10 realArray = larik[1..5] dari Real;
11 letter = char;
12 flag = boolean;
13 koordinat = rekaman
14   x: Real;
15   y: Real;
16   valid: boolean;
17 selesai;
18
19 variabel
20 x, y, z: integer;
21 sum: Real;
22 avg: Real;
23 count: integer;
24 c: char;
25 s: string;
26 arr: larik[1..5] dari Real;
27 done: boolean;
28
29 prosedur ok(msg: char);
30 mulai
31 jika tidak done maka
32   writeln('Program ', msg, ' seru sekali')
33 selain_itu
34   writeln('Selesai');
35 selesai;
36
37 fungsi add(a, b: integer): integer;
38 mulai
39   add := a + b;
40 selesai;
41
42 mulai
43   x := 22;
44   y := 3;
45   z := 2018;
46   sum := x + y * (z bagi 10) - (x mod y);
47   avg := (x + y + z) / 3.0;
48
49   done := false;
50
51   jika (sum ≥ maxVal) dan tidak done maka
52     done := true
53   selain_itu jika (sum < maxVal) atau (x > y) maka
54     done := false;
55
56   jika (x ≤ y) maka
57     done := false;
58
59   selama (x > 0) lakukan
60     mulai
61       x := x - 1;
62     selesai;
63
64   arr[1] := 1.0;
65   arr[2] := 2.5555;
66   arr[3] := 3.14;
67   arr[4] := 4.0;
68   arr[5] := 2;
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
767
768
769
769
770
771
772
773
774
775
776
777
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
827
828
829
829
830
831
832
833
834
835
836
837
837
838
839
839
840
841
842
843
844
845
846
847
847
848
849
849
850
851
852
853
854
855
856
857
857
858
859
859
860
861
862
863
864
865
866
866
867
868
868
869
869
870
871
872
873
874
875
876
876
877
878
878
879
879
880
881
882
883
884
885
886
886
887
888
888
889
889
890
891
892
893
894
895
895
896
896
897
897
898
898
899
899
900
901
902
903
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656

```

## 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
30  test_full_tree program 0      0      1      0      0      0
31  maxVal   constant 1      0      1      0      100    31
32  piVal    constant 4      0      1      0      3.14   32
33  greet    constant 5      0      1      0      0      33
34  index    type     1      0      1      0      0      34
35  realArray type     6      0      1      0      0      35
36  letter   type     3      0      1      0      0      36
37  flag     type     2      0      1      0      0      37
38  koordinat type     0      0      1      0      0      38
39  x         variable 1      0      1      0      0      39
40  y         variable 1      0      1      0      0      40
41  z         variable 1      0      1      0      0      41
42  sum      variable 4      0      1      0      0      42
43  avg      variable 4      0      1      0      0      43
44  count    variable 1      0      1      0      0      44
45  c         variable 3      0      1      0      0      45
46  s         variable 5      0      1      0      0      46
47  arr      variable 6      1      1      0      0      47
48  done     variable 2      0      1      0      0      48
49  ok       procedure 0      0      1      0      0      49
50  msg      variable 3      0      1      1      0      0
51  add      function 1      0      1      0      0      50
52  a         variable 1      0      1      1      0      0
53  b         variable 1      0      0      1      1      0      53

otab:
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

```

07:48:29 [155/3109]

```

tab:
  ldx last lper psze vsize
  S2 0 0 10
  S1 0 1 1
  S4 0 2 2
  0 0 0 0

  ldx
  ldx xtyp etyp low high elsz size
  1 4 2 5 1 1 5

generated AST:
ProgramNode('test_full_tree')
|- ConstantNode('true')
|- ConstantNode('false')
|- ConstantNode('sival')
|- ConstantNode('greet')
|- VarNode('x') > tab_idx=40, type:real
|- TypeDeclNode('realarray')
|- TypeDeclNode('letterarray')
|- TypeDeclNode('chararray')
|- TypeDeclNode('koordinat')
|- VarDeclNode('x') > tab_idx=40, lev:0
|- VarDeclNode('y') > tab_idx=41, lev:0
|- VarDeclNode('z') > tab_idx=42, lev:0
|- VarDeclNode('avg') > tab_idx=43, lev:0
|- VarDeclNode('count') > tab_idx=44, lev:0
|- VarDeclNode('arr') > tab_idx=45, lev:0
|- VarDeclNode('c') > tab_idx=46, lev:0
|- VarDeclNode('e') > tab_idx=47, lev:0
|- VarDeclNode('arr') > tab_idx=48, lev:0
|- VarDeclNode('arr') > tab_idx=49, lev:0
ProcedureDeclNode('ok')
|- Block
  |- IfNode
    |- UnaryOpNode('!dok') > type:boolean
    |- VarNode('ok') > tab_idx=50, type:boolean
    |- ProcedureCallNode('writeln') > type:voidnone
    |- StringNode('Program') > type:string
    |- StringNode('writeln') > type:string
    |- StringNode('se sei') > type:string
    |- ProcedureCallNode('writeln') > type:voidnone
    |- StringNode('ok') > type:string
  |- FunctionDeclNode('add')
    |- Block
      |- AssignNode('sum') > tab_idx=51, type:integer
      |- VarNode('a') > tab_idx=52, type:integer
        |- VarNode('add') > tab_idx=53, type:integer
        |- VarNode('b') > tab_idx=54, type:integer
      |- AssignNode('Num') > tab_idx=55, type:integer
        |- VarNode('y') > tab_idx=56, type:integer
        |- VarNode('z') > tab_idx=57, type:integer
      |- AssignNode('Num') > tab_idx=58, type:integer
        |- VarNode('x') > tab_idx=59, type:integer
        |- NumberNode(2018) > tab_idx=60, type:integer
      |- AssignNode('BindOp+') > tab_idx=61, type:real
        |- BindNode('op+') > tab_idx=62, type:real
        |- BindNode('+', type:real)
        |- VarNode('x') > tab_idx=63, type:integer
        |- VarNode('y') > tab_idx=64, type:integer
        |- BindNode('+', type:real)
        |- VarNode('x') > tab_idx=65, type:integer
        |- VarNode('y') > tab_idx=66, type:integer
      |- AssignNode('BindOp*') > tab_idx=67, type:real
        |- BindNode('op*') > tab_idx=68, type:real
        |- BindNode('*', type:real)
        |- VarNode('x') > tab_idx=69, type:integer
        |- VarNode('y') > tab_idx=70, type:integer
        |- BindNode('op*') > tab_idx=71, type:real
        |- VarNode('x') > tab_idx=72, type:integer
        |- VarNode('y') > tab_idx=73, type:integer
      |- AssignNode('BindOp/') > tab_idx=74, type:real
        |- BindNode('op/') > tab_idx=75, type:real
        |- BindNode('/', type:real)
        |- VarNode('x') > tab_idx=76, type:integer
        |- VarNode('y') > tab_idx=77, type:integer
        |- NumberNode(3.0) > tab_idx=78, type:real
      |- AssignNode('Bool')(<false>)
        |- VarNode('done') > tab_idx=79, type:boolean
        |- BooleanNode(False) > type:boolean
      |- IfNode
        |- BindNode('dnn') > type:boolean
        |- BindNode('>') > type:boolean
        |- VarNode('sum') > tab_idx=80, type:real
        |- NumberNode(100) > type:integer
        |- UnaryOpNode('!dak') > type:boolean
        |- VarNode('done') > tab_idx=84, type:boolean
      |- AssignNode('Bool')(<true>)
        |- VarNode('done') > tab_idx=85, type:boolean
        |- BooleanNode(True) > type:boolean
      |- IfNode
        |- BindNode('<') > type:boolean
        |- BindNode('<=' type:real
        |- VarNode('sum') > tab_idx=86, type:real
        |- NumberNode(100) > type:integer
        |- BindNode('>') > type:boolean
        |- VarNode('x') > tab_idx=87, type:integer
        |- VarNode('y') > tab_idx=88, type:integer
        |- AssignNode('Bool')(<false>)
        |- VarNode('done') > tab_idx=89, type:boolean
        |- BooleanNode(False) > type:boolean
      |- IfNode
        |- BindNode('<=' type:boolnode
        |- VarNode('x') > tab_idx=90, type:integer
        |- VarNode('y') > tab_idx=91, type:integer
        |- AssignNode('Bool')(<false>)
        |- VarNode('done') > tab_idx=92, type:boolean
        |- BooleanNode(False) > type:boolean
      |- WhileNode
        |- BindNode('>') > type:boolean
        |- VarNode('x') > tab_idx=93, type:integer
        |- NumberNode(0) > type:integer
      |- Block
        |- AssignNode('BindOp-') > tab_idx=94, type:integer
        |- VarNode('x') > tab_idx=95, type:integer
        |- BindNode('-') > type:integer
        |- VarNode('x') > tab_idx=96, type:integer
        |- NumberNode(1) > type:integer
      |- AssignNode('ArrayAccess')(<1.0>)
        |- ArrayAccessNode('arr') > tab_idx=97, type:real
        |- NumberNode(1.0) > type:integer
        |- NumberNode(1.0) > type:real
      |- AssignNode('Num')(<2.5555>)
        |- ArrayAccessNode('arr') > tab_idx=98, type:real
        |- NumberNode(2) > type:integer
        |- NumberNode(2.5555) > type:real
      |- AssignNode('Num')(<3.14>)
        |- ArrayAccessNode('arr') > tab_idx=99, type:real
        |- NumberNode(3) > type:integer
        |- NumberNode(3.14) > type:real
      |- AssignNode('Num')(<4.0>)
        |- ArrayAccessNode('arr') > tab_idx=100, type:real

```

```

    └ 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('serv sekali')
    └ VarNode('s') → tab_index:47, type:string
    └ StringNode serv 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

```

## 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)

```
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.
```

### 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 direspon 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 - <i>Testing</i>	25%
13523116	Fityatul Haq Rosyidi	- Menyesuaikan Parser - <i>Testing</i>	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](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%2012.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](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.ulg.ac.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\\_\(linguistics\)](https://en.wikipedia.org/wiki/Semantic_analysis_(linguistics))