

IF2224 TEORI BAHASA FORMAL DAN OTOMATA

IMPLEMENTASI SEMANTIC ANALYSIS PADA PASCAL-S

COMPILER BERBASIS ABSTRACT SYNTAX TREE

Laporan Milestone 3

Disusun untuk memenuhi tugas mata kuliah Teori Bahasa Formal dan Otomata pada
Semester 5 (lima)

Tahun Akademik 2025/2026



Disusun Oleh:

Brian Ricardo Tamin 13523126

Jovandra Otniel P. S. 13523141

Andrew Tedjapratama 13523148

Theo Kurniady 13523154

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2025

DAFTAR ISI

| | |
|---|----|
| DAFTAR ISI..... | 2 |
| BAB I | |
| DESKRIPSI PERSOALAN..... | 4 |
| BAB II | |
| LANDASAN TEORI..... | 7 |
| 2.1 Semantic Analysis and Attributed Grammar..... | 7 |
| 2.1.1 Semantic Analysis..... | 7 |
| 2.1.2 Attributed Grammar..... | 8 |
| 2.2 Abstract Syntax Tree..... | 9 |
| 2.3 Symbol Table..... | 9 |
| 2.3.1 Identifier Table (tab)..... | 10 |
| 2.3.2 Block Table (btab)..... | 12 |
| 2.3.3 Array Table (atab)..... | 12 |
| 2.4 Type Check..... | 13 |
| 2.4.1 Representasi Tipe..... | 13 |
| 2.4.2 Assignment Checking..... | 14 |
| 2.4.3 Pemeriksaan Operasi Aritmatika, Relasional, dan Logika..... | 14 |
| 2.5 Error Handling..... | 15 |
| BAB III | |
| PERANCANGAN & ALUR PROGRAM..... | 16 |
| 3.1 Arsitektur dan Teknologi yang Digunakan..... | 16 |
| 3.2 Struktur Program dan Kelas Utama..... | 16 |
| 3.3 Alur Kerja Program..... | 17 |
| 3.4 Justifikasi Pilihan Implementasi..... | 18 |
| BAB IV | |
| IMPLEMENTASI PROGRAM..... | 19 |
| 4.1 AST Node..... | 19 |
| 4.2 Nodes..... | 20 |
| 4.3 AST Builder..... | 27 |
| 4.4 Symbol Table..... | 35 |
| 4.5 Constants..... | 38 |
| 4.6 TypeChecker..... | 39 |
| 4.7 Errors..... | 40 |
| 4.8 Semantic Analyzer..... | 42 |
| BAB V | |
| PENGUJIAN & TEST CASES..... | 43 |
| 5.1 Test Case General..... | 43 |
| 5.2 Test Case untuk Milestone 3..... | 47 |
| BAB VI | |
| KESIMPULAN & SARAN..... | 51 |
| 6.1 Kesimpulan..... | 51 |
| 6.2 Saran..... | 51 |
| REFERENSI..... | 52 |

| | |
|---------------|----|
| LAMPIRAN..... | 53 |
|---------------|----|

BAB I

DESKRIPSI PERSOALAN

1.1 Pendahuluan

Bahasa pemrograman Pascal-S merupakan sebuah subset dari bahasa Pascal yang dirancang dengan aturan sintaks yang lebih terbatas. Pada tahap ketiga pembangunan compiler Pascal-S, setelah mengembangkan tahap Lexical Analysis dan Syntax Analysis, compiler yang dibangun sudah bisa berhasil memecah kode sumber menjadi token dan menyusunnya menjadi struktur pohon parsing (Parse Tree) berdasarkan aturan tata bahasa (*grammar*) yang ditentukan. Parser dalam milestone 2 memastikan bahwa kode program ditulis dengan struktur kalimat yang benar.

Namun, kebenaran secara sintaks atau tata bahasa tidak menjamin kebenaran secara makna (*semantic*). Sebuah program bisa saja lolos valid dari parser meskipun melakukan operasi yang tidak masuk akal, contohnya seperti menjumlahkan tipe data boolean dengan integer, atau menggunakan variabel yang belum dideklarasikan. Kesalahan-kesalahan ini berkaitan dengan konsistensi tipe dan lingkup (*scope & type checking*) variabel yang tidak dapat dideteksi hanya dengan *Context-Free Grammar* (CFG).

Oleh karena itu, dalam Milestone 3, fokus akan berpindah pada implementasi Semantic Analysis. Tahap ini bertujuan untuk memvalidasi makna dari struktur program yang telah dibentuk. Proses ini melibatkan penggunaan *Attributed Grammar* dan pengelolaan Symbol Table untuk melacak deklarasi identifier, tipe data, dan scope, serta adanya *Visitor Functions* untuk menelusuri *Abstract Syntax Tree* (AST). Pada akhir, proses ini akan menghasilkan suatu hasil akhir berupa *Decorated Abstract Syntax Tree* (AST) yang telah dianotasi dengan informasi tipe ekspresi, referensi ke symbol table, dan informasi scope, serta jaminan bahwa program bebas dari kesalahan semantik sebelum masuk ke tahap berikutnya, yaitu tahap *Intermediate Code Generation*.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, permasalahan utama yang harus diselesaikan dalam milestone ini adalah:

1. Pengelolaan Symbol Table: Bagaimana merancang struktur data Symbol Table yang efisien (menggunakan tabel tab, btab, dan atab) untuk menyimpan informasi identifier, menangani *nested scope* (misalnya prosedur dalam prosedur), dan membedakan antara *pass-by-value* dengan *pass-by-reference*?
2. Validasi Tipe (*Type Checking*): Bagaimana mekanisme pengecekan kompatibilitas tipe data pada operasi assignment, ekspresi aritmatika, dan pemanggilan fungsi agar sesuai dengan aturan ketat Pascal-S?
3. Konstruksi Decorated AST: Bagaimana mengubah Parse Tree mentah menjadi Abstract Syntax Tree (AST) yang lebih bersih dengan informasi yang lengkap, lalu melakukan traversals (menggunakan pola Visitor) untuk menyematkan atribut semantik (seperti tipe data dan referensi tabel) pada setiap node berbeda?
4. Penanganan Error Semantik: Bagaimana mendeteksi dan melaporkan kesalahan semantik seperti Undeclared Identifier, Type Mismatch, atau Redeclared Identifier secara akurat tanpa menghentikan atau melakukan *crash* kasar pada compiler?

1.3 Tujuan

Milestone ini bertujuan untuk:

1. Mengimplementasikan Semantic Analyzer yang mampu menelusuri AST untuk memverifikasi kebenaran logika program.
2. Membangun dan mengelola Symbol Table yang terdiri dari tabel identifier (tab), tabel blok (btab), dan tabel array (atab) sesuai spesifikasi Pascal-S untuk manajemen memori dan scope.
3. Menghasilkan Decorated AST, yaitu pohon sintaks abstrak yang setiap node-nya telah dilengkapi dengan informasi tipe data (type) dan referensi tabel simbol (tab_index).
4. Memastikan aturan strong typing pada Pascal-S terpenuhi, termasuk validasi operasi matematika, logika, dan akses array/record.

5. Menangani deklarasi tipe kompleks seperti Array dan Record beserta akses elemennya (seperti arr[i] atau rec.field).

1.4 Batasan Masalah

Lingkup penggerjaan pada tahap ini dibatasi pada:

1. Analisis semantik dilakukan secara statis (compile-time), tidak menangani runtime error.
2. Implementasi mengacu pada spesifikasi bahasa Pascal-S, termasuk aturan *predefined procedures* (writeln, readln), tipe data dasar (integer, real, boolean, char), dan daftar *reserved words*.

BAB II

LANDASAN TEORI

2.1 Semantic Analysis and Attributed Grammar

2.1.1 Semantic Analysis

Semantic analysis adalah salah satu tahapan krusial dari compiler yang memeriksa struktur semantik (makna) dari kode pemrograman yang sudah diubah menjadi *syntax tree* oleh parser. Sebelumnya, *Syntax Analysis* (Parser) hanya peduli pada “apakah struktur kalimatnya benar secara tata bahasa?”, namun pada tahap ini, Semantic Analysis berfokus pada “apakah kalimat tersebut masuk akal?”. Hal ini diperlukan karena parser berbasis Context-Free Grammar (CFG) memiliki keterbatasan dalam mendeteksi kesalahan yang bergantung pada konteks (context-sensitive), seperti penggunaan variabel yang belum dideklarasikan.

Dalam implementasinya, Semantic Analyzer terdiri dari tiga komponen fundamental: **Abstract Syntax Tree (AST)** sebagai representasi struktur program, **Symbol Table** sebagai memori persisten untuk melacak entitas, dan **Type Checking** untuk memvalidasi kompatibilitas data. Ketiganya bekerja secara sinergis melalui mekanisme tree traversal (sering menggunakan pola Visitor) untuk menelusuri setiap node dan melakukan validasi.

Selain validasi, tahap ini juga mencakup mekanisme penanganan kesalahan (error handling) untuk isu-isu semantik, seperti Type Mismatch (ketidaksesuaian tipe data pada assignment atau operasi), Argument Mismatch (jumlah/tipe parameter fungsi tidak sesuai), serta Undeclared Identifier. Validasi ini sangat penting terutama bagi Pascal-S yang menganut sistem **Strong Static Typing**, di mana setiap variabel wajib dideklarasikan secara eksplisit dengan tipe data tertentu sebelum digunakan.

Keluaran atau output akhir dari fase ini adalah Decorated AST, yaitu pohon sintaks yang telah dianotasi dengan atribut semantik (seperti tipe data hasil evaluasi) dan referensi ke Symbol Table. Symbol Table itu sendiri adalah struktur data yang menyimpan informasi komprehensif mengenai identifier (nama, tipe, scope) dan diorganisasikan menggunakan struktur stack terpisah (tab, btab, atab) untuk menangani nested scope secara efisien.

2.1.2 Attributed Grammar

Dalam implementasi milestone ini, digunakan algoritma L-Attributed Grammar, yang secara definisi umum merupakan grammar yang dilengkapi dengan atribut dan aturan semantik. Attributed Grammar adalah perluasan dari Context-Free Grammar (CFG) yang memungkinkan setiap simbol (terminal maupun non-terminal) memiliki atribut. Setiap aturan produksi dalam attributed grammar dilengkapi dengan aturan semantik yang menentukan bagaimana atribut dihitung.

Dalam teori klasik, atribut dibagi menjadi dua jenis:

- **Synthesized Attribute**

Mengalir dari *child* ke *parent*, digunakan untuk menghitung nilai ekspresi atau menentukan tipe hasil operasi.

- **Inherited Attribute**

Mengalir dari *parent* atau *sibling* ke *child*. Berguna untuk meneruskan informasi konteks, seperti scope atau tipe deklarasi.

Penggunaan attributed grammar sangat penting dalam *semantic analysis* karena grammar CFG saja tidak cukup untuk memvalidasi konteks dan makna program. Misalnya, pada aturan produksi untuk assignment:

$$< \text{assignment} > \rightarrow < \text{variable} > := < \text{expression} >$$

Semantic rule yang terkait dapat berupa:

- $\text{variable.type} = \text{lookup}(\text{variable.name})$
- $\text{expression.type} = \text{evaluate}(\text{expression})$
- $\text{check}(\text{variable.type} == \text{expression.type})$

Melalui skema ini, compiler dapat memastikan bahwa operasi yang dilakukan konsisten secara semantik. Dalam compiler Pascal-S, Attributed Grammar menjadi dasar Syntax Directed Translation (SDT) yang berperan besar pada proses pembentukan AST dan type checking, sebagaimana dijelaskan pada bagian lain dalam laporan. Dengan demikian, attributed grammar berfungsi sebagai jembatan antara struktur sintaks (hasil parsing) dan struktur makna (data semantik) yang diperlukan pada tahap selanjutnya.

2.2 Abstract Syntax Tree

Abstract Syntax Tree adalah representasi hierarkis terstruktur dari kode program yang lebih ringkas daripada *syntax tree* pada umumnya. AST merepresentasikan suatu tree yang telah diabstraksi dari Parse Tree yang dibentuk pada milestone 2, berisi detail derivasi tata bahasa (termasuk elemen sintaks dekoratif seperti titik koma, tanda kurung, dan keyword begin/end), maka AST hanya menyisakan atau mempertahankan elemen-elemen yang dianggap penting dan esensial secara semantik.

Konstruksi AST dalam tugas ini didasarkan pada prinsip *Syntax-Directed Translation Scheme*, mengidentifikasi setiap aturan produksi (production rule) dan aksi semantik (semantic action) yang terkait. Saat aksi semantik tersebut dijalankan pada node parse tree, sistem akan secara dinamis membentuk objek node baru yang lebih ringkas dan terstruktur, yang kemudian disusun menjadi Abstract Syntax Tree (AST). Pendekatan ini memastikan bahwa transformasi dari struktur sintaksis mentah menjadi struktur logis berjalan secara deterministik dan akurat.

Setelah AST sudah terbentuk, proses Semantic Analysis kemudian dilakukan dengan menelusuri setiap node menggunakan *semantic visitor*, yaitu serangkaian fungsi visit yang pada spesifikasi milestone ini, dilakukan secara top-down mengikuti struktur tree. Pendekatan ini memungkinkan informasi konteks seperti scope, level blok, dan tipe deklarasi diwariskan dari node induk ke node anak. Selama traversal, visitor mencatat deklarasi baru ke dalam symbol table, memvalidasi penggunaan identifier, mengecek konsistensi tipe pada ekspresi serta assignment, dan memberi anotasi semantik (seperti tipe hasil evaluasi atau referensi tabel simbol) langsung ke node AST.

2.3 Symbol Table

Symbol table adalah struktur data sentral dalam proses kompilasi di tahap *Semantic Analysis*, yang berfungsi seolah sebagai “database” untuk menyimpan informasi mengenai semua identifier yang dideklarasikan dalam program (variabel, konstanta, tipe, prosedur, fungsi). Perlu dipahami bahwa fase sebelumnya, yaitu *Syntax Analysis* berbasis CFG memiliki keterbatasan fundamental, ia bersifat *stateless* atau tidak memiliki memori tentang apa yang telah dideklarasikan sebelumnya. Parser hanya mampu memvalidasi urutan token (subjek-predikat-objek), tetapi buta terhadap konteks. Oleh karena itu, Symbol Table hadir sebagai struktur data persisten yang bertugas merekam, memelihara, dan melacak status serta

informasi dari setiap entitas sepanjang proses kompilasi, memastikan bahwa setiap penggunaan variabel konsisten secara makna, dengan deklarasi awalnya.

Sesuai dengan spesifikasi Pascal-S compiler, implementasi symbol table dibagi menjadi tiga jenis symbol table:

2.3.1 Identifier Table (tab)

Tabel tab menyimpan entri untuk setiap identifier yang dideklarasikan. Struktur data ini diorganisasikan sebagai stack global, namun secara logis dikelola menggunakan Linked List per scope untuk resolusi identifier yang efisien. Setiap entri pada tab memiliki atribut yang dapat dilihat pada tabel berikut:

Tabel 2.3.1 List atribut tabel tab dan deskripsinya

| 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. |
|-----|--|

2.3.2 Block Table (btab)

Tabel btab digunakan untuk menyimpan informasi mengenai struktur blok kode, yaitu program utama, prosedur, fungsi, dan definisi tipe record. Setiap entri pada btab memiliki atribut yang dapat dilihat pada tabel berikut:

Tabel 2.3.2 List atribut tabel btab dan deskripsinya

| 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) |

2.3.3 Array Table (atab)

Karena array memiliki struktur yang kompleks (tipe elemen, tipe indeks, dan rentang), informasinya dipisahkan ke dalam tabel atab. Setiap entri array atab didefinisikan pada tabel berikut:

Tabel 2.3.3 List atribut tabel atab dan deskripsinya

| 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 |

2.4 Type Check

Type checking adalah proses yang penting dalam menentukan kesesuaian tipe pada statement, deklarasi, atau ekspresi pada kode program. Type check merupakan tahap selanjutnya dari Symbol Table. Hal ini karena proses tersebut mengambil informasi dari symbol table untuk mengecek kesesuaian tipe data yang dipakai. Output dari Type Check adalah decorated abstract syntax tree dimana setiap simpul (node) dilabeli tipe data yang sesuai. Jika terdapat semantic error, type checker akan mengeluarkan Error (*raise error*) yang merupakan bagian dari *compile error*.

2.4.1 Representasi Tipe

Berdasarkan implementasi type_check pada aplikasi semantic analyzer, tipe data dibagi menjadi 7 macam, yaitu integer, real, boolean, char, array, record, dan notype. Integer merupakan bilangan bulat. Real merupakan bilangan floating-point (bilangan desimal). Boolean merupakan tipe data yang berisi nilai kebenaran (true/false). Char merupakan tipe data yang menyimpan satu karakter yang direpresentasikan oleh ASCII. Array merupakan tipe data yang menyimpan beberapa nilai dengan tipe sama pada suatu blok memori kontigu. Record merupakan tipe data yang menyimpan

beberapa nilai dengan tipe yang berbeda pada suatu blok memori kontigu. NoType merupakan label tipe data yang tidak dikenali di Pascal-S.

2.4.2 Assignment Checking

Pada bahasa pemrograman Pascal-S, assignment wajib mengikuti urutan ekspresi seperti berikut:

$$< \text{target} > := < \text{expression} >$$

Aturan kompatibilitas tipe data terdiri dari:

- Tipe target harus sama dengan nilai
- Integer boleh diassign ke real
- Target harus berupa i-value (identifier yang bisa diubah)
- Tidak boleh melakukan assignment ke const.

2.4.3 Pemeriksaan Operasi Aritmatika, Relasional, dan Logika

Pada Pascal-S, terdapat tiga operasi yang sangat penting, yaitu operasi aritmatika, relasional, dan logika. Operasi aritmatika melibatkan penjumlahan, pengurangan, perkalian, pembagian (real dan integer), dan modulo. Operasi relasional merupakan operasi yang membandingkan nilai dari kedua operand (lebih dari, kurang dari, sama dengan). Sementara itu, operasi logika menggunakan operasi logika dasar (and, or, not) untuk mencari nilai true/false dari operand yang terlibat.

Pada Pascal-S, penjumlahan, pengurangan, dan perkalian menghasilkan integer jika kedua operand integer atau real jika paling sedikit salah satu operand berupa real. Pembagian integer (div) menghasilkan nilai integer yang dibulatkan ke bawah, sedangkan pembagian real (/) menghasilkan nilai real. Modulo selalu menghasilkan integer.

Pada operasi relasional, hasil selalu berupa boolean. Operasi kesetaraan (= dan \neq) harus melibatkan 2 tipe data yang sama. Sementara itu, operasi perbandingan (seperti $>$, $<$, \geq , \leq) harus melibatkan 2 tipe data ordinal seperti integer dan char.

2.4.4 Pemeriksaan Index Array, Ekspresi Kondisi, dan Rentang Perulangan For

Pengaksesan array arr[index] diperiksa oleh semantic analyzer, dimana arr harus berupa variabel dengan tipe data array, dan index merupakan nilai integer. Pada blok if-then, bagian yang terdapat dalam blok kondisional (if) harus menghasilkan nilai boolean, baik itu dari operasi relasional maupun logika. Pada blok perulangan seperti *for i := angka_start to angka_end do ...*, variabel loop (i) harus berupa ordinal (integer, boolean, char). Selain itu, tipe data *angka_start* dan *angka_end* harus sama.

2.5 Error Handling

Jika pada analisis semantik ditemukan kesalahan, compiler Pascal-S akan melempar error sesuai kesalahan pada kode program tersebut. Kesalahan tersebut mencakup variabel yang tidak dideklarasikan, ketidakcocokan tipe data, dan pemanggilan fungsi/prosedur yang argumennya tidak sesuai. Kesalahan tersebut menjadi rincian kesalahan yang terdapat pada kode yang sedang dianalisis secara semantik.

BAB III

PERANCANGAN & ALUR PROGRAM

Penjelasan ini merinci rancangan program *semantic analyzer* untuk *compiler* Pascal-S, mencakup teknologi, struktur, alur kerja, dan justifikasi atas pilihan desain yang diambil.

3.1 Arsitektur dan Teknologi yang Digunakan

Program ini dibangun murni menggunakan bahasa pemrograman Python 3. Tidak ada *library* eksternal yang digunakan untuk logika inti. Pemilihan Python didasarkan pada keterbacaan kode, kemudahan pengelolaan struktur data seperti stack dan dictionary yang digunakan pada symbol table, serta dukungan Python terhadap pendekatan rekursif yang sesuai dengan pola kerja visitor functions dan tree traversal, menghasilkan implementasi yang sederhana namun tetap terstruktur.

3.2 Struktur Program dan Kelas Utama

```
BBC-TUBES-IF2224
├── main.py
├── README.md
└── doc/
src/
├── compiler.py
├── utils.py
└── config/
    ├── states.json
    ├── token_maps.json
    └── transitions.json
└── dfa/
└── lexer/
└── parser/
semantic/
    ├── semantic_analyzer.py
    ├── string_handler.py
    ├── errors.py
    └── AST/
        ├── ast_builder.py
        ├── ast_node.py
        └── nodes.py
    └── symbol/
        ├── symbol_table.py
        └── constants.py
    └── type_checker/
        └── type_checker.py
test/
    ├── milestone-1/
    │   ├── input/
    │   ├── input-indo/
    │   └── output/
    └── milestone-2/
        ├── input/
        └── output/
    └── milestone-3/
        ├── input/
        └── output/
```

3.3 Alur Kerja Program

Alur kerja program dimulai dari pembacaan parse tree hingga pembuatan AST, tabel simbol, dan pemeriksaan semantik. Proses ini dapat dijelaskan secara bertahap untuk memberikan pemahaman menyeluruh tentang mekanisme analisis semantik yang diterapkan.

1. Pembentukan AST (*Abstract Syntax Tree*)

Program memulai analisis dengan menerima parse tree yang dihasilkan oleh parser pada milestone-2. Parse tree ini merepresentasikan struktur sintaksis program secara hierarkis berdasarkan grammar Pascal.

2. Inisialisasi tabel simbol dan built-in

Setelah AST terbentuk, program menginisialisasi Symbol Table yang menyimpan semua identifier, tipe, variabel, prosedur/fungsi, serta array. Reserved words dan prosedur/fungsi bawaan (built-in) dimasukkan pada level global, sehingga dapat dikenali sepanjang analisis. Setiap blok baru atau scope lokal akan otomatis dicatat untuk mendukung scoping dan lookup identifier.

3. Traversal AST untuk mengisi tabel simbol

Program melakukan traversal AST secara rekursif menggunakan berbagai metode visit. Setiap node dikunjungi, dan deklarasi variabel, konstanta, tipe, prosedur, dan fungsi dicatat ke tabel simbol lengkap dengan level leksikal, alamat, dan referensi blok. Record dan array ditangani secara khusus agar akses field atau elemen array sesuai dengan tipe yang benar.

4. Pemeriksaan tipe dan konsistensi

Selama traversal, tipe ekspresi, assignment, operasi biner/unary, serta pemanggilan fungsi dan prosedur diperiksa. Program memastikan kompatibilitas tipe operand, tipe return fungsi, serta jumlah dan tipe argumen pada prosedur/fungsi. Pemeriksaan ini mencegah kesalahan semantik, seperti penggunaan variabel yang belum dideklarasikan atau tipe yang tidak sesuai.

5. Dekorasi AST

Semantic analyzer kemudian melakukan traversal parse tree, mengunjungi setiap node untuk mengekstrak informasi deklarasi, tipe data, ekspresi, dan pernyataan. Setiap node dihias (decorated) dengan atribut tambahan seperti tipe dan indeks tabel simbol, sehingga memudahkan proses pemeriksaan semantik pada tahap berikutnya.

6. Penanganan error semantik

Bila ada pelanggaran semantik, akan ada error handling dari kelas-kelas error pada file Errors.py. Setiap pelanggaran semantik dicatat ke daftar errors tanpa menghentikan analisis. Hal ini memungkinkan program untuk mendeteksi dan melaporkan semua kesalahan sekaligus.

3.4 Justifikasi Pilihan Implementasi

- 1. Penggunaan AST (*Abstract Syntax Tree*)**

Pemilihan AST sebagai dasar analisis semantik karena AST sudah menyederhanakan struktur kode dibandingkan parse tree lengkap. Dengan AST, node-node yang tidak relevan (seperti tanda kurung atau token terminal yang sederhana) dihilangkan, sehingga traversal menjadi lebih efisien dan fokus pada elemen semantik penting. Hal ini memudahkan anotasi tipe, referensi simbol, dan deteksi kesalahan semantik secara lebih terstruktur.

- 2. Penggunaan Symbol Table untuk Scope dan Tipe**

Implementasi tabel simbol mendukung pencatatan entri untuk variabel, konstanta, fungsi, prosedur, dan tipe kustom seperti record maupun array. Selain itu, struktur ini memfasilitasi manajemen nested scope, sehingga deklarasi lokal dan global dapat dibedakan dengan jelas. Pendekatan ini juga memungkinkan akses cepat ke informasi tipe saat melakukan pemeriksaan semantik.

- 3. Pemisahan TypeChecker dari SemanticAnalyzer**

Implementasi checker tipe terpisah dari traversal semantic analyzer meningkatkan modularitas dan keterbacaan kode. TypeChecker bertanggung jawab pada logika kompatibilitas tipe, sementara SemanticAnalyzer fokus pada traversal AST, anotasi node, dan pencatatan error. Pemisahan ini memudahkan debugging, pengujian, dan kemungkinan reuse untuk analisis semantik bahasa lain di masa depan.

- 4. Gabungan Analisis Semantik dan Error Handling**

Dengan mengintegrasikan analisis tipe dan penanganan kesalahan dalam satu proses traversal, semantic analyzer dapat memberikan umpan balik kesalahan semantik secara real-time. Pendekatan ini mengurangi kebutuhan pass tambahan dan membantu mendeteksi error lebih awal, meningkatkan keandalan pemeriksaan semantik.

BAB IV

IMPLEMENTASI PROGRAM

4.1 AST Node

Kelas ini menjadi struktur utama dari AST dalam membentuk pohon.

| Class | Penjelasan Singkat |
|--|---|
| <pre>•••••\n\nfrom ..symbol.constants import TypeKind\nclass ASTNode:\n def __init__(self):\n self.children = []\n self.attr = {} \n\n def add_child(self, child):\n self.children.append(child)\n\n def __str__(self):\n return self._format_tree("", True)\n\n def _format_tree(self, prefix, is_last):\n connector = "└ " if is_last else "├ "\n annotation = self._get_annotation_str()\n\n # Bersihkan repr dari newline\n node_repr = repr(self).strip()\n line = f'{prefix}{connector}{node_repr}{annotation}\n'\n\n child_prefix = prefix + (" " if is_last else " ") \n valid_children = [c for c in self.children if c is not None]\n count = len(valid_children)\n\n for i, child in enumerate(valid_children):\n if hasattr(child, '_format_tree'):\n line += child._format_tree(child_prefix, i == count - 1)\n else:\n line += f'{child_prefix}{connector}{str(child)}\n'\n return line\n\n def _get_annotation_str(self):\n parts = [\n type_map = {\n TypeKind.NOTYPE: "void", TypeKind.INTEGER: "integer",\n TypeKind.REAL: "real", TypeKind.BOOLEAN: "boolean",\n TypeKind.CHAR: "char", "predefined": "predefined"\n }\n\n if self.attr.get('type') == 'predefined': parts.append("predefined")\n\n if 'tab_index' in self.attr and self.attr['tab_index'] is not None:\n parts.append(f"tab_index:{self.attr['tab_index']}")\n\n if 'block_index' in self.attr:\n parts.append(f"block_index:{self.attr['block_index']}")\n\n if 'type' in self.attr and self.attr['type'] != 'predefined':\n val = self.attr['type']\n t_str = type_map.get(val, str(val)) if isinstance(val, int) else str(val)\n parts.append(f"type:{t_str}")\n\n if 'lev' in self.attr: parts.append(f"lev:{self.attr['lev']}")\n\n return " " + ", ".join(parts) if parts else ""</pre> | <p>Class ASTNode</p> <p>Kelas ASTNode adalah representasi dasar dari sebuah node dalam Abstract Syntax Tree (AST), struktur data yang digunakan untuk merepresentasikan hierarki sintaks program. Setiap node menyimpan daftar children, yaitu node anak yang merepresentasikan substruktur dari program, serta dictionary attr yang menyimpan atribut tambahan seperti tipe data, indeks di symbol table, level blok, atau informasi lain yang relevan untuk analisis semantik. Node ini bersifat fleksibel, sehingga dapat menampung node lain sebagai anaknya atau data sederhana seperti string atau angka.</p> <p>Kelas ini juga menyediakan mekanisme untuk mencetak AST secara visual menggunakan metode <code>__str__</code> dan <code>_format_tree</code>, menghasilkan tampilan tree ASCII yang rapi dengan indentasi dan connector untuk menunjukkan hubungan hierarki antar node. Setiap node dapat menampilkan anotasi tambahan melalui <code>_get_annotation_str</code>, yang mengekstrak informasi dari atribut seperti tipe data (type), indeks tabel simbol (tab_index), level blok (block_index), atau level lexical (lev). Dengan cara ini, ASTNode tidak hanya menyimpan struktur program, tetapi juga metadata penting yang memudahkan debugging dan analisis semantik.</p> |

4.2 Nodes

Kumpulan kelas yang merepresentasikan setiap node dalam AST.

| Class | Penjelasan Singkat |
|---|---|
|  <pre>class ProgramNode(ASTNode): def __init__(self, name, declarations=None, block=None): super().__init__() self.name = name self.declarations = declarations or [] self.block = block if self.declarations: self.children.extend(self.declarations) if self.block: self.children.append(self.block) def __repr__(self):</pre> | Kelas ProgramNode Mewakili program Pascal, berisi nama program, deklarasi, dan block utama. |
|  <pre>class BlockNode(ASTNode): def __init__(self, statements=None): super().__init__() self.statements = statements or [] self.children.extend(self.statements) def __repr__(self): return f"Block"</pre> | Kelas BlockNode Mewakili sekumpulan statement yang dieksekusi sebagai satu blok. |
|  <pre>class DeclarationsNode(ASTNode): def __init__(self, declarations=None): super().__init__() self.declarations = declarations or [] self.children.extend(self.declarations) def __repr__(self): return "Declarations"</pre> | Kelas DeclarationsNode Menyimpan daftar deklarasi (const, var, type). |

```
class ConstDeclNode(ASTNode):
    def __init__(self, name, consttype):
        super().__init__()
        self.name = name
        self.consttype = consttype

    def __repr__(self):
        return f"ConstDecl(name='{self.name}')"
```

Kelas ConstDeclNode

Deklarasi konstanta dengan nama dan nilai bertipe tertentu.

```
class VarDeclNode(ASTNode):
    def __init__(self, name, vartype):
        super().__init__()
        self.name = name
        self.vartype = vartype
        if isinstance(vartype, ASTNode):
            self.children.append(vartype)

    def __repr__(self):
        return f"VarDecl('{self.name}')"
```

Kelas VarDeclNode

Deklarasi variabel dengan nama dan tipe variabel.

```
class TypeDeclarationNode(ASTNode):
    def __init__(self, name, type_node):
        super().__init__()
        self.name = name
        self.type_node = type_node
        self.children.append(type_node)

    def __repr__(self):
        return f"TypeDecl(name='{self.name}')"
```

Kelas TypeDeclarationsNode

Deklarasi tipe baru dengan nama tipe dan node tipe terkait.

```
class AssignNode(ASTNode):
    def __init__(self, target, value):
        super().__init__()
        self.target = target
        self.value = value
        self.children.extend([target, value])

    def __repr__(self):
        t_str = getattr(self.target, 'name', 'target')

        if hasattr(self.value, 'value'):
            v_str = self.value.value
        elif hasattr(self.value, 'name'):
            v_str = self.value.name
        else:
            v_str = "expr"

        return f"Assign('{t_str}' := {v_str})"
```

Kelas AssignNode

Mewakili operasi assignment target dengan value.



```
class VarNode(ASTNode):
    def __init__(self, name):
        super().__init__()
        self.name = name

    def __repr__(self):
        return f"target '{self.name}'"
```

Kelas VarNode

Mewakili referensi variabel berdasarkan nama.



```
class NumNode(ASTNode):
    def __init__(self, value):
        super().__init__()
        self.value = value

    def __repr__(self):
        return f"value {self.value}"
```

Kelas NumNode

Literal angka.



```
class StringNode(ASTNode):
    def __init__(self, value):
        super().__init__()
        self.value = value

    def __repr__(self):
        return f"String ({self.value})"
```

Kelas StringNode

Literal string.



```
class BooleanNode:
    def __init__(self, value):
        super().__init__()
        self.value = value

    def __repr__(self):
        return f"Boolean ({self.value})"
```

Kelas BooleanNode

Literal boolean.

```

● ● ●

class UnaryOpNode(ASTNode):
    def __init__(self, op, operand):
        super().__init__()
        self.op = op
        self.operand = operand
        self.children.append(operand)

    def __repr__(self):
        return f"UnaryOp(op='{self.op}')"

```

Kelas UnaryOpNode

Operasi unary seperti negasi dan kebalikan.

```

● ● ●

class BinOpNode(ASTNode):
    def __init__(self, left, op, right):
        super().__init__()
        self.left = left
        self.op = op
        self.right = right
        self.children.extend([left, right])

    def __repr__(self):
        return f"BinOp '{self.op}'"

```

Operasi biner seperti aritmetika atau perbandingan.

```

● ● ●

class ProcedureDeclNode(ASTNode):
    def __init__(self, name, params, block):
        super().__init__()
        self.name = name
        self.params = params
        self.block = block

        self.children.extend(self.params)
        self.children.append(self.block)

    def __repr__(self):
        return f"ProcedureDecl(name='{self.name}')"

```

Deklarasi prosedur dengan nama, parameter, dan isi block.

```

● ● ●

class FunctionDeclNode(ASTNode):
    def __init__(self, name, params, return_type, block):
        super().__init__()
        self.name = name
        self.params = params
        self.return_type = return_type
        self.block = block

        self.children.extend(self.params)
        self.children.append(self.block)

    def __repr__(self):
        return f"FunctionDecl(name='{self.name}')"

```

Deklarasi fungsi dengan parameter, tipe kembali, dan block.

```

● ● ●

class ParamNode(ASTNode):
    def __init__(self, names, type_node, is_var=False):
        super().__init__()
        self.names = names
        self.type_node = type_node
        self.is_var = is_var

    def __repr__(self):
        return f"Param(names='{self.names}', type='{self.type_node}', is_var={self.is_var})"

```

Parameter pada fungsi/prosedur, bisa bernilai **var**-parameter.

```

● ● ●

class ProcedureFunctionCallNode(ASTNode):
    def __init__(self, name, args=None):
        super().__init__()
        self.name = name
        self.args = args or []
        self.children.extend(self.args)

    def __repr__(self):
        return f"ProcedureFunctionCall('{self.name}')"

```

Pemanggilan fungsi atau prosedur dengan argumen.

```

● ● ●

class IfNode(ASTNode):
    def __init__(self, condition, then_block, else_block=None):
        super().__init__()
        self.condition = condition
        self.then_block = then_block
        self.else_block = else_block
        self.children.extend([condition, then_block])
        if else_block:
            self.children.append(else_block)

    def __repr__(self):
        return "If Condition"

```

Struktur if condition berisi kondisi, then-block, dan optional else-block.

```

● ● ●

class WhileNode(ASTNode):
    def __init__(self, condition, body):
        super().__init__()
        self.condition = condition
        self.body = body
        self.children.extend([condition, body])

    def __repr__(self):
        return f"While Loop"

```

Struktur while loop berisi kondisi dan body loop.

```

● ● ●

class RepeatNode(ASTNode):
    def __init__(self, body, condition):
        super().__init__()
        self.body = body
        self.condition = condition
        self.children.extend([body, condition])

    def __repr__(self):
        return f"Repeat Until"

```

Struktur repeat until loop berisi body dan kondisi penghentian.

```

● ● ●

class ForNode(ASTNode):
    def __init__(self, var_node, start_expr, end_expr, direction, body):
        super().__init__()
        self.var_node = var_node
        self.start_expr = start_expr
        self.end_expr = end_expr
        self.direction = direction
        self.body = body
        self.children.extend([var_node, start_expr, end_expr, body])

    def __repr__(self):
        varname = getattr(self.var_node, 'name', 'var')
        return f"For Loop"

```

Kelas ForNode

Struktur for loop dengan variabel iterasi, batas awal/akhir, arah, dan body.

```

● ● ●

class CaseBranchNode(ASTNode):
    def __init__(self, constants, statement):
        super().__init__()
        self.constants = constants
        self.statement = statement
        self.children.extend(constants + [statement])

    def __repr__(self):
        consts = ", ".join(str(c.value) if hasattr(c, 'value') else str(c) for c in self.constants)
        stmt_type = type(self.statement).__name__
        return f"CaseBranch({consts} => {stmt_type})"

```

Kelas CaseBranchNode

Satu cabang kasus, berisi daftar konstanta dan statement yang dijalankan.

```

● ● ●

class CaseNode(ASTNode):
    def __init__(self, expr_node, branches):
        super().__init__()
        self.expr_node = expr_node
        self.branches = branches
        self.children.append(expr_node)
        self.children.extend(branches)

    def __repr__(self):
        expr_type = type(self.expr_node).__name__
        return f"Case(expr={expr_type}, branches={len(self.branches)})"

```

Kelas CaseNode

Struktur kasus dengan ekspresi kondisi dan kumpulan cabang.

```

● ● ●

class ArrayTypeNode(ASTNode):
    def __init__(self, base_type, bounds):
        super().__init__()
        self.base_type = base_type
        self.bounds = bounds
    def __repr__(self):
        return f"ArrayType(base_type={self.base_type}, bounds={self.bounds})"

```

Kelas ArrayTypeNode

Tipe array berisi tipe dasar dan rentang indeks.

```

● ● ●

class ArrayAccessNode(ASTNode):
    def __init__(self, array, index):
        super().__init__()
        self.array = array
        self.index = index
        self.children.extend([array, index])

    def __repr__(self):
        if hasattr(self.array, 'name'):
            array_name = self.array.name
        elif isinstance(self.array, ArrayAccessNode):
            array_name = repr(self.array)
        else:
            array_name = "array"

        if hasattr(self.index, 'value'):
            index_val = self.index.value
        else:
            index_val = "index"

        return f"Array Variable(array={array_name}, index={index_val})"

```

Kelas ArrayAccessNode

Node akses elemen array dan index

| | |
|--|---|
| <pre><code>● ● ● class RecordFieldNode(ASTNode): def __init__(self, name, type_): super().__init__() self.name = name self.type_ = type_ def __repr__(self): return f"RecordField({self.name})"</code></pre> | <p>Kelas RecordFieldNode</p> <p>Satu field dalam tipe record dengan nama dan tipe.</p> |
| <pre><code>● ● ● class RecordTypeNode(ASTNode): def __init__(self, fields=None): super().__init__() self.fields = fields or [] self.children.extend(self.fields) def __repr__(self): return f"RecordType(fields={self.fields})"</code></pre> | <p>Kelas UnaryOpNode</p> <p>Tipe record berisi daftar field.</p> |
| <pre><code>● ● ● class RangeTypeNode(ASTNode): def __init__(self, lower, upper): super().__init__() self.lower = lower self.upper = upper self.children.extend([lower, upper]) def __repr__(self): lval = self.lower.value if hasattr(self.lower, 'value') else self.lower uval = self.upper.value if hasattr(self.upper, 'value') else self.upper return f"RangeType({lval}-{uval})"</code></pre> | <p>Kelas RangeTypeNode</p> <p>Tipe rentang.</p> |

4.3 AST Builder

Kelas ini mengandung semua fungsi yang digunakan dalam membangun AST.

| Functions | Penjelasan Singkat |
|---|---|
| <pre><code>● ● ● class ASTBuilder: def __init__(self, parse_root): self.parse_root = parse_root</code></pre> | <p>Kelas ASTBuilder</p> <p>Konstruktor ASTBuilder yang menerima akar parse tree (parse_root). Fungsi ini menyimpan parse root sebagai atribut kelas agar dapat digunakan saat membangun AST.</p> |
| <pre><code>● ● ● def build(self): return self.build_node(self.parse_root)</code></pre> | <p>Fungsi build</p> <p>Fungsi utama untuk memulai proses pembangunan AST. Ia memanggil build_node pada parse root dan mengembalikan node AST yang</p> |

| | |
|--|--|
| | <p>merepresentasikan keseluruhan program.</p> |
| <pre> def build_node(self, node): if isinstance(node, Token): return self.build_token(node) elif isinstance(node, ParseNode): controller_map = { "<program>": self.build_program_node, "<program-header>": lambda n: None, "<declaration-parts>": self.build_declarations_node, "<const-declaration>": self.build_const_decl_node, "<var-declaration>": self.build_var_decl_node, "<type-declaration>": self.build_type_declaration_node, "<record-type>": self.build_record_type_node, "<compound-statement>": self.build_block_node, "<assignment-statement>": self.build_assign_node, "<expression>": self.build_expression_node, "<simple-expression>": self.build_simple_expression_node, "<term>": self.build_term_node, "<factor>": self.build_factor_node, "<procedure-declaration>": self.build_procedure_decl_node, "<function-declaration>": self.build_function_decl_node, "<procedure/function-call>": self.build_procedure_function_call_node, "<variables>": self.build_variable_node, "<number>": self.build_number_node, "<string>": self.build_string_node, "<if-statement>": self.build_if_node, "<while-statement>": self.build_while_node, "<for-statement>": self.build_for_node, "<repeat-statements>": self.build_repeat_node, "<case-statement>": self.build_case_node } builder = controller_map.get(node.type) if builder: return builder(node) children_nodes = [self.build_node(c) for c in node.child if self.build_node(c)] if len(children_nodes) == 1: return children_nodes[0] return BlockNode(children_nodes) return None </pre> | <h3>Fungsi build_node</h3> <p>Fungsi pengendali (controller) untuk Syntax Directed Translation (SDT). Memeriksa tipe node parse tree, memanggil builder spesifik jika node merupakan non-terminal, atau build_token jika terminal. Jika tidak ada builder, ia membuat BlockNode dari child nodes sebagai fallback.</p> |
| <pre> def build_token(self, token): if token.type == "NUMBER": return NumNode(token.value) elif token.type in ["STRING_LITERAL", "CHAR_LITERAL"]: return StringNode(token.value) elif token.type == "IDENTIFIER": return VarNode(token.value) return None </pre> | <h3>Fungsi build_token</h3> <p>Membuat AST node dari token terminal. Token bertipe NUMBER menjadi NumNode, STRING_LITERAL/CHAR_LITERAL menjadi StringNode, dan IDENTIFIER menjadi VarNode. Token lain diabaikan.</p> |
| <pre> def build_number_node(self, node): token = next((t for t in node.child if isinstance(t, Token)), None) return NumNode(token.value) if token else None </pre> | <h3>Fungsi build_number_node</h3> <p>Mengambil token NUMBER dari child node dan membungkusnya menjadi NumNode. Digunakan untuk membangun AST dari ekspresi numerik.</p> |
| <pre> def build_string_node(self, node): token = next((t for t in node.child if isinstance(t, Token)), None) return StringNode(token.value) if token else None </pre> | <h3>Fungsi build_string_node</h3> <p>Mengambil token literal string dari child node dan membungkusnya menjadi StringNode. Berguna untuk literal string di program.</p> |

```

def build_variable_node(self, node):
    base_token = next((t for t in node.child if isinstance(t, Token) and t.type == "IDENTIFIER"), None)
    if not base_token:
        return None

    current_node = VarNode(base_token.value)

    i = 0
    while i < len(node.child):
        c = node.child[i]

        # Skip Identifier pertama yang sudah diambil
        if isinstance(c, Token) and c.value == base_token.value and i == 0:
            pass

        # DOT --> akses field
        elif isinstance(c, Token) and c.type == "DOT":
            next_child = node.child[i+1] if i < len(node.child) else None
            if next_child and (isinstance(next_child, Token) and next_child.type == "IDENTIFIER"):
                current_node = RecordFieldNode(next_child.value, current_node)
                i += 1

        # <variable-index> --> akses array
        elif getattr(c, "type", None) == "<variable-index>":
            expr_node = next((self.build_node(x) for x in c.child if getattr(x, "type", None) == "<expression>"), None)
            if expr_node:
                current_node = ArrayAccessNode(current_node, expr_node)

        i += 1

    return current_node

```

Fungsi build_variable_node

Membuat node variabel yang dapat menangani identifier dasar, field record (dengan DOT), dan akses array (<variable-index>). Traversal dilakukan untuk menggabungkan akses bertingkat menjadi AST yang lengkap.

```

def build_statement_list(self, nodes):
    statements = []
    for n in nodes:
        node = self.build_node(n)
        if node:
            if isinstance(node, BlockNode):
                statements.extend(node.children)
            else:
                statements.append(node)
    return BlockNode(statements) if len(statements) > 1 else (statements[0] if statements else None)

```

Fungsi build_statement_list

Membuat BlockNode dari list child nodes, memastikan jika terdapat nested block, child-nya di-flatten agar menghasilkan daftar statement linear.

```

def build_program_node(self, node):
    prog_name = None
    decls_attr = []
    block_attr = None

    for c in node.child:
        if c.type == "<program-header>":
            token = next((t for t in c.child if isinstance(t, Token) and t.type == "IDENTIFIER"), None)
            if token:
                prog_name = token.value
        elif c.type == "<declaration-part>":
            decl_node = self.build_node(c)
            decls_attr = [decl_node] if decl_node else []
        elif c.type == "<compound-statement>":
            block_attr = self.build_node(c)

    return ProgramNode(prog_name, decls_attr, block_attr)

```

Fungsi build_program_node

Membuat ProgramNode dari parse node <program>. Mengekstrak nama program dari <program-header>, deklarasi dari <declaration-part>, dan body dari <compound-statement>.

```

def build_declarations_node(self, node):
    decls = []
    for c in node.child:
        child_decl = self.build_node(c)
        if child_decl:
            if isinstance(child_decl, list):
                decls.extend(child_decl)
            else:
                decls.append(child_decl)
    return DeclarationsNode(decls)

```

Fungsi build_declarations_node

Menggabungkan berbagai deklarasi (const, var, type) menjadi DeclarationsNode. Menangani child yang bisa berupa list atau single node.

```

def build_const_decl_node(self, node):
    consts = []
    i = 0
    children = node.child

    while i < len(children):
        c = children[i]
        if isinstance(c, Token) and c.type == "IDENTIFIER":
            name = c.value
            j = i + 1
            while j < len(children) and children[j].type != "<expression>":
                j += 1
            expr_node = None
            if j < len(children) and children[j].type == "<expression>":
                expr_node = self.build_node(children[j])
            else:
                expr_node = None

            const_node = ConstDeclNode(name, expr_node)
            consts.append(const_node)

            k = j + 1
            while k < len(children) and not (isinstance(children[k], Token) and
                                             children[k].type == "SEMICOLON"):
                k += 1
            i = k + 1
            continue

        i += 1

    return consts

```

Fungsi build_const_decl_node

Membuat daftar ConstDeclNode dari parse node <const-declaration>. Mengambil identifier, mengekstrak ekspresi terkait, dan mengabaikan semicolon.

```

def build_var_decl_node(self, node):
    declarations = []
    identifiers_nodes = [c for c in node.child if c.type == "<identifier-list>"]
    type_nodes = [c for c in node.child if c.type == "<type>"]

    for id_node, type_node in zip(identifiers_nodes, type_nodes):
        vartype = self.build_type_definition_node(type_node)

        for item in id_node.child:
            if item.type == "IDENTIFIER":
                declarations.append(VarDeclNode(item.value, vartype))

    return declarations

```

Fungsi build_var_decl_node

Membuat daftar VarDeclNode dari <var-declaration>. Mengambil <identifier-list> dan tipe, kemudian menggabungkannya untuk tiap variabel yang dideklarasikan.

```

def build_type_definition_node(self, node):
    if node is None:
        return None

    # handle token
    if isinstance(node, Token):
        if node.type in ("IDENTIFIER", "KEYWORD"):
            return node.value
        return None

    # handle range
    expr_nodes = [c for c in getattr(node, "child", []) if getattr(c, "type", None) == "<expression>"]
    range_op = next((c for c in getattr(node, "child", []) if (isinstance(c, Token) and c.type == "RANGE_OPERATOR"), None))
    if len(expr_nodes) == 2 and range_op:
        lower = self.build_node(expr_nodes[0])
        upper = self.build_node(expr_nodes[1])
        return RangeTypeNode(lower, upper)

    # handle record / array / primitive
    for c in getattr(node, "child", []):
        if getattr(c, "type", None) == "<record-type>":
            return self.build_record_type_node(c)
        if getattr(c, "type", None) == "<array-type>":
            bounds = []
            range_node = next((x for x in c.child if getattr(x, "type", None) == "<range>"), None)
            if range_node:
                lower = self.build_node(range_node.child[0])
                upper = self.build_node(range_node.child[2])

            bounds.append(lower, upper)
            base_type_node = next((x for x in c.child if getattr(x, "type", None) == "<type>"), None)
            base_type_node = self.build_type_definition_node(base_type_node) if base_type_node else None
            return ArrayTypeNode(bounds, base_type_node)

        if isinstance(c, Token) and c.type in ("IDENTIFIER", "KEYWORD"):
            return c.value
        if res:
            res.append(c)
        return res

    return None

```

Fungsi build_type_definition_node

Mengonversi node tipe parse tree menjadi tipe AST. Menangani primitive, range, record, array, dan rekursi untuk nested type.

```

def build_type_declaration_node(self, node):
    type_decls = []
    if not hasattr(self, "type_table"):
        self.type_table = {}
    i = 0
    while i < len(node.child):
        if node.child[i].type == "IDENTIFIER" and i+2 < len(node.child):
            identifier = node.child[i].value
            if node.child[i+1].type == "RELATIONAL_OPERATOR" and node.child[i+2].value == "=":
                type_def_node = node.child[i+2]
                type_def = self.build_type_definition_node(type_def_node)
                type_decl = TypeDeclarationNode(identifier, type_def)
                type_decls.append(type_decl)
                self.type_table[identifier] = type_def # simpan di type table
            i += 3
            if i < len(node.child) and node.child[i].type == "SEMICOLON":
                i += 1
            continue
        i += 1
    return type_decls

```

Fungsi build_type_declaration_node

Membuat daftar TypeDeclarationNode dari <type-declaration>. Menyimpan definisi tipe ke type_table untuk referensi internal.

```

def build_record_type_node(self, node):
    fields = []
    for group in node.child:
        if group.type == "<parameter-group>":
            id_list_node = next((c for c in group.child if c.type == "<identifier-list>"), None)
            ids = [t.value for t in getattr(id_list_node, "child", [])] if isinstance(t, Token) and t.type == "IDENTIFIER" else []
            type_node = next((c for c in group.child if c.type == "<types>"), None)
            field_type = self.build_type_definition_node(type_node) if type_node else None
            for name in ids:
                fields.append(RecordFieldNode(name, field_type))
    return RecordTypeNode(fields)

```

Fungsi build_record_type_node

Membuat RecordTypeNode yang berisi field-field. Field diekstrak dari <parameter-group> dengan identifier dan tipe masing-masing, lalu dibungkus menjadi RecordFieldNode.

```

def build_block_node(self, node):
    statements_nodes = [c for c in node.child if c.type not in ("KEYWORD",)]
    statements = []
    for statement in statements_nodes:
        statement_node = self.build_node(statement)
        if isinstance(statement_node, BlockNode):
            statements.extend(statement_node.children)
        elif statement_node:
            statements.append(statement_node)
    return BlockNode(statements)

```

Fungsi build_block_node

Membuat BlockNode dari <compound-statement> dengan mengekstrak semua statement child. Nested block di-flatten agar menghasilkan daftar statement linear.

```

def build_variable_access(self, node):
    base_token = next((t for t in node.child if isinstance(t, Token) and t.type == "IDENTIFIER"), None)
    if not base_token:
        return None
    current_node = VarNode(base_token.value)

    i = 0
    while i < len(node.child):
        c = node.child[i]
        if c.type == "variable-index":
            expr_node = next((self.build_node(x) for x in c.child if x.type == "<expression>"), None)
            current_node = ArrayAccessNode(current_node, expr_node)
        elif isinstance(c, Token) and c.type == "DOT":
            i += 1
            field_token = node.child[i] if i < len(node.child) else None
            if field_token and isinstance(field_token, Token) and field_token.type == "IDENTIFIER":
                current_node = RecordFieldNode(field_token.value, current_node)
        i += 1
    return current_node

```

Fungsi build_variable_access

Membuat node akses variabel alternatif (mirip build_variable_node). Menangani identifier, akses array, dan field record.

```

def build_assign_node(self, node):
    target_node = next((self.build_node(c) for c in node.child if c.type == "<variable>"), None)
    value_node = next((self.build_node(c) for c in node.child if c.type == "<expression>"), None)
    if not target_node or not value_node:
        raise ASTError("Assignment statement incomplete", node)
    return AssignNode(target_node, value_node)

```

Fungsi build_type_definition_node

Membuat AssignNode dari <assignment-statement>. Menentukan target (variabel) dan value (ekspresi). Jika salah satu tidak lengkap, menimbulkan ASTError.

```

def build_expression_node(self, node):
    simple_exprs = [self.build_simple_expression_node(c) for c in node.child if c.type == "<simple-expression>"]
    ops = [c for c in node.child if isinstance(c, Token) and c.type in ("ARITHMETIC_OPERATOR", "RELATIONAL_OPERATOR")]

    if len(simple_exprs) == 2 and ops:
        return BinOpNode(simple_exprs[0], ops[0].value, simple_exprs[1])
    elif simple_exprs:
        return simple_exprs[0]
    return None

```

Fungsi build_expression_node

Membuat node ekspresi dari <expression>. Mengambil simple expressions dan operator, membangun BinOpNode jika ada dua operand, atau mengembalikan single operand jika tunggal.

```

def build_simple_expression_node(self, node):
    # Ambil semua term dan operator
    terms = [self.build_term_node(c) for c in node.child if c.type == "<term>"]
    ops = [c for c in node.child if isinstance(c, Token) and c.type == "ARITHMETIC_OPERATOR"]

    if not terms:
        return None

    # Cek apakah operasi pertama adalah Unary
    first_child = node.child[0]
    is_unary_start = isinstance(first_child, Token) and first_child.type == "ARITHMETIC_OPERATOR"

    current = None
    term_idx = 0
    op_idx = 0

    # SPLIT UNARY VS BINARY
    if is_unary_start:
        op_val = ops[0].value
        term_val = terms[0]
        current = UnaryOpNode(op_val, term_val)

        term_idx = 1
        op_idx = 1
    else:
        current = terms[0]
        term_idx = 1
        op_idx = 0

    while op_idx < len(ops):
        if term_idx >= len(terms):
            break

        op_val = ops[op_idx].value
        right_val = terms[term_idx]
        current = BinOpNode(current, op_val, right_val)

        op_idx += 1
        term_idx += 1

    return current

```

Fungsi build_simple_expression_node

Membuat node simple expression dari <simple-expression>. Menangani unary dan binary operator, looping manual untuk menggabungkan term menjadi BinOpNode atau UnaryOpNode.

```

def build_term_node(self, node):
    factors = [self.build_factor_node(c) for c in node.child if c.type == "<factor>"]
    ops = [c for c in node.child if isinstance(c, Token) and c.type == "ARITHMETIC_OPERATOR"]
    if not ops:
        return factors[0] if factors else None
    current = factors[0]
    for i, op in enumerate(ops):
        current = BinOpNode(current, op.value, factors[i+1])
    return current

```

Fungsi build_term_node

Membuat node term dari <term>. Menggabungkan faktor dengan operator aritmatika menjadi BinOpNode.

```

def build_factor_node(self, node):
    if any(getattr(c, 'value', '').lower() == "tidak" for c in node.child):
        Factor = next((self.build_factor_node(c) for c in node.child if getattr(c, 'value', '').lower() != "tidak"), None)
        return UnaryOpNode("tidak", Factor)

    if node.type == "BOOLEAN_LITERAL":
        token = next(t for t in node.child if isinstance(t, Token) and t.type == "KEYWORD" and t.value.lower() in ("true", "false"), None)
        if token:
            return BooleanNode(token.value.lower() == "true")

    # Jika ada logical operator (dan/atau)
    logical_ops = [t for t in node.child if isinstance(t, Token) and t.type == "LOGICAL_OPERATOR"]
    if logical_ops:
        left = self.build_factor_node(node.child[0])
        for i, op in enumerate(logical_ops):
            right = self.build_factor_node(node.child[i+1])
            left = BinOpNode(left, op.value.lower(), right)
        return left

    # Parentheses
    if any(c.type == "LPARENTHESIS" for c in node.child):
        expr = next((self.build_expression_node(c) for c in node.child if getattr(c, 'type', None) == "<expressions>"), None)
        return expr

    # Default fallback
    return self.build_node(node.child[0])

```

Fungsi build_factor_node

Membuat node faktor dari <factor>. Menangani unary (tidak), literal boolean, logical operator (dan/atau), parenthesis, dan fallback ke child pertama.

```

def build_param_list(self, node):
    params = []
    if node == None:
        return params
    for child in node.child:
        if child.type == "parameter-group":
            is_var = False
            for grandchild in child.child:
                if grandchild.type == "identifier-list":
                    is_var = True
                    break
            # Ambil identifier list
            id_list_node = next((c for c in child.child if c.type == "<identifier-list>"), None)
            ids = [t.value for t in id_list_node.child if isinstance(t, Token) and t.type == "IDENTIFIER"] if id_list_node else []
            if id_list_node:
                is_var = True
                break
            # Ambil type
            type_node = next((c for c in child.child if c.type == "<type>"), None)
            var_type = None
            if type_node:
                t = next((tk for tk in type_node.child if hasattr(tk, "value")), None)
                var_type = t.value if t else None
            for name in ids:
                params.append(ParamNode(name, var_type, is_var))
    return params

```

Fungsi build_param_list

Membuat daftar parameter (ParamNode) dari <formal-parameter-list>. Mengecek modifier variabel, identifier list, dan tipe parameter.

```

def build_procedure_decl_node(self, node):
    # Nama procedure
    name_token = next((t for t in node.child if isinstance(t, Token) and t.type == "IDENTIFIER"), None)
    name = name_token.value

    # Parameter list
    param_list_node = next((c for c in node.child if c.type == "<formal-parameter-list>"), None)
    params = self.build_param_list(param_list_node)

    # Block (isi prosedur)
    block_node = next((c for c in node.child if c.type in ("<block>", "<compound-statement>"), None))
    block = self.build_node(block_node)

    return ProcedureDeclNode(name, params, block)

```

Fungsi build_procedure_decl_node

Membuat ProcedureDeclNode dari parse node procedure. Mengekstrak nama, parameter list, dan body block.

```

def build_function_decl_node(self, node):
    # Nama function
    name_token = next((t for t in node.child if isinstance(t, Token) and t.type == "IDENTIFIER"), None)
    name = name_token.value

    # Parameter list
    param_list_node = next((c for c in node.child if c.type == "<formal-parameter-list>"), None)
    params = self.build_param_list(param_list_node)

    # Return type
    type_node = next((c for c in node.child if c.type == "<type>"), None)
    return_type = None
    if type_node:
        t = next((tk for tk in type_node.child if hasattr(tk, "value")), None)
        return_type = t.value if t else None

    # Function body (block)
    block_node = next((c for c in node.child if c.type in ("<block>", "<compound-statements>"), None))
    block = self.build_node(block_node)

    return FunctionDeclNode(name, params, return_type, block)

```

Fungsi build_function_decl_node

Membuat FunctionDeclNode dari parse node function. Mengekstrak nama, parameter list, return type, dan body block.

```

def build_procedure_function_call_node(self, node):
    name_token = next((t for t in node.child if isinstance(t, Token) and t.type == "IDENTIFIER"), None)
    if not name_token:
        raise ASTError("Procedure call missing name", node)

    param_list_node = next((c for c in node.child if c.type == "<parameter-list>"), None)
    args = []
    if param_list_node:
        for expr_node in param_list_node.child:
            built_expr = self.build_node(expr_node)
            if built_expr:
                args.append(built_expr)

    return ProcedureFunctionCallNode(name_token.value, args)

```

Fungsi build_procedure_function_call_node

Membuat ProcedureFunctionCallNode. Mengekstrak nama prosedur/function dan argumen dari <parameter-list>.

```

def build_if_node(self, node):
    idx_maka = next((i for i, c in enumerate(node.child) if c.type == "KEYWORD" and c.value.lower() == "jika")
    idx_maka = next((i, c for i, c in enumerate(node.child) if c.type == "KEYWORD" and c.value.lower() == "maka")
    idx_selain_ltu = next((i for i, c in enumerate(node.child) if c.type == "KEYWORD" and c.value.lower() == "selain_ltu"), None)

    expr_node = next((c for c in node.child[(idx_jika + 1):idx_maka] if c.type == "<expression>"))
    condition = self.build_node(expr_node)

    if idx_selain_ltu:
        then_nodes = node.child[idx_maka + 1:idx_selain_ltu]
        else_nodes = node.child[idx_selain_ltu + 1:]
    else:
        then_nodes = node.child[idx_maka + 1:]
        else_nodes = []

    then_body = self.build_statement_list(then_nodes)
    else_body = self.build_statement_list(else_nodes) if else_nodes else None

    return IfNode(condition=condition, then_block=then_body, else_block=else_body)

```

Fungsi build_if_node

Membuat IfNode dari <if-statement>. Menentukan kondisi dari ekspresi antara jika dan maka, lalu membuat then-block dan optional else-block.

```

def build_case_node(self, node):
    expr_node = next((self.build_node(c) for c in node.child if getattr(c, "type") == "<expression>"), None)
    case_list_node = next((c for c in node.child if getattr(c, "type") == "<case-list>"), None)
    branches = []
    if case_list_node:
        i = 0
        children = case_list_node.child
        while i < len(children):
            # ambil value case
            value_token = children[i]
            value_node = self.build_node(value_token) if hasattr(value_token, "type") else None
            i += 1
            # skip COLON
            if i < len(children) and isinstance(children[i], Token) and children[i].type == "COLON":
                i += 1
            # ambil statement setelah colon
            stmt_nodes = []
            while i < len(children) and not (isinstance(children[i], Token) and children[i].type in ("NUMBER", "IDENTIFIER")):
                stmt_nodes.append(children[i])
                i += 1
            stmt_node = self.build_statement_list(stmt_nodes)
            branches.append(CaseBranchNode(value_node, stmt_node))
    return CaseNode(expr_node, branches)

```

Fungsi build_case_node

Membuat CaseNode dari <case-statement>. Mengambil ekspresi utama, lalu membangun daftar CaseBranchNode dengan value dan statement.

```

def build_while_node(self, node):
    expr_node = next((c for c in node.child if c.type == "<expression>"), None)
    if not expr_node:
        raise ASTError("While statement missing condition", node)
    condition = self.build_node(expr_node)

    body_candidates = [c for c in node.child if c.type not in ("KEYWORD", "<expression>")]
    if not body_candidates:
        raise ASTError("While statement missing body", node)
    body = self.build_statement_list(body_candidates)

    return WhileNode(condition, body)

```

Fungsi build_while_node

Membuat WhileNode dari <while-statement>. Mengekstrak kondisi dan body statements. Menimbulkan ASTError jika salah satu tidak ada.

```

def build_repeat_node(self, node):
    body = next((self.build_node(c) for c in node.child if c.type == "<statement-list>"), None)
    condition = next((self.build_node(c) for c in node.child if c.type in ("<expression>", "<factors>")), None)
    if not body or not condition:
        raise ASTError("Repeat statement incomplete", node)
    return RepeatNode(body, condition)

```

Fungsi build_repeat_node

Membuat RepeatNode dari <repeat-statement>. Mengekstrak body statements dan kondisi, memvalidasi kelengkapan.

```

def build_for_node(self, node):
    var_node = next((self.build_node(c) for c in node.child if c.type == "IDENTIFIER"), None)
    start_expr = next((self.build_node(c) for c in node.child if c.type == "<expression>"), None)
    direction_token = next((c for c in node.child if getattr(c, "value", "").lower() in ("ke", "turun_ke")), None)
    end_expr = None
    if direction_token:
        idx = node.child.index(direction_token)
        for c in node.child[idx+1:]:
            if c.type == "<expression>":
                end_expr = self.build_node(c)
                break
    lakukan_idx = next((i for i, c in enumerate(node.child) if getattr(c, "value", "").lower() == "lakukan"), None)
    body_nodes = node.child[lakukan_idx+1:] if lakukan_idx is not None else []
    body_node = self.build_statement_list(body_nodes)

    direction = getattr(direction_token, "value", "ke").lower() if direction_token else "ke"
    return ForNode(var_node, start_expr, end_expr, direction, body_node)

```

Fungsi for_node

Membuat ForNode dari <for-statement>. Mengekstrak variabel loop, start/end expression, arah iterasi (ke/turun_ke), dan body.

4.4 Symbol Table

Kelas ini merupakan otak atau memori dari compiler itu sendiri, segala variabel yang dideklarasikan akan tersimpan pada tabel ini, dalam bentuk data tertentu mencakup berbagai informasi.

| Class | Penjelasan Singkat |
|---|--|
| <pre> 1 class SymbolTable: 2 def __init__(self): 3 self.tab: List[Dict[str, Any]] = [] 4 self.btab: List[Dict[str, Any]] = [] 5 self.atab: List[Dict[str, Any]] = [] 6 self.display: List[int] = [] 7 self.current_level = 0 8 9 for word in ReservedWords.LIST: 10 self.tab.append({ 11 "name": word, 12 "link": 0, 13 "obj": ObjKind.RESERVED, 14 "type": TypeKind.NOTYPE, 15 "ref": 0, 16 "nrm": 0, 17 "lev": 0, 18 "adr": 0 19 }) 20 21 self.btab.append({ 22 "index": 0, 23 "last": 0, 24 "lpar": 0, 25 "psze": 0, 26 "vsze": 0 27 }) 28 self.display = [0] </pre> | <p>Kelas SymbolTable memiliki 3 tabel yang terpisah :</p> <p>tab : tabel utama yang menyimpan seluruh deklarasi variabel dari nama , jenis, tipe, level, dan alamat memori relatif. tabel ini bentuknya list panjang yang cara kerjanya seperti linked list, menghubungkan variabel-variabel dalam satu fungsi yang sama</p> <p>btab : tabel ini mengelola informasi per blok atau fungsi , isinya berupa indeks variabel terakhir yang ada di tab milik fungsinya, psze yang merupakan jumlah parameter/argumen yang dimiliki fungsi, dan vsze yang merupakan jumlah memori yang dibutuhkan untuk variabel lokal di dalam fungsi ini</p> <p>atab: berisi informasi mengenai suatu array dari batas bawah (low), batas atas (high) , tipe elemen (etyp), dan total ukuran memori (size)</p> |
| <pre> 1 def enter_scope(self): 2 """Masuk procedure / function""" 3 self.current_level += 1 4 new_block_index = len(self.btab) 5 6 self.btab.append({ 7 "index": new_block_index, 8 "last": 0, 9 "lpar": 0, 10 "psze": 0, 11 "vsze": 0 12 }) 13 14 # Update Display Vector 15 if len(self.display) <= self.current_level: 16 self.display.append(new_block_index) 17 else: 18 self.display[self.current_level] = new_block_index 19 return new_block_index 20 21 def exit_scope(self): 22 """Keluar procedure / function""" 23 if self.current_level > 0: 24 self.current_level -= 1 </pre> | <p>enter_scope dipanggil saat masuk ke scope baru seperti deklarasi fungsi/prosedur baru, current_level akan increment naik 1 tingkat (basenya 0 di global), disini juga bakal bikin btab baru karena masuk ke block baru milik fungsi / prosedur.</p> <p>exist_scope dipanggil saat keluar ruangan , saat parser menemukan end, current_level decrement. btab disimpan dan tidak dihapus sebagai memori.</p> |

```

1 # Add Entries
2 def add_variable(self, name: str, type_kind: int, size: int = 1):
3     """Menambah variabel level"""
4     current_btob_idx = self.display[self.current_level]
5     current_block = self.btab[current_btob_idx]
6
7     t_idx = Link.LIST
8     link_idx = current_block["list"]
9     adr = current_block["vsze"]
10
11    new_entry = {
12        "name": name,
13        "link": link_idx,
14        "obj": ObjKind.VARIABLE,
15        "type": type_kind,
16        "ref": 0,
17        "size": size,
18        "lev": self.current_level,
19        "adr": adr
20    }
21    self.tab.append(new_entry)
22
23    # update block
24    new_id_idx = len(self.tab) - 1
25    current_block["last"] = new_id_idx
26    current_block["vsze"] += size
27    return new_id_idx
28
29 def add_parameter(self, name: str, type_kind: int, is_vor: bool = False):
30     """Menambah parameter prosedur/fungsi"""
31     current_btob_idx = self.display[self.current_level]
32     current_block = self.btab[current_btob_idx]
33
34     link_idx = current_block["list"]
35     nm_vo1 = 0 if is_vor else 1
36
37    new_entry = {
38        "name": name,
39        "link": link_idx,
40        "obj": ObjKind.VARIABLE,
41        "type": type_kind,
42        "ref": 0,
43        "nm_vo": nm_vo1,
44        "lev": self.current_level,
45        "adr": current_block["psze"]
46    }
47    self.tab.append(new_entry)
48
49    # Update block (last + 1)
50    new_id_idx = len(self.tab) - 1
51    current_block["last"] = new_id_idx
52    current_block["list"] = new_id_idx
53    current_block["psze"] += 1
54    return new_id_idx
55
56 def add_program_name(self, name: str):
57     ...
58     Khusus untuk nama program: Masukkan ke tabel topi JIKAADA update "last".
59     Agar variabel percontoh (a) nanti Link-nya ke b, bukan ke Hello.
60     ...
61    new_entry = {
62        "name": name,
63        "link": 0,
64        "obj": ObjKind.PROGRAM,
65        "type": Typekind.NOTYPE,
66        "ref": 0,
67        "nm": 0,
68        "lev": 0,
69        "adr": 0
70    }
71    self.tab.append(new_entry)
72    return len(self.tab) - 1
73
74 def add_constant(self, name: str, type_kind: int, value: Any):
75     """Add constant"""
76     current_btob_idx = self.display[self.current_level]
77     current_block = self.btab[current_btob_idx]
78     link_idx = current_block["list"]
79
80    new_entry = {
81        "name": name,
82        "link": link_idx,
83        "obj": ObjKind.CONSTANT,
84        "type": type_kind,
85        "ref": 0,
86        "nm": 1,
87        "lev": self.current_level,
88        "adr": value
89    }
90    self.tab.append(new_entry)
91
92    new_id_idx = len(self.tab) - 1
93    current_block["last"] = new_id_idx
94    return new_id_idx
95
96 def add_array_info(self, typ: int, eltyp: int, low: int, high: int, size: int):
97     """Menambah Info Array ke tab"""
98     size = (high - low + 1) * elsz
99     entry = [
100         {"index": len(self.tab) + 1,
101          "typ": typ,
102          "eltyp": eltyp,
103          "ref": 0,
104          "low": low,
105          "high": high,
106          "size": elsz,
107          "size": size
108      }
109     self.tab.append(entry)
110     return len(self.tab) - 1
111
112 def add_procedure(self, name: str, kind: str):
113     """Teman nama fungsi / prosedur di scope saat ini"""
114     current_btob_idx = self.display[self.current_level]
115     current_block = self.btab[current_btob_idx]
116
117     link_idx = current_block["list"]
118
119    new_entry = {
120        "name": name,
121        "link": link_idx,
122        "obj": kind,
123        "type": Typekind.NOTYPE, # Nanti diupdate kalo function (return type)
124        "ref": 0,
125        "nm": 0,
126        "lev": self.current_level,
127        "adr": 0
128    }
129    self.tab.append(new_entry)
130
131    new_id_idx = len(self.tab) - 1
132    current_block["last"] = new_id_idx
133    return new_id_idx
134
135 def add_builtin_procedure(self, name: str):
136     """Teman nama fungsi / prosedur built-in (predefined) ke symbol table di global scope"""
137     global_btob = self.btab[0]
138     link_idx = 0
139
140    new_entry = {
141        "name": name,
142        "link": link_idx,
143        "obj": ObjKind.PROCEDURE,
144        "type": Typekind.NOTYPE,
145        "ref": 0,
146        "nm": 0,
147        "lev": 0,
148        "adr": 0
149    }
150    self.tab.append(new_entry)
151    new_id_idx = len(self.tab) - 1
152
153    return new_id_idx
154
155 def add_type(self, name: str, type_kind: int):
156     current_btob_idx = self.display[self.current_level]
157     current_block = self.btab[current_btob_idx]
158     link_idx = current_block["list"]
159    new_entry = [
160        {"name": name,
161         "link": link_idx,
162         "obj": ObjKind.TYPE,
163         "type": type_kind,
164         "ref": 0,
165         "nm": 1,
166         "lev": self.current_level,
167         "adr": 0
168     }
169    self.tab.append(new_entry)
170    new_id_idx = len(self.tab) - 1
171    current_block["last"] = new_id_idx
172    return new_id_idx

```

add_variable & add_parameter:

fungsi untuk mendaftarkan variabel/parameter, otomatis menghitung alamat memori (adr) dan mengupdate ukuran memori yang dibutuhkan fungsi tersebut (vsze / psze) di btab

add_constant:

sama, bedanya adr tidak dipakai untuk alamat memori, tapi untuk menyimpan nilai konstanta nya langsung

add_array_info:

Khusus array, detailnya (batas bawah, atas, tipe elemen) disimpan terpisah di tabel atab, bukan tab utama

add_procedure & add_program_name:

Mendaftarkan nama fungsi/program itu sendiri sebagai head dari sebuah blok kode

add_builtin_procedure:

mendaftarkan fungsi bawaan seperti write, read secara paksa di level 0 (global) agar bisa dipanggil dari mana saja.

add_type:

Digunakan untuk menangani user-defined type, jadi declare type sendiri bukan built-in dari pascal-s nya.

```

1 # Lookup
2     def lookup(self, name: str) -> Optional[Dict]:
3         for level in range(self.current_level, -1, -1):
4             btab_idx = self.display[level]
5             current_id_idx = self.btab[btab_idx]["last"]
6
7             while current_id_idx > 0:
8                 entry = self.tab[current_id_idx]
9                 if entry["name"] == name:
10                     return entry
11                 current_id_idx = entry["link"]
12
13             reserved_count = ReservedWords.count()
14             for i in range(reserved_count):
15                 if self.tab[i]["name"] == name:
16                     return self.tab[i]
17
18             return None
19
20     def lookup_current_scope(self, name: str) -> Optional[Dict]:
21         btab_idx = self.display[self.current_level]
22         current_id_idx = self.btab[btab_idx]["last"]
23
24         while current_id_idx > 0:
25             entry = self.tab[current_id_idx]
26             if entry["name"] == name:
27                 return entry
28             current_id_idx = entry["link"]
29

```

lookup:

Fungsi ini digunakan saat kita menggunakan variabel di dalam kode (misal $x := y + 1$). compiler harus mencari definisi variabel y . mulai dari `current_level`, cek parent jika tidak ketemu, cek global jika tidak ketemu, cek `reserved words` jika masih tidak ketemu. intinya akan terus mundur ke level di atasnya.

lookup_current_scope:

Fungsi ini digunakan saat kita mendeklarasikan variabel baru. Jadi cara kerjanya ia melihat dan mengecek apakah sudah ada variabel yang sama dideklarasikan di scope yang sama.berbeda dengan `lookup` dia tidak loop mundur ke level induknya,

```

1     def get_parameters(self, symbol: Dict) -> List[Dict]:
2         if symbol['obj'] not in [ObjKind.PROCEDURE, ObjKind.FUNCTION]:
3             return []
4
5         block_ref = symbol.get('ref', 0)
6         if block_ref <= 0 or block_ref >= len(self.btab):
7             return []
8
9         block = self.btab[block_ref]
10        params = []
11        param_count = block.get('psze', 0)
12        if param_count == 0:
13            return []
14
15        current_id_idx = block.get('lpar', 0)
16        for _ in range(param_count):
17            if current_id_idx > 0 and current_id_idx < len(self.tab):
18                params.append(self.tab[current_id_idx])
19                current_id_idx = self.tab[current_id_idx]['link']
20
21        params.reverse()
22        return params

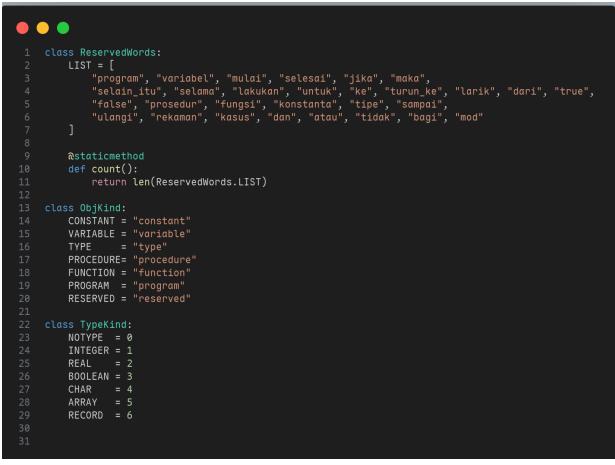
```

get_parameters:

Fungsi yang bertugas untuk mengambil daftar parameter (argumen) yang dimiliki oleh sebuah prosedur atau fungsi. Misal untuk suatu fungsi Hitung (`a:int,b:int`), fungsi inilah yang akan mereturn informasi mengenai parameter pada Hitung pada kasus ini adalah `a` dan `b`.

4.5 Constants

Terdapat 3 kelas yang masing2 bertugas untuk menyimpan suatu hardcoded constants yang akan digunakan nantinya pada symbol table untuk keperluan declaration dan type checking

| Class | Penjelasan Singkat |
|---|---|
|  <pre>1 class ReservedWords: 2 LIST = [3 "program", "varlabel", "mulai", "selesai", "jika", "maka", 4 "selain_itu", "selama", "lakukan", "untuk", "ke", "turun_ke", "larik", "dari", "true", 5 "false", "prosedur", "fungsi", "konstanta", "tipe", "sampai", 6 "ulangi", "rekanan", "kasus", "dan", "atau", "tidak", "bagi", "mod" 7] 8 9 @staticmethod 10 def count(): 11 return len(ReservedWords.LIST) 12 13 class ObjKind: 14 CONSTANT = "constant" 15 VARIABLE = "variable" 16 TYPE = "type" 17 PROCEDURE = "procedure" 18 FUNCTION = "function" 19 PROGRAM = "program" 20 RESERVED = "reserved" 21 22 class TypeKind: 23 NOTYPE = 0 24 INTEGER = 1 25 REAL = 2 26 BOOLEAN = 3 27 CHAR = 4 28 ARRAY = 5 29 RECORD = 6 30 31</pre> | <p>Class ReservedWords: merupakan daftar konstanta identifier list yang digunakan agar user tidak membuat variabel dengan nama yang sudah di reserved pada list</p> <p>Class objKind: merupakan kelas yang disimpan untuk menyimpan identifier yang berupa suatu jenis objek yang mencakup konstanta, variabel , prosedur/fungsi, tipe, dan program.</p> <p>Class TypeKind merupakan kelas yang digunakan untuk menyimpan tipe data builtin pada program yang mencakup tipe data primitif seperti integer, real, boolean, char, dan juga tipe data kompleks/struktur seperti record dan array.</p> |

4.6 TypeChecker

Kelas ini digunakan untuk melakukan type check

| Class | Penjelasan Singkat |
|--|---|
| <pre> 1 class TypeChecker: 2 """Menangani pemeriksaan tipe untuk ekspresi, pernyataan (statements), dan deklarasi""" 3 4 def __init__(self): 5 self.type_names = { 6 TypeKind.INTEGER: "integer", 7 TypeKind.REAL: "real", 8 TypeKind.BOOLEAN: "boolean", 9 TypeKind.CHAR: "char", 10 TypeKind.ARRAY: "array", 11 TypeKind.RECORD: "record", 12 TypeKind.NOTYPE: "notype" 13 } </pre> | <p>Class type checker bertugas memvalidasi keamanan tipe data (type safety) dalam compiler untuk mencegah illegal operation didalam terdapat kamus dictionary bernama <code>type_names</code> yang berfungsi sebagai penerjemah kode numerik internal, menjadi teks yang mudah dibaca manusia seperti “integer” atau ‘real’.</p> |
| <pre> 1 def get_field_type(self, record_type, field_name, lineNone, columnNone): 2 if isinstance(record_type, dict) and record_type.get('kind') == TypeKind.RECORD: 3 if field_name not in record_type['fields']: 4 raise TypeMismatchError(f"field '{field_name}' not found", "field access", line, column) 5 return record_type['fields'][field_name] 6 7 8 if record_type == TypeKind.RECORD: 9 raise TypeMismatchError("record", "record.get_type(record_type)", "field access 'record'", line, column) 10 raise TypeMismatchError("record", self.get_type_name(record_type), "field access 'record'", line, column) 11 12 def get_type_name(self, type_kind): 13 """Typekind menjadi string yang mudah dibaca""" 14 return self.type_names.get(type_kind, f"unknown({type_kind})") </pre> | <p>get_field_type: Fungsi dipanggil saat semantic analyzer menemukan syntax A.B, mengecek apakah A (<code>record_type</code>) benar-benar bertipe record yang lengkap. Jika A memang record, apakah diap unya field bernama B.</p> <p>get_type_name: bertugas menerjemahkan code angka internal seperti 1, menjadi teks yang bisa dibaca manusia. (seperti “integer”)</p> |
| <pre> 1 def is_numeric_type(self, type_kind): 2 """Periksa apakah tipe bersifat numerik (integer atau real)""" 3 return type_kind in [TypeKind.INTEGER, TypeKind.REAL] 4 5 def is_ordinal_type(self, type_kind): 6 """Periksa apakah tipe bersifat ordinal (integer, boolean, char)""" 7 return type_kind in [TypeKind.INTEGER, TypeKind.BOOLEAN, TypeKind.CHAR] 8 9 def is_compatible(self, type1, type2): 10 """Periksa apakah dua tipe kompatibel untuk penugasan (assignment)""" 11 # Tipe yang sama selalu kompatibel 12 if type1 == type2: 13 return True 14 15 # Integer dapat ditugaskan ke real 16 if type1 == TypeKind.REAL and type2 == TypeKind.INTEGER: 17 return True 18 19 return False </pre> | <p>is_numeric_type: Mengecek apakah sebuah tipe bisa dipakai untuk numeric (tambah, kurang, kali bagi)</p> <p>is_ordinal_type: Mengecek apakah tipe data tersebut memiliki urutan yang pasti</p> <p>is_compatible: Mengecek apakah tipe2 boleh dimasukan ke dalam variabel bertipe tipe1</p> |

```

1 def check_arithmetic_operation(self, op, left_type, right_type, line=None, column=None):
2     """Periksa apakah operasi aritmetika valid untuk tipe yang diberikan. Mengembalikan tipe hasil operasi."""
3     # Kondisi operasi aritmetika
4     if not self.is_numeric_type(right_type):
5         raise InvalidOperationError(
6             op, self.get_type_name(left_type),
7             self.get_type_name(right_type), line, column
8         )
9
10    if not self.is_numeric_type(left_type):
11        raise InvalidOperationError(
12            op, self.get_type_name(left_type),
13            self.get_type_name(right_type), line, column
14        )
15
16    # Operasi bagi (div) dan mod pada integer menghasilkan integer
17    if op in ['bagi', 'mod']:
18        if left_type == TypeKind.INTEGER or right_type != TypeKind.INTEGER:
19            raise TypeMismatchError(
20                "integer",
21                f"({self.get_type_name(left_type)} and {self.get_type_name(right_type)})",
22                op, line, column
23            )
24
25    return TypeKind.INTEGER
26
27    # Jika salah satu operand real, hasil adalah real
28    if left_type == TypeKind.REAL or right_type == TypeKind.REAL:
29        return TypeKind.REAL
30
31    # Kondisi integer, hasil integer (kecuali pembagian '/')
32    if op == '/':
33        return TypeKind.REAL # Division always returns real in Pascal
34
35    return TypeKind.INTEGER
36
37 def check_relational_operation(self, op, left_type, right_type, line=None, column=None):
38     """Periksa apakah operasi relasional valid untuk tipe yang diberikan. Mengembalikan tipe boolean."""
39     # Operator equality bekerja pada tipe apa pun (salah sama)
40     if op in ['sama', 'berbeda']:
41         if left_type != right_type:
42             raise TypeMismatchError(
43                 self.get_type_name(left_type),
44                 self.get_type_name(right_type),
45                 f"comparison ({op})", line, column
46             )
47
48     return TypeKind.BOOLEAN
49
50     # Operator perbandingan urutan memerlukan tipe ordinal
51     if op in ['<', '<', '>', '>']:
52         if not (self.is_numeric_type(left

```

check_arithmetic_operation:

Fungsi dipanggil saat ada operasi tambah, kurang, kali , bagi atau mod

check_relational_operation:

Fungsi ini dipanggil saat ada operasi =, , >, <, <>. Tujuannya selalu mengembalikan tipe boolean (benar / salah)

```

1 def get_result_type(self, op, left_type, right_type, line=None, column=None):
2     """Dapatkan tipe hasil dari operasi biner, menggabungkan seluruh pemeriksaan tipe operasi."""
3     # Operator aritmetika
4     if op in ['+', '-', '*', '/', 'bagi', 'mod']:
5         return self.check_arithmetic_operation(op, left_type, right_type, line, column)
6
7     # Operator relasional
8     if op in ['=', '<', '<', '>', '>']:
9         return self.check_relational_operation(op, left_type, right_type, line, column)
10
11    # Operator logika
12    if op in ['dan', 'dan', 'atau', 'atau']:
13        return self.check_logical_operation(op, left_type, right_type, line, column)
14
15    raise InvalidOperationError(op, self.get_type_name(left_type),
16                               self.get_type_name(right_type), line, column)

```

Bertugas sebagai dispatcher untuk menentukan tipe data hasil dari binary operation. Fungsi ini mendeteksi jenis operator yang digunakan (aritmatika?, relational? , ato logika), lalu dipper ke tugas validasi ke fungsi spesifik yang sesuai seperti check_arithmetic, check_relational, atau check_logical). Jika operator ga recognize, fungsi ini akan langsung throw error InvalidOperationError.

4.7 Errors

Kelas ini digunakan untuk mengecek error

| Class | Penjelasan Singkat |
|---|---|
| <pre> 1 class SemanticError(Exception): 2 # Kelas dasar untuk semua error semantik 3 def __init__(self, message, line=None, column=None): 4 self.message = message 5 self.line = line 6 self.column = column 7 super().__init__(self.format_message()) 8 9 def format_message(self): 10 if self.line and self.column: 11 return f"Semantic error at line {self.line}, column {self.column}: {self.message}" 12 elif self.line: 13 return f"Semantic error at line {self.line}: {self.message}" 14 return f"Semantic error: {self.message}" 15 16 class UndeclaredIdentifierError(SemanticError): 17 # Dilempar ketika sebuah identifier diminta tetapi belum dideklarasikan 18 def __init__(self, identifier, line=None, column=None): 19 message = f"Undeclared identifier '{identifier}'" 20 super().__init__(message, line, column) 21 22 class RedeclaredIdentifierError(SemanticError): 23 # Raised when an Identifier is declared multiple times in the same scope 24 def __init__(self, identifier, line=None, column=None): 25 message = f"Identifier '{identifier}' is already declared in this scope" 26 super().__init__(message, line, column) 27 28 class TypeMismatchError(SemanticError): 29 # Dilempar ketika tipe tidak cocok dalam operasi atau penugasan 30 def __init__(self, expected, actual, operation=None, line=None, column=None): 31 if operation: 32 message = f"Type mismatch in {operation}: expected {expected}, got {actual}" 33 else: 34 message = f"Type mismatch: expected {expected}, got {actual}" 35 super().__init__(message, line, column) 36 37 class InvalidOperationError(SemanticError): 38 # Dilempar ketika sebuah operasi tidak valid untuk tipe yang diberikan 39 def __init__(self, operation, type1, type2=None, line=None, column=None): 40 if type2: 41 message = f"Invalid operation '{operation}' for types {type1} and {type2}" 42 else: 43 message = f"Invalid operation '{operation}' for type {type1}" 44 super().__init__(message, line, column) 45 46 class InvalidArrayIndexError(SemanticError): 47 # Dilempar ketika indeks array bukan integer 48 def __init__(self, actual_type, line=None, column=None): 49 message = f"Array index must be integer, got {actual_type}" 50 super().__init__(message, line, column) 51 52 class NotAnArrayError(SemanticError): 53 # Dilempar ketika mencoba mengindeks variabel yang bukan array 54 def __init__(self, identifier, actual_type, line=None, column=None): 55 message = f"'{identifier}' is not an array (type: {actual_type})" 56 super().__init__(message, line, column) 57 58 class NotAProcedureError(SemanticError): 59 # Dilempar ketika mencoba memanggil identifier yang bukan prosedur 60 def __init__(self, identifier, line=None, column=None): 61 message = f"'{identifier}' is not a procedure" 62 super().__init__(message, line, column) 63 64 class ArgumentTypeError(SemanticError): 65 # Dilempar ketika pemanggilan fungsi/prosedur memiliki jumlah argument yang salah 66 def __init__(self, parameter_name, expected, actual, line=None, column=None): 67 message = f"Argument '{parameter_name}' expects {expected} arguments, got {actual}" 68 super().__init__(message, line, column) 69 70 class InvalidAssignmentError(SemanticError): 71 # Dilempar ketika variabel ditetapkan nilai ke konstanta atau entities yang tidak dapat diubah 72 def __init__(self, identifier, reason=None, line=None, column=None): 73 if reason: 74 message = f"Cannot assign to '{identifier}': {reason}" 75 else: 76 message = f"Cannot assign to '{identifier}'" 77 super().__init__(message, line, column) 78 79 class ReturnTypeMismatchError(SemanticError): 80 # Dilempar ketika tipe hasil fungsi tidak cocok dengan deklarasinya 81 def __init__(self, function_name, expected, actual, line=None, column=None): 82 message = f"Function '{function_name}' expects return type {expected}, got {actual}" 83 super().__init__(message, line, column) 84 85 class MissingReturnError(SemanticError): 86 # Dilempar ketika fungsi tidak memiliki penugasan nilai ke namanya (returnimplisit Pascal) 87 def __init__(self, function_name, line=None, column=None): 88 message = f"Function '{function_name}' must assign a value to its name" 89 super().__init__(message, line, column) 90 91 class ReturnNoneError(SemanticError): 92 # Dilempar ketika fungsi mengembalikan nilai None 93 def __init__(self, function_name, line=None, column=None): 94 message = f"Function '{function_name}' returned None" 95 super().__init__(message, line, column) 96 97 class InvalidIdentifierError(SemanticError): 98 # Dilempar ketika identifier diminta tetapi belum dideklarasikan 99 def __init__(self, identifier, line=None, column=None): 100 message = f"Invalid identifier '{identifier}'" 101 super().__init__(message, line, column) 102 </pre> | <p>SemanticError: Base class semua error semantik</p> <p>UndeclaredIdentifierError: pakai variabel x tapi blom di deklarasikan</p> <p>RedeclaredIdentifierError: deklaras xi , trus di bawah deklaras lagi di scope yang sama</p> <p>TypeMismatchError: Operasi matematika tapi ga nyambung ama tipenya malah string</p> <p>InvalidAssignmentError: Mengubah nilai konstanta atau variabel yang gabole diubah manually</p> <p>InvalidOperationError: Operasi tidak masuk akal , (“theo” - “learning”)</p> <p>NotAnArrayError: coba meng-indeks variabel x, tapi x itu integer</p> <p>InvalidArrayIndexError: akses array pakai tipe data aneh, contoh : myarr[“baby”]</p> <p>NotAFunctionError/NotAProcedureError: variabel x itu integer , tapi panggil x()</p> <p>ArgumentCountError: Fungsi butuh 2 argumen, malah masukin 1</p> <p>ReturnTypeMismatchError: Bilangnya return Integer, tapi didalam return “theo”</p> <p>MissingReturnError: Fungsi kelar tapi lupa return</p> |

4.8 Semantic Analyzer

Kelas ini babi haram

| Class | Penjelasan Singkat |
|---|---|
| <pre> 1 class SemanticAnalyzer: 2 3 def __init__(self): 4 self.symbolTable = SymbolTable() 5 self.type_checker = TypeChecker() 6 self.errors = [] 7 self.current_function = None # Lacak fungsi saat ini untuk pemeriksaan tipe kembalian 8 9 # Prosedur dan fungsi bawaan (built-in) 10 self.builtins = [11 'writeln', 'write', 'readln', 'read', 12 'writeln', 'write', 13] 14 15 self.string_handler = None </pre> | <p>Kelas SemanticAnalyzer ini bertujuan untuk mengemas (enkapsulasi) fungsi-fungsi yang terlibat dalam analisis semantik Pascal-S.</p> |
| <pre> 1 def analyze(self, ast_root): 2 try: 3 self.visit(ast_root) 4 if self.errors: 5 return False, self.errors 6 return True, [] 7 except SemanticError as e: 8 self.errors.append(e) 9 return False, self.errors 10 11 def report_error(self, error): 12 self.errors.append(error) 13 14 def visit(self, node): 15 method_name = f'visit_{node.__class__.__name__}' 16 visitor = getattr(self, method_name, self.generic_visit) 17 return visitor(node) 18 19 def generic_visit(self, node): 20 for child in getattr(node, 'children', []): 21 if child: 22 self.visit(child) 23 </pre> | <p>__init__(): Inisialisasi analyzer dengan symbol table, type checker, list error, dan set builtin functions</p> <p>analyze(ast_root): Entry point analisis semantik, menjalankan visitor dan mengembalikan status (sukses/gagal) + list error</p> <p>report_error(error): Menambahkan error ke list errors</p> <p>visit(node): Dispatcher pattern untuk memanggil visitor method sesuai tipe node (misal: visit_ProgramNode)</p> <p>generic_visit(node): Fallback visitor yang mengunjungi semua children node secara rekursif</p> |

```
1 def visit_ProgramNode(self, node):
2     # Tambahkan nama program ke symbol table
3     if node.name:
4         prog_idx = self.symbol_table.add_program_name(node.name)
5         # Dekorasi node
6         node.attr['tab_index'] = prog_idx
7         node.attr['type'] = TypeKind.NOTYPE
8         node.attr['lev'] = 0
9
10    # Kunjungi bagian deklarasi
11    if node.declarations:
12        for decl in node.declarations:
13            self.visit(decl)
14
15    # Kunjungi blok utama ~ enter scope untuk compound statement
16    if node.block:
17        block_idx = self.symbol_table.enter_scope()
18        # Dekorasi block node
19        node.block.attr['block_index'] = block_idx
20        node.block.attr['lev'] = self.symbol_table.current_level
21
22        self.visit(node.block)
23        self.symbol_table.exit_scope()
24
25    def visit_BlockNode(self, node):
26        """Kunjungi blok (compound statement)"""
27        for statement in node.statements:
28            if statement:
29                self.visit(statement)
30
31    def visit_DeclarationsNode(self, node):
32        """Kunjungi node deklarasi"""
33        for decl in node.declarations:
34            if decl:
35                self.visit(decl)
```

visit_ProgramNode(node): Proses node program utama, tambah nama program ke symbol table, kunjungi deklarasi & blok utama

visit_BlockNode(node): Kunjungi compound statement (blok begin..end), iterasi semua statements di dalamnya

visit_DeclarationsNode(node): Kunjungi bagian deklarasi (konstanta, variabel, tipe, prosedur/fungsi)

```

1  def visit_ConstDeclNode(self, node):
2      """Kunjungi deklarasi konstanta"""
3
4      # Periksa apakah sudah di deklarasikan di scope saat ini
5      existing = self.symbol_table.lookup_current_scope(node.name)
6      if existing:
7          self.report_error(RedeclaredIdentifierError(node.name))
8          return
9
10     # Tentukan tipe dari ekspresi nilai konstanta
11     type_kind = TypeKind.NOTYPE
12     if node.constant:
13         # Kunjungi ekspresi untuk memperoleh tipenya
14         type_kind = self.visit(node.constant)
15
16     # Tambahkan ke tabel simbol
17     self.symbol_table.add_constant(node.name, type_kind, None)
18
19     def visit_VarDeclNode(self, node):
20         existing = self.symbol_table.lookup_current_scope(node.name)
21         if existing:
22             self.report_error(RedeclaredIdentifierError(node.name))
23             return
24
25     # Handle variable of a user-defined RECORD type
26     if isinstance(node.vartype, RecordTypeNode):
27         # cek tipe record di symbol table jika ada type_name
28         type_name = getattr(node.vartype, 'name', None)
29         type_kind = TypeKind.RECORD
30
31         var_idx = self.symbol_table.add_variable(node.name, type_kind)
32         node.attr['tab_index'] = var_idx
33         node.attr['lev'] = self.symbol_table.current_level
34
35         # isi Fields
36         fields = {}
37         if hasattr(node.vartype, 'fields'):
38             for f in node.vartype.fields:
39                 # gunakan get type kind topo jika f.type adalah user-defined record, ambil fields
40                 ft = self._get_type_kind(f.type)
41                 # Jika user-defined record, ambil fields dari symbol table
42                 if ft == TypeKind.RECORD:
43                     rec_symbol = self.symbol_table.lookup(f.type._str_)
44                     if rec_symbol and 'fields' in rec_symbol:
45                         fields[f.name] = rec_symbol['fields']
46                     else:
47                         fields[f.name] = TypeKind.NOTYPE
48                 else:
49                     fields[f.name] = ft
50             node.attr['fields'] = fields
51             self.symbol_table.tab[var_idx]['fields'] = fields
52             return
53
54     # Handle array
55     if getattr(node.vartype, '__class__', None).__name__ == 'ArrayTypeNode':
56         type = self._get_type_kind(node.vartype.base_type)
57         low, high = 1, 10
58         if hasattr(node.vartype, 'bounds') and node.vartype.bounds:
59             b = node.vartype.bounds[0]
60             if isinstance(b, int) and b > 0 and b < 2:
61                 low = int(getattr(b, 'value', 1))
62                 high = int(getattr(b, 'value', 10))
63
64         var_idx = self.symbol_table.add_variable(node.name, TypeKind.ARRAY)
65         otab_idx = self.symbol_table.add_array_info(
66             xtype=Typekind.ARRAY, etype=base_type, low=low, high=high, elsize=1
67         )
68         self.symbol_table.tab[var_idx]['ref'] = otab_idx
69         node.attr['tab_index'] = var_idx
70         node.attr['type'] = Typekind.ARRAY
71         node.attr['lev'] = self.symbol_table.current_level
72         return
73
74     # Tipe sederhana
75     type_kind = self._get_type_kind(node.vartype)
76     var_idx = self.symbol_table.add_variable(node.name, type_kind)
77     node.attr['tab_index'] = var_idx
78     node.attr['type'] = type_kind
79     node.attr['lev'] = self.symbol_table.current_level
80
81     def visit_TypeDeclarationNode(self, node):
82         """Kunjungi deklarasi tipe"""
83         # Periksa apakah sudah di deklarasikan di scope saat ini
84         existing = self.symbol_table.lookup_current_scope(node.name)
85         if existing:
86             self.report_error(RedeclaredIdentifierError(node.name))
87             return
88
89         # Ambil tipe dari node.type
90         type_kind = self._resolve_type_node(node.type_node)
91
92         # Tambahkan ke tabel simbol
93         type_idx = self.symbol_table.add_type(node.name, type_kind)
94
95         # Jika ini RecordTypeNode, simpan field info ke symbol table entry
96         if isinstance(node.type_node, RecordTypeNode):
97             # Jika ada field info, tambahkan ke entry type_node.attr['fields']
98             field_info = node.type_node.attr.get('fields', {})
99             if type_idx is not None and 0 <= type_idx < len(self.symbol_table.tab):
100                 self.symbol_table.tab[type_idx]['fields'] = field_info

```

visit_ConstDeclNode(node): Validasi deklarasi konstanta: cek redeclaration, tentukan tipe dari nilai, tambah ke symbol table

visit_VarDeclNode(node): Validasi deklarasi variabel: cek redeclaration, handle tipe sederhana/array/record, tambah ke symbol table & ATAB

visit_TypeDeclarationNode(node): Validasi deklarasi tipe user-defined: cek redeclaration, resolve tipe (record/array/range), tambah ke symbol table

```

1 def visit_ProcedureDeclNode(self, node):
2     """Kunjungi deklarasi prosedur"""
3     # Periksa apakah sudah dideklarasikan di scope saat ini
4     existing = self.symbol_table.lookup_current_scope(node.name)
5     if existing:
6         self.report_error(ReddeclaredIdentifierError(node.name))
7         return
8
9     # Tambahkan prosedur ke tabel simbol
10    proc_idx = self.symbol_table.add_procedure(node.name, ObjKind.PROCEDURE)
11
12    # Masuk ke scope prosedur
13    block_idx = self.symbol_table.enter_scope()
14
15    # Kaitkan entri prosedur ke bloknya
16    if proc_idx >= 0 and proc_idx < len(self.symbol_table.tab):
17        self.symbol_table.tab[proc_idx]['ref'] = block_idx
18
19    # Tambahkan parameter
20    if node.params:
21        for param in node.params:
22            self.visit(param)
23
24    # Kunjungi badan prosedur
25    if node.block:
26        self.visit(node.block)
27
28    # Keluar dari scope prosedur
29    self.symbol_table.exit_scope()
30
31 def visit_FunctionDeclNode(self, node):
32     """Kunjungi deklarasi fungsi"""
33     # Periksa apakah sudah dideklarasikan di scope saat ini
34     existing = self.symbol_table.lookup_current_scope(node.name)
35     if existing:
36         self.report_error(ReddeclaredIdentifierError(node.name))
37         return
38
39     # Ambil tipe kembalian
40     return_type = self._get_type_kind(node.return_type)
41
42     # Tambahkan fungsi ke tabel simbol
43     func_idx = self.symbol_table.add_procedure(node.name, ObjKind.FUNCTION)
44
45     # Perbarui entri fungsi dengan tipe kembalian
46     self.symbol_table.tab[func_idx]['type'] = return_type
47
48     # Lacak fungsi saat ini untuk pemeriksaan return
49     prev_function = self.current_function
50     self.current_function = (node.name, return_type)
51
52     # Masuk ke scope fungsi
53     block_idx = self.symbol_table.enter_scope()
54
55     # Kaitkan entri fungsi ke bloknya
56     if func_idx >= 0 and func_idx < len(self.symbol_table.tab):
57         self.symbol_table.tab[func_idx]['ref'] = block_idx
58
59     # Tambahkan parameter
60     if node.params:
61         for param in node.params:
62             self.visit(param)
63
64     # Kunjungi badan fungsi
65     if node.block:
66         self.visit(node.block)
67
68     # Keluar dari scope fungsi
69     self.symbol_table.exit_scope()
70
71     # Kembalikan fungsi sebelumnya
72     self.current_function = prev_function
73
74 def visit_ParamNode(self, node):
75     """Kunjungi node parameter"""
76     type_kind = self._get_type_kind(node.type_node)
77     is_var_flag = getattr(node, 'is_var', False)
78
79     for name in node.names:
80         self.symbol_table.add_parameter(name, type_kind, is_var_flag)

```

visit_ProcedureDeclNode(node):

Proses deklarasi prosedur: cek redeclaration, enter scope baru, tambah parameter, kunjungi body, exit scope

visit_FunctionDeclNode(node):

Proses deklarasi fungsi: sama seperti prosedur tapi dengan return type, track current function untuk validasi return

visit_ParamNode(node):

Tambahkan parameter (formal parameter) ke symbol table dengan tipe & flag var/value parameter

```

1
2     def visit_AssignNode(self, node):
3         """Kunjungi pernyataan assignment"""
4         target_type = self.visit(node.target)
5         value_type = self.visit(node.value)
6
7         # cek kesesuaian tipe
8         try:
9             self.type_checker.check_assignment(target_type, value_type)
10        except SemanticError as e:
11            self.report_error(e)
12
13        node.attr['type'] = TypeKind.NOTYPE
14
15
16    def visit_IfNode(self, node):
17        """Kunjungi pernyataan if"""
18        # Periksa kondisi bertipe boolean
19        condition_type = self.visit(node.condition)
20        try:
21            self.type_checker.check_condition(condition_type, "if statement")
22        except SemanticError as e:
23            self.report_error(e)
24
25        # Kunjungi blok then
26        if node.then_block:
27            self.visit(node.then_block)
28
29        # Kunjungi blok else jika ada
30        if node.else_block:
31            self.visit(node.else_block)
32
33    def visit_WhileNode(self, node):
34        """Kunjungi pernyataan while"""
35        # Periksa kondisi bertipe boolean
36        condition_type = self.visit(node.condition)
37        try:
38            self.type_checker.check_condition(condition_type, "while statement")
39        except SemanticError as e:
40            self.report_error(e)
41
42        # Kunjungi body
43        if node.body:
44            self.visit(node.body)
45
46    def visit_RepeatNode(self, node):
47        """Kunjungi pernyataan repeat-until"""
48        # Kunjungi body terlebih dahulu
49        if node.body:
50            self.visit(node.body)
51
52        # Periksa kondisi bertipe boolean
53        condition_type = self.visit(node.condition)
54        try:
55            self.type_checker.check_condition(condition_type, "repeat-until statement")
56        except SemanticError as e:
57            self.report_error(e)
58
59    def visit_ForNode(self, node):
60        """Kunjungi pernyataan for"""
61        # Ambil type variabel loop
62        var_symbol = self.symbol_table.lookup(node.var_node.name)
63        if not var_symbol:
64            self.report_error(UndeclaredIdentifierError(node.var_node.name))
65        return
66
67        var_type = var_symbol['type']
68
69        # Ambil type ekspresi awal dan akhir
70        start_type = self.visit(node.start_expr)
71        end_type = self.visit(node.end_expr)
72
73        # Periksa batas kompatibel
74        try:
75            self.type_checker.check_for_loop_bounds(var_type, start_type, end_type)
76        except SemanticError as e:
77            self.report_error(e)
78
79        # Kunjungi body
80        if node.body:
81            self.visit(node.body)
82
83    def visit_ProcedureFunctionCallNode(self, node):
84        """Kunjungi pemanggilan prosedur/fungsi"""
85        # Periksa apakah ini built-in
86        call_name = node.name.lower() if isinstance(node.name, str) else str(node.name).lower()
87
88        if call_name in self.builtins:
89            # Built-in menerima argumen apa pun, lewati pemeriksaan rinci
90            if node.args:
91                for arg in node.args:
92                    self.visit(arg)
93
94            # Cek apakah built-in sudah ada di symbol table
95            symbol = self.symbol_table.lookup(call_name)
96            if not symbol:
97                # Tambahkan built-in ke symbol table saat pertama kali digunakan
98                tab_idx = self.symbol_table.add_builtin_procedure(call_name)
99            else:
100                # Cari tab_index untuk built-in yang sudah ada
101                tab_idx = None
102                for idx, entry in enumerate(self.symbol_table.tab):
103                    if entry.get('name') == call_name:
104                        tab_idx = idx
105                        break
106
107                node.attr['tab_index'] = tab_idx
108                node.attr['type'] = 'predefined'
109            return TypeKind.NOTYPE
110
111        # Cari entri prosedur/fungsi
112        symbol = self.symbol_table.lookup(node.name)
113        if not symbol:
114            self.report_error(UndeclaredIdentifierError(node.name))
115            return TypeKind.NOTYPE
116
117        # Periksa bahwa simbol dapat dipanggil
118        if symbol['obj'] not in [ObjKind.PROCEDURE, ObjKind.FUNCTION]:
119            self.report_error(NotAProcedureError(node.name))
120            return TypeKind.NOTYPE
121
122        # Ambil parameter yang diharapkan
123        expected_params = self.symbol_table.get_parameters(symbol)
124
125        # Periksa jumlah argumen
126        actual_count = len(node.args) if node.args else 0
127        expected_count = len(expected_params)
128
129        if actual_count != expected_count:
130            self.report_error(ArgumentCountError(expected_count, actual_count, node.name))
131
132        # Periksa tipe argumen
133        if node.args:
134            for i, arg in enumerate(zip(node.args, expected_params)):
135                org_type = self.visit(arg)
136                expected_type = expected_params[i]['type']
137
138                if not self.type_checker.is_compatible(expected_type, org_type):
139                    self.report_error(ArgumentTypeMismatchError(
140                        f"Argument {i+1}",
141                        self.type_checker.get_type_name(expected_type),
142                        self.type_checker.get_type_name(org_type)
143                    ))
144
145        # Kompolikan tipe untuk fungsi
146        if symbol['obj'] == ObjKind.FUNCTION:
147            return symbol['type']
148
149        return TypeKind.NOTYPE

```

visit_AssignNode(node): Validasi assignment: kunjungi target & value, cek kompatibilitas tipe menggunakan type checker

visit_IfNode(node): Validasi if statement: cek kondisi boolean, kunjungi then block & else block (jika ada)

visit_WhileNode(node): Validasi while loop: cek kondisi boolean, kunjungi body

visit_RepeatNode(node): Validasi repeat-until loop: kunjungi body dulu, lalu cek kondisi boolean

visit_ForNode(node): Validasi for loop: cek tipe variabel loop ordinal, validasi bounds (start/end) kompatibel, kunjungi body

visit_ProcedureFunctionCallNode(node): Validasi pemanggilan: handle builtin secara khusus, cek identifier ada & callable, validasi jumlah & tipe argumen

```

1  def visit_BinOpNode(self, node):
2      """Kunjungi operasi biner"""
3      left_type = self.visit(node.left)
4      right_type = self.visit(node.right)
5
6      try:
7          result_type = self.type_checker.get_result_type(
8              node.op, left_type, right_type)
9      except SemanticError as e:
10         self.report_error(e)
11         return TypeKind.NOTYPE
12
13     node.attrs['type'] = result_type
14     return result_type
15
16     except SemanticError as e:
17         self.report_error(e)
18         return TypeKind.NOTYPE
19
20     except SemanticError as e:
21         self.report_error(e)
22         return TypeKind.NOTYPE
23
24     except SemanticError as e:
25         self.report_error(e)
26         return TypeKind.NOTYPE
27
28     except SemanticError as e:
29         self.report_error(e)
30         return TypeKind.NOTYPE
31
32     except SemanticError as e:
33         self.report_error(e)
34         return TypeKind.NOTYPE
35
36     except SemanticError as e:
37         self.report_error(e)
38         return TypeKind.NOTYPE
39
40     except SemanticError as e:
41         self.report_error(e)
42         return TypeKind.NOTYPE
43
44     except SemanticError as e:
45         self.report_error(e)
46         return TypeKind.NOTYPE
47
48     except SemanticError as e:
49         self.report_error(e)
50         return TypeKind.NOTYPE
51
52     except SemanticError as e:
53         self.report_error(e)
54         return TypeKind.NOTYPE
55
56     except SemanticError as e:
57         self.report_error(e)
58         return TypeKind.NOTYPE
59
60     except SemanticError as e:
61         self.report_error(e)
62         return TypeKind.NOTYPE
63
64     except SemanticError as e:
65         self.report_error(e)
66         return TypeKind.NOTYPE
67
68     except SemanticError as e:
69         self.report_error(e)
70         return TypeKind.NOTYPE
71
72     except SemanticError as e:
73         self.report_error(e)
74
75     except SemanticError as e:
76         self.report_error(e)
77         return TypeKind.NOTYPE
78
79     except SemanticError as e:
80         self.report_error(e)
81
82     except SemanticError as e:
83         self.report_error(e)
84
85     except SemanticError as e:
86         self.report_error(e)
87
88     except SemanticError as e:
89         self.report_error(e)
90
91     except SemanticError as e:
92         self.report_error(e)
93
94     except SemanticError as e:
95         self.report_error(e)
96
97     except SemanticError as e:
98         self.report_error(e)
99
100    except SemanticError as e:
101        self.report_error(e)
102
103    except SemanticError as e:
104        self.report_error(e)
105
106    except SemanticError as e:
107        self.report_error(e)
108
109    except SemanticError as e:
110        self.report_error(e)
111
112    except SemanticError as e:
113        self.report_error(e)
114
115    except SemanticError as e:
116        self.report_error(e)
117
118    except SemanticError as e:
119        self.report_error(e)
120
121    except SemanticError as e:
122        self.report_error(e)
123
124    except SemanticError as e:
125        self.report_error(e)
126
127    except SemanticError as e:
128        self.report_error(e)
129
130    except SemanticError as e:
131        self.report_error(e)
132
133    except SemanticError as e:
134        self.report_error(e)
135
136    except SemanticError as e:
137        self.report_error(e)
138
139    except SemanticError as e:
140        self.report_error(e)
141
142    except SemanticError as e:
143        self.report_error(e)
144
145    except SemanticError as e:
146        self.report_error(e)
147
148    except SemanticError as e:
149        self.report_error(e)
150
151    except SemanticError as e:
152        self.report_error(e)
153
154    except SemanticError as e:
155        self.report_error(e)
156
157    except SemanticError as e:
158        self.report_error(e)
159
160    except SemanticError as e:
161        self.report_error(e)
162
```

visit_BinOpNode(node): Validasi operasi biner (+, -, *, /, and, or, dll): kunjungi operands, cek tipe valid untuk operator, return tipe hasil

visit_UnaryOpNode(node): Validasi operasi unary (+, -, not): kunjungi operand, cek tipe valid, return tipe hasil

visit_VarNode(node): Validasi referensi variabel: cek identifier dideklarasikan, dekorasi node dengan info dari symbol table, return tipe

visit_ArrayAccessNode(node): Validasi akses array arr[i]: cek variabel adalah array, validasi index integer, ambil tipe elemen dari ATAB

visit_RecordFieldNode(node): Validasi akses field record rec.field: kunjungi parent, cek parent adalah record, validasi field ada, return tipe field

visit_ArrayTypeNode(node): Resolve info tipe array (tipe elemen, bounds) untuk deklarasi variabel/tipe

visit_NumNode(node): Return tipe INTEGER atau REAL berdasarkan apakah ada titik desimal

visit_StringNode(node): Return tipe CHAR jika 1 karakter (dengan strip quote), atau STRING jika > 1 karakter

visit_BooleanNode(node): Return tipe BOOLEAN untuk literal true/false

```

1 def _get_type_kind(self, type_str):
2     """Konversikan string tipe menjadi konstanta TypeKind"""
3     type_map = {
4         'integer': TypeKind.INTEGER,
5         'real': TypeKind.REAL,
6         'boolean': TypeKind.BOOLEAN,
7         'char': TypeKind.CHAR,
8         'string': TypeKind.STRING,
9         'array': TypeKind.ARRAY,
10        'larik': TypeKind.ARRAY,
11        'record': TypeKind.RECORD,
12        'rekman': TypeKind.RECORD
13    }
14
15    if isinstance(type_str, str):
16        # Tipe Primitif/Keyword
17        mapped_type = type_map.get(type_str.lower())
18        if mapped_type is not None:
19            return mapped_type
20
21        # Tipe Didefinisikan Pengguna (Cari di Symbol Table)
22        symbol = self.symbol_table.lookup(type_str)
23        if symbol and symbol['obj'] == ObjKind.TYPE:
24            type_kind = symbol['type']
25            # ambil fields jika record
26            if type_kind == TypeKind.RECORD and 'fields' in symbol:
27                return {'kind': TypeKind.RECORD, 'fields': symbol['fields']}
28            return type_kind
29
30        # Tipe Record/Array (fallback)
31        return TypeKind.NOTYPE
32
33    return TypeKind.NOTYPE
34
35 def _resolve_type_node(self, type_node):
36
37     # RangeTypeNode -> integer subrange
38     if isinstance(type_node, RangeTypeNode):
39         low = type_node.lower.value
40         high = type_node.upper.value
41         type_node.attr['range'] = (low, high)
42         return TypeKind.INTEGER # Pascal subrange dianggap integer
43
44     # ArrayTypeNode
45     if isinstance(type_node, ArrayTypeNode):
46         # simpan batasan dan tipe elemen
47         base_type = self._get_type_kind(type_node.base_type)
48         bound = type_node.bounds[0]
49         low = bound[0].value
50         high = bound[1].value
51         type_node.attr['array_info'] = (base_type, low, high)
52         return TypeKind.ARRAY
53
54     # RecordTypeNode
55     if isinstance(type_node, RecordTypeNode):
56         fields = {}
57         for f in type_node.fields:
58             fields[f.name] = self._get_type_kind(f.type_)
59         type_node.attr['fields'] = fields
60         return TypeKind.RECORD
61
62     # primitive / identifier type
63     if isinstance(type_node, str):
64         return self._get_type_kind(type_node)
65
66     return TypeKind.NOTYPE
67

```

_get_type_kind(type_str): Konversi string nama tipe ke konstanta TypeKind, support tipe primitif & user-defined (lookup di symbol table)

_resolve_type_node(type_node):

Resolve AST type node (RangeTypeNode, ArrayTypeNode, RecordTypeNode) ke TypeKind dengan info tambahan (bounds, fields)

BAB V

PENGUJIAN TEST CASE

Untuk dokumentasi pengujian mandiri, dilakukan beberapa test case yang dapat dilihat pada tabel-tabel berikut:

(Dapat diakses pada repository, baik input maupun outputnya di “test/milestone-3/...”)

5.1 Test Case General

| No. | Input | Output |
|-----|---|---|
| 1. | <p>milestone-3/input/input-1.pas</p> <pre> ● ● ● program Hello; variabel a, b: integer; mulai a := 5; b := a + 10; writeln('Result = ', b); selesai. </pre> | <pre>===== SEMANTIC ANALYSIS + SYMBOL TABLE ===== TAB (Identifier Table) Idx Name Link Obj Type Ref Nrm Lev Adr -----+ ... (reserved words 0-28) 29 Hello 0 program 0 0 0 0 0 30 a 0 variable 0 0 0 0 0 31 b 30 variable 0 0 0 0 0 32 writeln 0 procedure 0 0 0 0 0 BTAB (Block Table) Idx Last Lpar Psze Vsze -----+ 0 31 0 0 2 1 0 0 0 0 ATAB (Array Table) (kosong karena tidak ada array) ===== DECORATED AST ===== ProgramNode(name: 'Hello') + tab_index:29, type:void, lev:0 - Declarations - VarDecl('a') + tab_index:30, type:integer, lev:0 - VarDecl('b') + tab_index:31, type:integer, lev:0 Block + block_index:1, lev:1 - Assign('a' := 5) + type:void - target 'a' + tab_index:30, type:integer, lev:0 - value 5 + type:integer - Assign('b' := expr) + type:void - target 'b' + tab_index:31, type:integer, lev:0 - BinOp '+' + type:integer - target 'a' + tab_index:30, type:integer, lev:0 - value 10 + type:integer ProcedureFunctionCall('writeln') + predefined, tab_index:32 String ('Result = ') + type:char - target 'b' + tab_index:31, type:integer, lev:0 </pre> |
| 2. | <p>milestone-3/input/input-2.pas</p> <pre> ● ● ● program Hello2; variabel a, b, i: integer; arr: larik[1 .. 5] dari integer; mulai a := 5; b := a + 10; writeln('Result = ', b); untuk i := 1 ke 5 lakukan mulai arr[i] := i + 1; writeln(arr[i]); selesai; selesai. </pre> | <pre>===== SEMANTIC ANALYSIS + SYMBOL TABLE ===== TAB (Identifier Table) Idx Name Link Obj Type Ref Nrm Lev Adr -----+ ... (reserved words 0-28) 29 Hello2 0 program 0 0 1 0 0 30 a 0 variable 1 0 1 0 0 31 b 30 variable 1 0 1 0 1 32 i 31 variable 1 0 1 0 2 33 arr 32 variable 5 0 1 0 3 34 writeln 0 procedure 0 0 1 0 0 35 writeln 0 procedure 0 0 1 0 0 BTAB (Block Table) Idx Last Lpar Psze Vsze -----+ 0 33 0 0 4 1 0 0 0 0 ATAB (Array Table) Idx Xtyp Etyp Eref Low High Elsz Size -----+ 1 5 1 0 1 5 1 5 </pre> |

| | | <pre>===== DECORATED AST ===== ProgramNode(name: 'Hello2') → tab_index:29, type:void, lev:0 └ Declarations └ VarDecl('a') → tab_index:30, type:integer, lev:0 └ VarDecl('b') → tab_index:31, type:integer, lev:0 └ VarDecl('i') → tab_index:32, type:integer, lev:0 └ VarDecl('arr') → tab_index:33, type:S, lev:0 └ ArrayType(base_type:integer, bounds=[(value 1, value 5)]) Block → block_index:1, lev:1 └ Assign('a' := 5) → type:void └ target 'a' → tab_index:30, type:integer, lev:0 └ value 5 → type:integer └ Assign('b' := expr) → type:void └ target 'b' → tab_index:31, type:integer, lev:0 └ BinOp('+') → type:integer └ target 'a' → tab_index:30, type:integer, lev:0 └ value 10 → type:integer └ ProcedureFunctionCall('writeln') → predefined, tab_index:34 └ String ('Result = ') → type:char └ target 'b' → tab_index:31, type:integer, lev:0 └ For Loop └ target 'i' └ value 1 → type:integer └ value 5 → type:integer └ Block └ Assign('target' := expr) → type:void └ Array Variable(array=arr, index=index) └ target 'arr' └ target 'i' → tab_index:32, type:integer, lev:0 └ BinOp('+') → type:integer └ target 'i' → tab_index:32, type:integer, lev:0 └ value 1 → type:integer └ ProcedureFunctionCall('writeln') → predefined, tab_index:35 └ Array Variable(array=arr, index=index) └ target 'arr' └ target 'i' → tab_index:32, type:integer, lev:0</pre> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|-------------------------------|--|-----------|-------|------|------|------|-----|-----|-----|-----|-----|------------------------|--|--|--|--|--|--|--|----|-------------|---|---------|---|---|---|---|---|----|---|---|----------|---|---|---|---|---|----|---|----|----------|---|---|---|---|---|----|---------|---|-----------|---|---|---|---|---|-----|------|------|-------|-------|---|----|---|---|---|---|---|---|---|---|-----|------|------|------|-----|------|------|------|---|---|---|---|---|---|---|---|-----|------|------|-----|------|-----|-----|-----|-----|-----|------------------------|--|--|--|--|--|--|--|----|-------------|---|---------|---|---|---|---|---|----|---|---|----------|---|---|---|---|---|----|---|----|----------|---|---|---|---|---|----|---------|---|-----------|---|---|---|---|---|-----|------|------|-------|-------|---|----|---|---|---|---|---|---|---|---|-----|------|------|------|-----|------|------|------|---|---|---|---|---|---|---|---|
| 3. | milestone-3/input/input-3.pas | <p>● ● ●</p> <pre>program RangesLoops; variabel i: integer; a: larik[0 .. 3] dari integer; mulai a[0] := 1; untuk i := 0 ke 3 lakukan a[i] := i; untuk i := 3 turun_ke 0 lakukan a[i] := a[i] + 1; jika a[1] < a[2] maka a[1] := a[2]; selama i > 0 lakukan i := i - 1; writeln('ok', a[1]); selesai.</pre> <table border="1"> <thead> <tr> <th>Idx</th><th>Name</th><th>Link</th><th>Obj</th><th>Type</th><th>Ref</th><th>Nrm</th><th>Lev</th><th>Adr</th></tr> </thead> <tbody> <tr><td>...</td><td>(reserved words 0--28)</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td>29</td><td>RangesLoops</td><td>0</td><td>program</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>30</td><td>i</td><td>0</td><td>variable</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>31</td><td>a</td><td>30</td><td>variable</td><td>5</td><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>32</td><td>writeln</td><td>0</td><td>procedure</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr> </tbody> </table> <table border="1"> <thead> <tr> <th>Idx</th><th>Last</th><th>Lpar</th><th>Psize</th><th>Vsize</th></tr> </thead> <tbody> <tr><td>0</td><td>31</td><td>0</td><td>0</td><td>2</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> </tbody> </table> <table border="1"> <thead> <tr> <th>Idx</th><th>Xtyp</th><th>Etyp</th><th>Eref</th><th>Low</th><th>High</th><th>Elsz</th><th>Size</th></tr> </thead> <tbody> <tr><td>1</td><td>5</td><td>1</td><td>0</td><td>0</td><td>3</td><td>1</td><td>4</td></tr> </tbody> </table> <pre>===== SEMANTIC ANALYSIS + SYMBOL TABLE =====</pre> <p>TAB (Identifier Table)</p> <table border="1"> <thead> <tr> <th>Idx</th><th>Name</th><th>Link</th><th>Obj</th><th>Type</th><th>Ref</th><th>Nrm</th><th>Lev</th><th>Adr</th></tr> </thead> <tbody> <tr><td>...</td><td>(reserved words 0--28)</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td>29</td><td>RangesLoops</td><td>0</td><td>program</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>30</td><td>i</td><td>0</td><td>variable</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>31</td><td>a</td><td>30</td><td>variable</td><td>5</td><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>32</td><td>writeln</td><td>0</td><td>procedure</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr> </tbody> </table> <p>BTAB (Block Table)</p> <table border="1"> <thead> <tr> <th>Idx</th><th>Last</th><th>Lpar</th><th>Psize</th><th>Vsize</th></tr> </thead> <tbody> <tr><td>0</td><td>31</td><td>0</td><td>0</td><td>2</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> </tbody> </table> <p>ATAB (Array Table)</p> <table border="1"> <thead> <tr> <th>Idx</th><th>Xtyp</th><th>Etyp</th><th>Eref</th><th>Low</th><th>High</th><th>Elsz</th><th>Size</th></tr> </thead> <tbody> <tr><td>1</td><td>5</td><td>1</td><td>0</td><td>0</td><td>3</td><td>1</td><td>4</td></tr> </tbody> </table> <pre>===== DECORATED AST =====</pre> <p>ProgramNode(name: 'RangesLoops') → tab_index:29, type:void, lev:0</p> <ul style="list-style-type: none"> Declarations <ul style="list-style-type: none"> VarDecl('i') → tab_index:30, type:integer, lev:0 VarDecl('a') → tab_index:31, type:S, lev:0 <ul style="list-style-type: none"> ArrayType(base_type:integer, bounds=[(value 0, value 3)]) Block → block_index:1, lev:1 <ul style="list-style-type: none"> Assign('target' := 1) → type:void <ul style="list-style-type: none"> target 'a' value 0 → type:integer value 1 → type:integer For Loop <ul style="list-style-type: none"> target 'i' value 0 → type:integer value 3 → type:integer Assign('target' := i) → type:void <ul style="list-style-type: none"> target 'a' target 'i' → tab_index:30, type:integer, lev:0 target 'i' → tab_index:30, type:integer, lev:0 For Loop <ul style="list-style-type: none"> target 'i' value 3 → type:integer value 0 → type:integer Assign('target' := expr) → type:void <ul style="list-style-type: none"> target 'a' target 'i' → tab_index:30, type:integer, lev:0 BinOp('+') → type:integer <ul style="list-style-type: none"> target 'a' target 'i' → tab_index:30, type:integer, lev:0 value 1 → type:integer | Idx | Name | Link | Obj | Type | Ref | Nrm | Lev | Adr | ... | (reserved words 0--28) | | | | | | | | 29 | RangesLoops | 0 | program | 0 | 0 | 1 | 0 | 0 | 30 | i | 0 | variable | 1 | 0 | 1 | 0 | 0 | 31 | a | 30 | variable | 5 | 0 | 1 | 0 | 1 | 32 | writeln | 0 | procedure | 0 | 0 | 1 | 0 | 0 | Idx | Last | Lpar | Psize | Vsize | 0 | 31 | 0 | 0 | 2 | 1 | 0 | 0 | 0 | 0 | Idx | Xtyp | Etyp | Eref | Low | High | Elsz | Size | 1 | 5 | 1 | 0 | 0 | 3 | 1 | 4 | Idx | Name | Link | Obj | Type | Ref | Nrm | Lev | Adr | ... | (reserved words 0--28) | | | | | | | | 29 | RangesLoops | 0 | program | 0 | 0 | 1 | 0 | 0 | 30 | i | 0 | variable | 1 | 0 | 1 | 0 | 0 | 31 | a | 30 | variable | 5 | 0 | 1 | 0 | 1 | 32 | writeln | 0 | procedure | 0 | 0 | 1 | 0 | 0 | Idx | Last | Lpar | Psize | Vsize | 0 | 31 | 0 | 0 | 2 | 1 | 0 | 0 | 0 | 0 | Idx | Xtyp | Etyp | Eref | Low | High | Elsz | Size | 1 | 5 | 1 | 0 | 0 | 3 | 1 | 4 |
| Idx | Name | Link | Obj | Type | Ref | Nrm | Lev | Adr | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | (reserved words 0--28) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 29 | RangesLoops | 0 | program | 0 | 0 | 1 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 30 | i | 0 | variable | 1 | 0 | 1 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | a | 30 | variable | 5 | 0 | 1 | 0 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 32 | writeln | 0 | procedure | 0 | 0 | 1 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Idx | Last | Lpar | Psize | Vsize | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 31 | 0 | 0 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 0 | 0 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Idx | Xtyp | Etyp | Eref | Low | High | Elsz | Size | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 5 | 1 | 0 | 0 | 3 | 1 | 4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Idx | Name | Link | Obj | Type | Ref | Nrm | Lev | Adr | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | (reserved words 0--28) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 29 | RangesLoops | 0 | program | 0 | 0 | 1 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 30 | i | 0 | variable | 1 | 0 | 1 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | a | 30 | variable | 5 | 0 | 1 | 0 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 32 | writeln | 0 | procedure | 0 | 0 | 1 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Idx | Last | Lpar | Psize | Vsize | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 31 | 0 | 0 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 0 | 0 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Idx | Xtyp | Etyp | Eref | Low | High | Elsz | Size | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 5 | 1 | 0 | 0 | 3 | 1 | 4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| | | |
|----|-------------------------------|--|
| | | <pre> - If Condition - BinOp '<' -> type:boolean - Array Variable(array=a, index=1) - target 'a' - value 1 -> type:integer - Array Variable(array=a, index=2) - target 'a' - value 2 -> type:integer - Assign('target' := expr) -> type:void - Array Variable(array=a, index=1) - target 'a' - value 1 -> type:integer - Array Variable(array=a, index=2) - target 'a' - value 2 -> type:integer - While Loop - BinOp '>' -> type:boolean - target 'i' -> tab_index:30, type:integer, lev:0 - value 0 -> type:integer - assign('i' := expr) -> type:void - target 'i' -> tab_index:30, type:integer, lev:0 - BinOp '-' -> type:integer - target 'i' -> tab_index:30, type:integer, lev:0 - value 1 -> type:integer - ProcedureFunctionCall('writeln') -> predefined, tab_index:32 - String ('ok') -> type:char - Array Variable(array=a, index=1) - target 'a' - value 1 -> type:integer </pre> |
| 4. | milestone-3/input/input-4.pas | <pre> ===== SEMANTIC ANALYSIS + SYMBOL TABLE ===== TAB (Identifier Table) Idx Name Link Obj Type Ref Nm Lev Adr ----- ... (reserved words 0-28) 29 Ops 0 program 0 0 1 0 0 30 x 0 variable 1 0 1 0 0 31 y 30 variable 1 0 1 0 1 32 r 31 variable 2 0 1 0 2 33 ch 32 variable 4 0 1 0 3 34 ok 33 variable 3 0 1 0 4 35 writeln 0 procedure 0 0 1 0 0 BTAB (Block Table) Idx Last Lpar Psze Vsze ----- 0 34 0 0 5 1 0 0 0 0 ATAB (Array Table) (Kosong karena tidak ada array) ===== DECORATED AST ===== ProgramNode(name: 'Ops') - tab_index:29, type:void, lev:0 - Declarations - VarDecl('x') - tab_index:30, type:integer, lev:0 - VarDecl('y') - tab_index:31, type:integer, lev:0 - VarDecl('r') - tab_index:32, type:real, lev:0 - VarDecl('ch') - tab_index:33, type:char, lev:0 - VarDecl('ok') - tab_index:34, type:boolean, lev:0 Block - block_index:1, lev:1 - Assign('x' := 10) -> type:void - target 'x' -> tab_index:30, type:integer, lev:0 - value 10 -> type:integer - Assign('y' := 3) -> type:void - target 'y' -> tab_index:31, type:integer, lev:0 - value 3 -> type:integer - Assign('r' := 12.5) -> type:void - target 'r' -> tab_index:32, type:real, lev:0 - value 12.5 -> type:real - Assign('ch' := 'Z') -> type:void - target 'ch' -> tab_index:33, type:char, lev:0 - String ('Z') -> type:char - Assign('ok' := expr) -> type:void - target 'ok' -> tab_index:34, type:boolean, lev:0 - BinOp '>' -> type:boolean - target 'x' -> tab_index:30, type:integer, lev:0 - target 'y' -> tab_index:31, type:integer, lev:0 - Assign('x' := expr) -> type:void - target 'x' -> tab_index:30, type:integer, lev:0 - BinOp '-' -> type:real - BinOp '+' -> type:integer - target 'x' -> tab_index:30, type:integer, lev:0 - target 'y' -> tab_index:31, type:integer, lev:0 - BinOp '*' -> type:real - BinOp '**' -> type:integer - value 2 -> type:integer - BinOp 'bagi' -> type:integer - target 'y' -> tab_index:31, type:integer, lev:0 - value 2 -> type:integer - value 1 -> type:integer - Assign('x' := expr) -> type:void - target 'x' -> tab_index:30, type:integer, lev:0 - BinOp 'mod' -> type:integer - target 'x' -> tab_index:30, type:integer, lev:0 - target 'y' -> tab_index:31, type:integer, lev:0 </pre> |

| | | <pre> If Condition ├ BinOp '<=' - type:boolean │ └ target 'x' - tab_index:30, type:integer, lev:0 │ └ target 'y' - tab_index:31, type:integer, lev:0 ├ Assign('x' := y) - type:void │ └ target 'x' - tab_index:30, type:integer, lev:0 │ └ target 'y' - tab_index:31, type:integer, lev:0 ├ Assign('x' := expr) - type:void │ └ target 'x' - tab_index:30, type:integer, lev:0 │ └ BinOp '+' - type:integer │ └ target 'y' - tab_index:31, type:integer, lev:0 │ └ value 1 - type:integer └ ProcedureFunctionCall('writeln') - predefined, tab_index:35 └ String ('done ') - type:char └ target 'x' - tab_index:30, type:integer, lev:0 </pre> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|-------------------------------|--|-----------|-------|------|------|------|------|-----|-----|-----|-----|-----------------------|--|--|--|--|--|--|--|----|--------------|---|---------|---|---|---|---|---|----|---|---|----------|---|---|---|---|------|----|---|----|----------|---|---|---|---|---|----|---|----|----------|---|---|---|---|---|----|----|----|----------|---|---|---|---|---|----|----|----|----------|---|---|---|---|---|----|-----|----|----------|---|---|---|---|---|----|--------|----|-----------|---|---|---|---|---|----|---|---|----------|---|---|---|---|---|----|--------|----|----------|---|---|---|---|---|----|---|---|----------|---|---|---|---|---|----|---------|---|-----------|---|---|---|---|---|-----|------|------|-------|-------|---|----|---|---|---|---|----|----|---|---|---|----|----|---|---|---|---|---|---|---|-----|------|------|------|-----|------|------|------|---|---|---|---|---|----|---|----|
| 5. | milestone-3/input/input-5.pas | <p>● ● ●</p> <pre> program DeclProcFunc; konstanta K = 10; variabel i: integer; r: real; ch: char; ok: boolean; arr: larik[1 .. 10] dari integer; prosedur IncVar(z: integer); mulai z := z + 1; selesai; fungsi AddOne(t: integer): integer; mulai AddOne := t + 1; selesai; mulai r := 3.14; ch := 'A'; ok := true dan tidak false; arr[1] := 2; jika (arr[1] >= K) maka arr[1] := AddOne(arr[1]) selain_itu IncVar(arr[1]); writeln('Value: ', arr[1], '.'); selesai. </pre> <p>===== SEMANTIC ANALYSIS + SYMBOL TABLE =====</p> <p>TAB (Identifier Table)</p> <table border="1"> <thead> <tr> <th>Idx</th><th>Name</th><th>Link</th><th>Obj</th><th>Type</th><th>Ref</th><th>Nrm</th><th>Lev</th><th>Adr</th></tr> </thead> <tbody> <tr><td>...</td><td>(reserved words 0-28)</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td>29</td><td>DeclProcFunc</td><td>0</td><td>program</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>30</td><td>K</td><td>0</td><td>constant</td><td>1</td><td>0</td><td>1</td><td>0</td><td>None</td></tr> <tr><td>31</td><td>i</td><td>30</td><td>variable</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>32</td><td>r</td><td>31</td><td>variable</td><td>2</td><td>0</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>33</td><td>ch</td><td>32</td><td>variable</td><td>4</td><td>0</td><td>1</td><td>0</td><td>2</td></tr> <tr><td>34</td><td>ok</td><td>33</td><td>variable</td><td>3</td><td>0</td><td>1</td><td>0</td><td>3</td></tr> <tr><td>35</td><td>arr</td><td>34</td><td>variable</td><td>5</td><td>0</td><td>1</td><td>0</td><td>4</td></tr> <tr><td>36</td><td>IncVar</td><td>35</td><td>procedure</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>37</td><td>z</td><td>0</td><td>variable</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>38</td><td>AddOne</td><td>36</td><td>function</td><td>1</td><td>2</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>39</td><td>t</td><td>0</td><td>variable</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>40</td><td>writeln</td><td>0</td><td>procedure</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr> </tbody> </table> <p>BTAB (Block Table)</p> <table border="1"> <thead> <tr> <th>Idx</th><th>Last</th><th>Lpar</th><th>Psize</th><th>Vsize</th></tr> </thead> <tbody> <tr><td>0</td><td>38</td><td>0</td><td>0</td><td>5</td></tr> <tr><td>1</td><td>37</td><td>37</td><td>1</td><td>0</td></tr> <tr><td>2</td><td>39</td><td>39</td><td>1</td><td>0</td></tr> <tr><td>3</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> </tbody> </table> <p>ATAB (Array Table)</p> <table border="1"> <thead> <tr> <th>Idx</th><th>Xtyp</th><th>Etyp</th><th>Eref</th><th>Low</th><th>High</th><th>Elsz</th><th>Size</th></tr> </thead> <tbody> <tr><td>1</td><td>5</td><td>1</td><td>0</td><td>1</td><td>10</td><td>1</td><td>10</td></tr> </tbody> </table> <p>===== DECORATED AST =====</p> <pre> ProgramNode(name: 'DeclProcFunc') -> tab_index:29, type:void, lev:0 Declarations └ ConstDecl(name='K') └ VarDecl('i') -> tab_index:31, type:integer, lev:0 └ VarDecl('r') -> tab_index:32, type:real, lev:0 └ VarDecl('ch') -> tab_index:33, type:char, lev:0 └ VarDecl('ok') -> tab_index:34, type:boolean, lev:0 └ VarDecl('arr') -> tab_index:35, type:5, lev:0 └ ArrayType(base_type:integer, bounds:[value 1, value 10]) └ ProcedureDecl(name='IncVar') └ Param(names='z', type='integer', is_var=False) └ Block └ Declarations └ Block └ Assign('z' := expr) -> type:void └ target 'z' -> tab_index:37, type:integer, lev:1 └ BinOp '+' -> type:integer └ target 'z' -> tab_index:37, type:integer, lev:1 └ value 1 -> type:integer └ FunctionDecl(name='AddOne') └ Param(names='t', type='integer', is_var=False) └ Block └ Declarations └ Block └ Assign('AddOne' := expr) -> type:void └ target 'AddOne' -> tab_index:38, type:integer, lev:0 └ BinOp '+' -> type:integer └ target 't' -> tab_index:39, type:integer, lev:1 └ value 1 -> type:integer </pre> | Idx | Name | Link | Obj | Type | Ref | Nrm | Lev | Adr | ... | (reserved words 0-28) | | | | | | | | 29 | DeclProcFunc | 0 | program | 0 | 0 | 1 | 0 | 0 | 30 | K | 0 | constant | 1 | 0 | 1 | 0 | None | 31 | i | 30 | variable | 1 | 0 | 1 | 0 | 0 | 32 | r | 31 | variable | 2 | 0 | 1 | 0 | 1 | 33 | ch | 32 | variable | 4 | 0 | 1 | 0 | 2 | 34 | ok | 33 | variable | 3 | 0 | 1 | 0 | 3 | 35 | arr | 34 | variable | 5 | 0 | 1 | 0 | 4 | 36 | IncVar | 35 | procedure | 0 | 1 | 1 | 0 | 0 | 37 | z | 0 | variable | 1 | 0 | 1 | 1 | 0 | 38 | AddOne | 36 | function | 1 | 2 | 1 | 0 | 0 | 39 | t | 0 | variable | 1 | 0 | 1 | 1 | 0 | 40 | writeln | 0 | procedure | 0 | 0 | 1 | 0 | 0 | Idx | Last | Lpar | Psize | Vsize | 0 | 38 | 0 | 0 | 5 | 1 | 37 | 37 | 1 | 0 | 2 | 39 | 39 | 1 | 0 | 3 | 0 | 0 | 0 | 0 | Idx | Xtyp | Etyp | Eref | Low | High | Elsz | Size | 1 | 5 | 1 | 0 | 1 | 10 | 1 | 10 |
| Idx | Name | Link | Obj | Type | Ref | Nrm | Lev | Adr | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | (reserved words 0-28) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 29 | DeclProcFunc | 0 | program | 0 | 0 | 1 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 30 | K | 0 | constant | 1 | 0 | 1 | 0 | None | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | i | 30 | variable | 1 | 0 | 1 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 32 | r | 31 | variable | 2 | 0 | 1 | 0 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 33 | ch | 32 | variable | 4 | 0 | 1 | 0 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 34 | ok | 33 | variable | 3 | 0 | 1 | 0 | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 35 | arr | 34 | variable | 5 | 0 | 1 | 0 | 4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 36 | IncVar | 35 | procedure | 0 | 1 | 1 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 37 | z | 0 | variable | 1 | 0 | 1 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 38 | AddOne | 36 | function | 1 | 2 | 1 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 39 | t | 0 | variable | 1 | 0 | 1 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 40 | writeln | 0 | procedure | 0 | 0 | 1 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Idx | Last | Lpar | Psize | Vsize | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 38 | 0 | 0 | 5 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 37 | 37 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | 39 | 39 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | 0 | 0 | 0 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Idx | Xtyp | Etyp | Eref | Low | High | Elsz | Size | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | 5 | 1 | 0 | 1 | 10 | 1 | 10 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| | | |
|--|--|--|
| | | <pre> └ Block + block_index:3, lev:1 └ Assign('r' := 3.14) + type:void └ target 'r' + tab_index:32, type:real, lev:0 └ value 3.14 + type:real └ Assign('ch' := 'A') + type:void └ target 'ch' + tab_index:33, type:char, lev:0 └ String ('A') + type:char └ Assign('ok' := True) + type:void └ target 'ok' + tab_index:34, type:boolean, lev:0 └ Boolean (True) └ Assign('target' := 2) + type:void └ target 'target' + tab_index:35, type:integer, lev:0 └ value 2 + type:integer └ If Condition └ BinOp '>' + type:boolean └ target 'arr' + tab_index:36, type:integer, lev:0 └ value 1 + type:integer └ Assign('target' := AddOne) + type:void └ target 'target' + tab_index:37, type:integer, lev:0 └ value 1 + type:integer └ ProcedureFunctionCall('AddOne') └ target 'arr' + tab_index:38, type:integer, lev:0 └ value 1 + type:integer └ ProcedureFunctionCall('IncVar') └ target 'arr' + tab_index:39, type:integer, lev:0 └ value 1 + type:integer └ ProcedureFunctionCall('writeln') + predefined, tab_index:40 └ target 'writeln' + tab_index:41, type:char └ value 1 + type:integer └ String ('Value: ') + type:char └ Array Variable(array=arr, index=1) └ target 'arr' + tab_index:42, type:integer, lev:0 └ value 1 + type:integer └ String ('..') + type:char </pre> |
|--|--|--|

5.2 Test Case untuk Milestone 3

| No. | Input | Output |
|-----|--|---|
| 1. | milestone-3/input/input-fail-1.pas <pre> ● ● ● program TestFailType; variabel angka: integer; status: boolean; mulai angka := 10; status := true; angka := status; jika angka + status > 10 mak a status := false; selesai. </pre> | <p>Semantic errors found:</p> <ul style="list-style-type: none"> - Semantic error: Type mismatch in assignment: expected integer, got boolean - Semantic error: Invalid operation '+' for types integer and boolean - Semantic error: Invalid operation '>' for types notype and integer - Semantic error: Type mismatch in if statement: expected boolean, got notype |
| 2. | milestone-3/input/input-fail-2.pas | <p>Semantic errors found:</p> <ul style="list-style-type: none"> - Semantic error: Undeclared identifier 'local_var' - Semantic error: Type mismatch in assignment: expected notype, got integer - Semantic error: Cannot access field 'x' of non-record type - Semantic error: Type mismatch in assignment: expected notype, got integer |

```

program TestFailScope;

variabel
    global_var: integer;

prosedur localScope;
variabel
    local_var: integer;
mulai
    local_var := 10;
selesai;

mulai
    global_var := 20;

    local_var := 5;

    global_var.x := 10;
selesai.

```

3. milestone-3/input/input-semantic-arrayrec.pas

```

program TestArrayRecord;

tipe
    Point = rekaman
        x, y: integer
    selesai;

variabel
    p: Point;
    matrix: larik[1 .. 5] dari integer;
    i, total: integer;

mulai
    p.x := 10;
    p.y := 20;
    total := p.x * p.y;

    untuk i := 1 ke 5 lakukan
        matrix[i] := i * 2;

    matrix[1] := matrix[1] + p.x;

    writeln(total);
selesai.

```

```

=====
 SEMANTIC ANALYSIS + SYMBOL TABLE =====

 TAB (Identifier Table)
 -----
 Idx Name Link Obj Type Ref Nm Lev Adr
 -----
 (reserved words 0-28)
 29 TestArrayRecord 0 program 0 0 1 0 0
 30 Point 0 type 5 0 1 0 0
 31 p 30 variable 5 0 1 0 0
 32 matrix 31 variable 5 0 1 0 1
 33 i 32 variable 1 0 1 0 2
 34 total 33 variable 1 0 1 0 3
 35 writeln 0 procedure 0 0 1 0 0

 BTAB (Block Table)
 -----
 Idx Last Lpar Psze Vsze
 -----
 0 34 0 0 4
 1 0 0 0 0

 ATAB (Array Table)
 -----
 Idx Xtyp Etyp Eref Low High Elsz Size
 -----
 1 5 1 0 1 5 1 5

=====
 DECORATED AST =====

ProgramNode(name: 'TestArrayRecord') - tab_index:29, type:void, lev:0
|- Declarations
  |- TypeDecl(name: Point)
    |- RecordField('x')
      |- RecDefField('x')
        |- RecDefField('y')
    |- VarDecl('p') - tab_index:31, type:(`kind': 6, 'fields': {'x': 1, 'y': 1}), lev:0
    |- VarDecl('matrix') - tab_index:32, type:5, lev:0
      |- ArrayType(base_type:integer, bounds:([value 1, value 5]))
    |- VarDecl('i') - tab_index:33, type:integer, lev:0
    |- VarDecl('total') - tab_index:34, type:integer, lev:0
  |- Block - block_index:1, lev:0
    |- Assign('x' := 10) - type:void
      |- RecDefField('x')
        |- target 'p' - tab_index:31, type:(`kind': 6, 'fields': {'x': 1, 'y': 1}), lev:0
        |- value 10 - type:integer
    |- Assign('y' := 20) - type:void
      |- RecDefField('y')
        |- target 'p' - tab_index:31, type:(`kind': 6, 'fields': {'x': 1, 'y': 1}), lev:0
        |- value 20 - type:integer
    |- Assign('total' := expr) - type:void
      |- target 'total' - tab_index:34, type:integer, lev:0
      |- BindExpr(expr)
        |- ConstField('1') - type:integer
          |- target 'p' - tab_index:31, type:(`kind': 6, 'fields': {'x': 1, 'y': 1}), lev:0
        |- RecDefField('y')
          |- target 'p' - tab_index:31, type:(`kind': 6, 'fields': {'x': 1, 'y': 1}), lev:0
    |- For Loop
      |- target 'i'
      |- value 1 - type:integer
      |- value 5 - type:integer
      |- Assign('target' := expr) - type:void
        |- Array Variable(ArrayMatrix, index:Index)
          |- target 'matrix'
            |- target 'i' - tab_index:33, type:integer, lev:0
            |- BindOp('*')
              |- target 'i' - tab_index:33, type:integer, lev:0
              |- target '1' - tab_index:33, type:integer, lev:0
            |- target 'matrix'
              |- target 'i' - tab_index:33, type:integer, lev:0
              |- RecDefField('x')
                |- target 'p' - tab_index:31, type:(`kind': 6, 'fields': {'x': 1, 'y': 1}), lev:0
      |- ProcedureFunctionCall('writeln') - predefined, tab_index:35
        |- target 'total' - tab_index:34, type:integer, lev:0

```

4.

milestone-3/input/input-semantic-basic.pas

```

● ● ●

program TestBasic;

variabel
  i, limit, sum: integer;
  flag: boolean;

mulai
  limit := 10;
  sum := 0;
  flag := true;

  untuk i := 1 ke limit lakukan
    sum := sum + i;

  jika sum > 50 maka
    flag := false
  selain_itu
    flag := true;

  selama limit > 0 lakukan
  mulai
    limit := limit - 1;
    writeln(limit);
  selesai;
selesai.

```

```

=====
 SEMANTIC ANALYSIS + SYMBOL TABLE =====

 TAB (Identifier Table)
 -----
 Idx Name Link Obj Type Ref Nrm Lev Adr
 ...
 (reserved words 0-28)
 29 TestBasic 0 program 0 0 1 0 0
 30 i 0 variable 1 0 1 0 0
 31 limit 30 variable 1 0 1 0 1
 32 sum 31 variable 1 0 1 0 2
 33 flag 32 variable 3 0 1 0 3
 34 writeln 0 procedure 0 0 1 0 0

 BTAB (Block Table)
 -----
 Idx Last Lpar Psze Vsze
 -----
 0 33 0 0 4
 1 0 0 0 0

 ATAB (Array Table)
 (kosong karena tidak ada array)

```

```

=====
 DECORATED AST =====

└ ProgramNode(name: 'TestBasic') → tab_index:29, type:void, lev:0
  └ Declarations
    └ VarDecl('i') → tab_index:30, type:integer, lev:0
    └ VarDecl('limit') → tab_index:31, type:integer, lev:0
    └ VarDecl('sum') → tab_index:32, type:integer, lev:0
    └ VarDecl('flag') → tab_index:33, type:boolean, lev:0
  └ Block + block_index:1, lev:1
    └ Assign('limit' := 10) → type:void
      └ target 'limit' → tab_index:31, type:integer, lev:0
      └ value 10 → type:integer
    └ Assign('sum' := 0) → type:void
      └ target 'sum' → tab_index:32, type:integer, lev:0
      └ value 0 → type:integer
    └ Assign('flag' := True) → type:void
      └ target 'flag' → tab_index:33, type:boolean, lev:0
      └ Boolean (True)
    └ For Loop
      └ target 'i'
      └ value 1 → type:integer
      └ target 'limit' → tab_index:31, type:integer, lev:0
      └ Assign('sum' := expr) → type:void
        └ target 'sum' → tab_index:32, type:integer, lev:0
        └ BinOp '+=' → type:integer
          └ target 'sum' → tab_index:32, type:integer, lev:0
          └ target 'i' → tab_index:30, type:integer, lev:0
    └ If Condition
      └ BinOp '>' → type:boolean
        └ target 'sum' → tab_index:32, type:integer, lev:0
        └ value 50 → type:integer
      └ Assign('flag' := False) → type:void
        └ target 'flag' → tab_index:33, type:boolean, lev:0
        └ Boolean (False)
      └ Assign('flag' := True) → type:void
        └ target 'flag' → tab_index:33, type:boolean, lev:0
        └ Boolean (True)
    └ While Loop
      └ BinOp '>' → type:boolean
        └ target 'limit' → tab_index:31, type:integer, lev:0
        └ value 0 → type:integer
      └ Block
        └ Assign('limit' := expr) → type:void
          └ target 'limit' → tab_index:31, type:integer, lev:0
          └ BinOp '-=' → type:integer
            └ target 'limit' → tab_index:31, type:integer, lev:0
            └ value 1 → type:integer
        └ ProcedureFunctionCall('writeln') → predefined, tab_index:34
          └ target 'limit' → tab_index:31, type:integer, lev:0

```

```

=====
 DECORATED AST =====

└ ProgramNode(name: 'TestBasic') → tab_index:29, type:void, lev:0
  └ Declarations
    └ VarDecl('global_X') → tab_index:30, type:variable, lev:0
    └ VarDecl('global_Y') → tab_index:31, type:variable, lev:0
    └ VarDecl('swap') → tab_index:32, type:procedure, lev:0
    └ VarDecl('a') → tab_index:33, type:variable, lev:0
    └ VarDecl('b') → tab_index:34, type:variable, lev:0
    └ VarDecl('temp') → tab_index:35, type:variable, lev:0
    └ VarDecl('add') → tab_index:36, type:function, lev:0
    └ VarDecl('a') → tab_index:37, type:variable, lev:0
  └ Block + block_index:1, lev:1
    └ Assign('global_X' := 30) → type:void
      └ target 'global_X' → tab_index:30, type:variable, lev:0
      └ value 30 → type:integer
    └ Assign('global_Y' := 31) → type:void
      └ target 'global_Y' → tab_index:31, type:variable, lev:0
      └ value 31 → type:integer
    └ Swap('global_X', 'global_Y') → type:void
      └ target 'swap' → tab_index:32, type:procedure, lev:0
      └ target 'global_X' → tab_index:30, type:variable, lev:0
      └ target 'global_Y' → tab_index:31, type:variable, lev:0
    └ Add('global_X', 'global_Y') → type:void
      └ target 'add' → tab_index:36, type:function, lev:0
      └ target 'global_X' → tab_index:30, type:variable, lev:0
      └ target 'global_Y' → tab_index:31, type:variable, lev:0
      └ target 'temp' → tab_index:35, type:variable, lev:0
      └ target 'temp' := 30 → type:void
        └ target 'temp' → tab_index:35, type:variable, lev:0
        └ value 30 → type:integer
    └ Print('global_X') → type:void
      └ target 'writeln' → predefined, tab_index:34
      └ target 'global_X' → tab_index:30, type:variable, lev:0

```

5.

milestone-3/input/input-semantic-proc.pas

```

=====
 SEMANTIC ANALYSIS + SYMBOL TABLE =====

 TAB (Identifier Table)
 -----
 Idx Name Link Obj Type Ref Nrm Lev Adr
 ...
 (reserved words 0-28)
 29 TestProc 0 program 0 0 1 0 0
 30 global_X 0 variable 1 0 1 0 0
 31 global_Y 30 variable 1 0 1 0 1
 32 swap 31 procedure 0 1 1 0 0
 33 a 0 variable 1 0 0 1 0
 34 b 33 variable 1 0 0 1 1
 35 temp 34 variable 1 0 1 1 0
 36 add 32 function 1 2 1 0 0
 37 a 0 variable 1 0 1 1 0
 38 b 37 variable 1 0 1 1 1

 BTAB (Block Table)
 -----
 Idx Last Lpar Psze Vsze
 -----
 0 36 0 0 2
 1 35 34 2 1
 2 38 38 2 0
 3 0 0 0 0

 ATAB (Array Table)
 (kosong karena tidak ada array)

```

```

● ● ●

program TestProc;
variabel
    global_x, global_y: integer;

prosedur swap(variabel a: integer; variabel b: integer);
variabel temp: integer;
mulai
    temp := a;
    a := b;
    b := temp;
selesai;

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

mulai
    global_x := 100;
    global_y := 200;

    swap(global_x, global_y);

    global_x := add(global_x, global_y);
selesai.

```

```

=====
 DECORATED AST =====

ProgramNode(name: 'TestProc') → tab_index:29, type:void, lev:0
└ Declarations
  └ VarDecl('global_x') → tab_index:30, type:integer, lev:0
  └ VarDecl('global_y') → tab_index:31, type:integer, lev:0
  └ ProcedureDecl(name='swap')
    └ Param(names='a', type='integer', is_var=True)
    └ Param(names='b', type='integer', is_var=True)
    └ Block
      └ Declarations
        └ VarDecl('temp') → tab_index:35, type:integer, lev:1
      └ Block
        └ Assign('temp' := a) → type:void
          └ target 'temp' → tab_index:35, type:integer, lev:1
          └ target 'a' → tab_index:33, type:integer, lev:1
        └ Assign('a' := b) → type:void
          └ target 'a' → tab_index:33, type:integer, lev:1
          └ target 'b' → tab_index:34, type:integer, lev:1
        └ Assign('b' := temp) → type:void
          └ target 'b' → tab_index:34, type:integer, lev:1
          └ target 'temp' → tab_index:35, type:integer, lev:1
    └ FunctionDecl(name='add')
      └ Param(names='a', type='integer', is_var=False)
      └ Param(names='b', type='integer', is_var=False)
      └ Block
        └ Declarations
        └ Block
          └ Assign('add' := expr) → type:void
            └ target 'add' → tab_index:36, type:integer, lev:0
            └ BinOp '+' → type:integer
            └ target 'a' → tab_index:33, type:integer, lev:1
            └ target 'b' → tab_index:34, type:integer, lev:1
    └ Block → block_index:3, lev:1
      └ Assign('global_x' := 100) → type:void
        └ target 'global_x' → tab_index:30, type:integer, lev:0
        └ value 100 → type:integer
      └ Assign('global_y' := 200) → type:void
        └ target 'global_y' → tab_index:31, type:integer, lev:0
        └ value 200 → type:integer
      └ ProcedureFunctionCall('swap')
        └ target 'global_x' → tab_index:30, type:integer, lev:0
        └ target 'global_y' → tab_index:31, type:integer, lev:0
      └ Assign('global_x' := add) → type:void
        └ target 'global_x' → tab_index:30, type:integer, lev:0
      └ ProcedureFunctionCall('add')
        └ target 'global_x' → tab_index:30, type:integer, lev:0
        └ target 'global_y' → tab_index:31, type:integer, lev:0

```

6. milestone-3/input/input-semantic-var reference.pas

```

● ● ●

program VarRef;
prosedur tukar(variabel x: integer; y: integer);
mulai
    x := y;
selesai;
mulai
    tukar(10, 20);
selesai.

```

```

=====
 SEMANTIC ANALYSIS + SYMBOL TABLE =====

TAB (Identifier Table)

Idx Name      Link Obj      Type Ref Nm Lev Adr
----- ...
... (reserved words 0-28)
29 VarRef    0 program 0  0   1  0   0
30 tukar     0 procedure 0  1   1  0   0
31 x         0 variable 1  0   0   0  1  0
32 y         31 variable 1  0   1   1  1  1

BTAB (Block Table)

Idx Last Lpar Psze Vsze
----- ...
0 30 0 0 0
1 32 32 2 0
2 0 0 0 0

ATAB (Array Table)
(kosong karena tidak ada array)

=====
 DECORATED AST =====

ProgramNode(name: 'VarRef') → tab_index:29, type:void, lev:0
└ Declarations
  └ ProcedureDecl(name='tukar')
    └ Param(names='x', type='integer', is_var=True)
    └ Param(names='y', type='integer', is_var=False)
    └ Block
      └ Declarations
        └ Block
          └ Assign('x' := y) → type:void
            └ target 'x' → tab_index:31, type:integer, lev:1
            └ target 'y' → tab_index:32, type:integer, lev:1
    └ Block → block_index:2, lev:1
      └ ProcedureFunctionCall('tukar')
        └ value 10 → type:integer
        └ value 20 → type:integer

```

7.

milestone-3/input/input-string.pas

```

    ● ○ ●

1 program contoh_string;
2 variabel
3     s1, s2 : String;
4 mulai
5     s1 := 'Informatika ';
6     s2 := 'ITB';
7     writeln(s1 + s2);
8 selesai.
9

```

```
===== SEMANTIC ANALYSIS + SYMBOL TABLE =====
```

TAB (Identifier Table)

| Idx | Name | Link | Obj | Type | Ref | Nrm | Lev | Adr |
|-----|-----------------------|------|-----|-----------|-----|-----|-----|-----|
| ... | (reserved words 0-28) | | | | | | | |
| 29 | contoh_string | 0 | | program | 0 | 0 | 1 | 0 |
| 30 | s1 | 0 | | variable | 7 | 0 | 1 | 0 |
| 31 | s2 | 30 | | variable | 7 | 0 | 1 | 0 |
| 32 | writeln | 0 | | procedure | 0 | 0 | 1 | 0 |

BTAB (Block Table)

| Idx | Last | Lpar | Pszs | Vsze |
|-----|------|------|------|------|
| 0 | 31 | 0 | 0 | 2 |
| 1 | 0 | 0 | 0 | 0 |

ATAB (Array Table)

(kosong karena tidak ada array)

```
===== DECORATED AST =====
```

```

└ ProgramNode(name: 'contoh_string') → tab_index:29, type:void, lev:0
  └ Declarations
    └ VarDecl('s1') → tab_index:30, type:string, lev:0
    └ VarDecl('s2') → tab_index:31, type:string, lev:0
  └ Block → block_index:1, lev:1
    └ Assign('s1' := 'Informatika ') → type:void
      └ target 's1' → tab_index:30, type:string, lev:0
      └ String ('Informatika ') → type:string
    └ Assign('s2' := 'ITB') → type:void
      └ target 's2' → tab_index:31, type:string, lev:0
      └ String ('ITB') → type:string
    └ ProcedureFunctionCall('writeln') → predefined, tab_index:32
      └ BinOp '+' → type:string
        └ target 's1' → tab_index:30, type:string, lev:0
        └ target 's2' → tab_index:31, type:string, lev:0

```

BAB VI

KESIMPULAN & SARAN

6.1 Kesimpulan

Implementasi Semantic Analysis untuk compiler Pascal-S pada Milestone 3 telah berhasil diselesaikan. Pada tahap ini, struktur program yang sebelumnya direpresentasikan sebagai parse tree dan AST telah dianalisis lebih lanjut untuk memastikan konsistensi makna dan kepatuhan terhadap aturan bahasa Pascal-S yang bersifat *strong static typing*. Proses analisis dilakukan dengan menggabungkan mekanisme Attributed Grammar, penyusunan dan manajemen Symbol Table yang mencakup tab, btab, dan atab, serta traversal AST secara top-down menggunakan semantic visitor. Pendekatan ini memungkinkan pemeriksaan menyeluruh terhadap deklarasi identifier, tipe data, scope blok bersarang, operasi aritmatika maupun logika, pemanggilan prosedur/fungsi, dan akses struktur komposit (array serta record).

Selain melakukan validasi, semantic analyzer juga menghasilkan *Decorated AST*, yaitu AST yang telah dianotasi dengan atribut semantik seperti tipe hasil ekspresi, indeks tabel simbol, dan informasi blok. Dengan demikian, milestone ini berhasil menyediakan pondasi semantik yang solid dan konsisten untuk tahapan berikutnya, yaitu Intermediate Code Generation.

6.2 Saran

Meskipun semantic analyzer telah berfungsi dengan baik untuk menangani aturan semantik Pascal-S, masih terdapat beberapa aspek yang dapat ditingkatkan. Pertama, mekanisme error reporting dapat diperluas untuk menampilkan pesan kesalahan yang lebih kontekstual dan rapih, termasuk pelacakan lokasi kesalahan yang lebih presisi dengan informasi line yang menyebabkan error. Selain itu, optimisasi traversal AST dan pengelolaan symbol table memungkinkan peningkatan efisiensi, terutama terhadap *edge cases* seperti tipe rekursif, pengecekan kompatibilitas operator yang lebih lengkap dan akurat, serta verifikasi parameter prosedur/fungsi yang lebih ketat juga.

REFERENSI

- [1] Edunex ITB. (n.d.). *Compiler Parsing dan Analisis Semantik*. ITB Lecture Module.

Accessed: November 24, 2025. [Online]. Available: <http://edunex.itb.ac.id>

[2] N. Wirth, PASCAL-S: A Subset and its implementation. Accessed: November 12, 2025. [Online]. Available:

<http://pascal.hansotten.com/uploads/pascals/PASCAL-S%20A%20subset%20and%20its%20Implementation%20012.pdf>

[3] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 3rd ed. Boston, MA: Addison-Wesley, 2006.

[4] ULiège, *Semantic Analysis* Available:

<https://people.montefiore.ulg.be/geurts/Cours/compil/2015/04-semantic-2015-2016.pdf>

[5] “Pascal for small machines” Available: <http://pascal.hansotten.com/niklaus-wirth/pascal-s/>

LAMPIRAN

Link Repository : [Link Repository \[GitHub\]](#)

Link Diagram DFA (Lexer) : [Link Diagram \[Draw.io\]](#)

Pembagian Tugas:

| Nama | NIM | Tugas | Kontribusi |
|-----------------------|----------|--|------------|
| Brian Ricardo Tamin | 13523126 | Implementasi Symbol Table, Laporan | 25% |
| Jovandra Otniel P. S. | 13523141 | Implementasi Type Checker, Laporan | 25% |
| Andrew Tedjapratama | 13523148 | Implementasi Decorated Abstract Syntax Tree, Laporan | 25% |
| Theo Kurniady | 13523154 | Implementasi AST Builder dan Struktur Data AST Node | 25% |