LAPORAN TUGAS BESAR
IF2224 TEORI BAHASA FORMAL DAN OTOMATA

MILESTONE 3 - Semantic Analysis

Disusun oleh

Kelompok 9 - TMP

Bertha Soliany Frandi	13523026
Brian Albar Hadian	13523048
Muhammad Izzat Jundy	13523092
Michael Alexander Angkawijaya	13523102

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2025

DAFTAR ISI

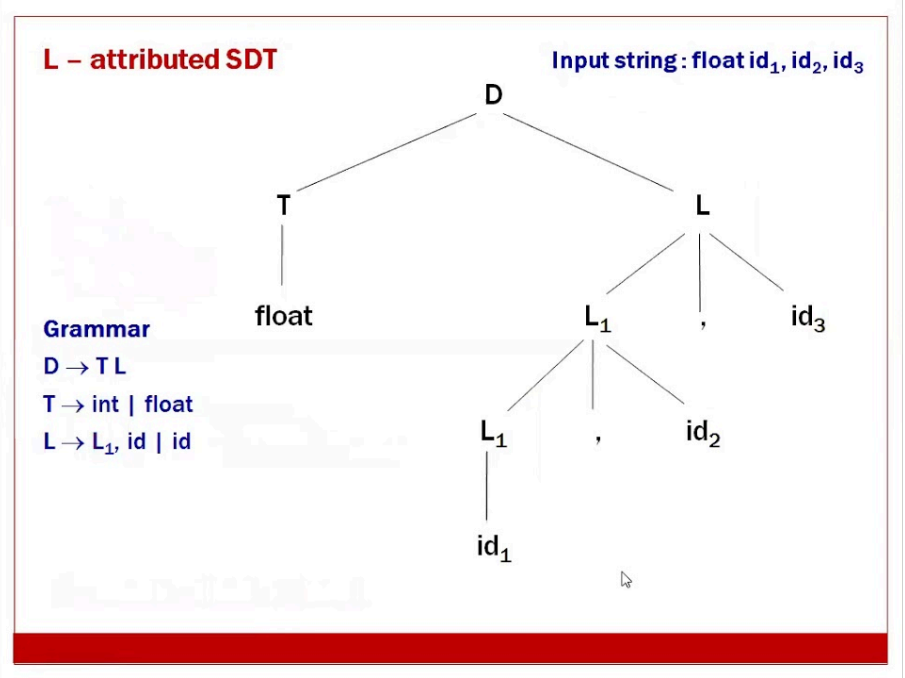
BAB I

LANDASAN TEORI

1.1 Semantic Analyzer

<pengertian semantic analyzer + peran>

Semantic analyzer merupakan komponen dalam desain kompiler yang bertujuan untuk menguji makna dan validitas dari urutan kata kunci (keyword) yang diberikan sebagai masukan. Tahap ini berperan penting untuk menghindari adanya operasi antara dua tipe yang berbeda yang seharusnya tidak valid, penggunaan variabel diluar lingkup tertentu, atau penggunaan keyword pada operasi yang tidak sesuai.



Gambar Tahapan *Semantic Analyzer*
(Sumber:https://i.ytimg.com/vi/AJGi8Fueo-0/maxresdefault.jpg)

Cara kerja dari *semantic analyzer* adalah dengan membuat terlebih dahulu tabel untuk menguji validitas bahwa suatu masukan dan format serta urutan dari kode memiliki makna yang tidak menyalahi cara pengolahan program serta membangun *parse tree* dengan menggunakan *parent node* dan/atau *sibling node* sebagai referensi untuk menentukan tipe dari *node* yang sedang diuji. *Semantic analyzer* memiliki beberapa pendekatan, diantaranya adalah *S-attributed grammar* dan *L-attributed grammar.* Dalam memproduksi *decorated Abstract Syntax Tree,* digunakan pendekatan *L-attributed grammar* yang menggunakan *parent node* dan *sibling node* sebagai referensi untuk menentukan tipe utama dari *node.*

Tujuan lain dari komponen *semantic analyzer* adalah untuk menguji validitas dari *keyword* yang digunakan bahwa apakah keyword tersebut masih diasumsikan ada pada lingkup (*scope*) tersebut atau tidak. Hal ini didekati dengan pendekatan

pembuatan tabel *tab, atab,* dan *btab* yang secara berurutan berfungsi untuk menyimpan metadata terkait *keyword* dan *identifier* yang ditemukan pada masukan, menyimpan metadata terkait *array* yang dibuat pada masukan, dan menyimpan metadata terkait *block* yang teridentifikasi ada pada masukan.

Tahap semantic analyzer diperlukan untuk menguji validitas berdasarkan makna yang dihasilkan dari urutan *keyword* yang diberikan oleh masukan selain dari pengujian urutan yang telah dilakukan pada proses *syntax analyzer*. Tahap ini akan berperan penting untuk proses selanjutnya, yakni *intermediate code generator*, yang bertujuan untuk menghasilkan kode antara yang dapat disimpan dalam berbagai format dan dijalankan pada berbagai platform dengan konfigurasi compiler sesuai dengan platform tersebut.

1.2 Context Free Grammar

Untuk mendeskripsikan struktur sintaksis sebuah bahasa pemrograman, digunakan suatu bentuk tata bahasa formal. Bentuk yang paling sering dipakai adalah Context-Free Grammar (CFG).

CFG adalah sebuah 4-tuple G=(V,T,P,S) yang terdiri atas:
V: himpunan non-terminal
T: himpunan terminal (biasanya token)
P: himpunan aturan produksi
S: simbol awal (start symbol)
Setiap produksi memiliki bentuk $A \rightarrow \alpha$, di mana $A \in V$ dan $\alpha \in (V \cup T)*$

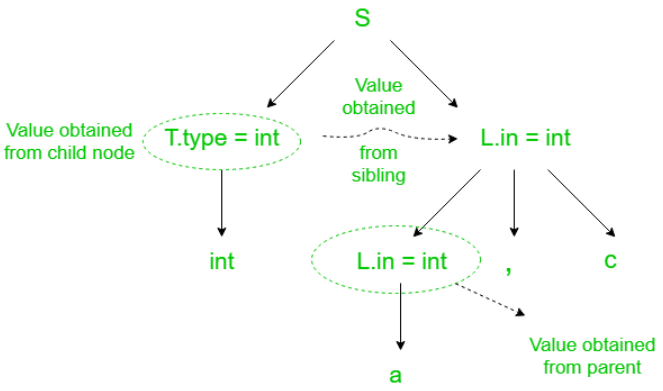
CFG digunakan secara luas untuk mendefinisikan sintaks bahasa pemrograman karena kemampuannya menggambarkan struktur rekursif, seperti ekspresi berpasangan kurung, blok program, atau pernyataan bersarang (Tutorialspoint, GfG). Grammar menjadi acuan parser untuk menentukan apakah suatu rangkaian token valid, menentukan bagaimana struktur sintaks disusun, dan memandu proses konstruksi parse tree.

Grammar yang baik harus bebas dari ambiguitas dan telah disesuaikan agar kompatibel dengan teknik parsing tertentu (mis., LL(1), LR(1), dll.)—Aho et al. (2007) menyebutnya sebagai grammar transformations.

1.3 Annotated Parse Tree

Annotated Parse tree adalah struktur pohon yang menunjukkan bagaimana token-token hasil lexical analysis membentuk sebuah konstruksi program berdasarkan aturan grammar dengan menggunakan tambahan node berupa definisi *block* dan *type* yang disematkan pada *identifier* serta beberapa tambahan production rule yang memiliki semantic rule yang dapat mengidentifikasi komponen tambahan tersebut Dalam annotated parse tree, setiap node non-terminal menggambarkan suatu aturan

produksi, sedangkan node terminal berada pada daun dan mewakili token asli dari program sama seperti parse tree biasa, tetapi terdapat beberapa tambahan node yang berfungsi untuk mengenali *block* (untuk membatasi *scope*) dan *type* (untuk melakukan pengujian terhadap *parent node* dan *sibling node*).



Contoh Gambar Annotated Parse Tree untuk *Conditional Statement*
(Sumber https://media.geeksforgeeks.org/wp-content/uploads/ag-3-1.png)

Struktur ini memberikan gambaran lengkap tentang bagaimana sebuah pernyataan atau ekspresi disusun sesuai grammar yang digunakan. Karena itu, parse tree sangat membantu untuk memahami struktur program dan memastikan bahwa urutan token mengikuti aturan sintaks. Parse tree juga sering menjadi dasar pembentukan struktur sintaks lanjutan seperti Abstract Syntax Tree (AST), yang lebih ringkas dan digunakan pada tahap kompilasi berikutnya.

1.4 Algoritma L-Attributed Grammar

L-Attributed Grammar merupakan salah satu pengembangan dari teknik parsing *recursive descent* yang memanfaatkan pendekatan *top-down* untuk menghasilkan AST dengan pengembangan berupa penambahan production rule untnuk mengidentifikasi scope dan type yang dimiliki oleh setiap *identifier node*. Pada metode ini, setiap terdapat pula penggunaan *tab, atab,* dan *btab* yang secara berurutan berfungsi untuk menyimpan metadata terkait *keyword* dan *identifier* yang ditemukan pada masukan, menyimpan metadata terkait *array* yang dibuat pada masukan, dan menyimpan metadata terkait *block* yang teridentifikasi ada pada masukan. Fungsi tersebut bertugas memeriksa type dari setiap token dengan menyalurkan type dari node paling dalam ke parent node yang membutuhkan (biasanya digunakan hingga mendekati root) dan memastikan bahwa pola yang muncul sesuai dengan aturan produksi *non-terminal* tersebut. Jika grammar menyatakan bahwa sebuah *non-terminal* terdiri dari beberapa simbol lain, maka fungsi yang bersangkutan akan memanggil fungsi-fungsi lain yang mewakili simbol-simbol tersebut.

Cara kerja *L-attributed grammar* bekerja dengan memanfaatkan production rule yang telah didefinisikan sebagaimana arsitektur dari bahasa pemrograman tersebut dengan tambahan production rule yang berfungsi untuk mengidentifikasi scope dan type dengan pendekatan DFS. Parser pertama-tama akan menelusuri seluruh node yang sudah ada pada Abstract Syntax Tree (AST) hasil *syntax analyzer* dan melakukan pengujian untuk menentukan type dari setiap node yang akan didapatkan ketika penelusuran telah mencapai node paling dalam, kemudian mencocokkan token tersebut dengan simbol yang diharapkan sesuai grammar dan validitas terhadap metadata yang disimpan oleh *tab, atab,* dan *btab* sesuai dengan tipe objek dari *identifier*. Jika cocok, parser akan menyatakan bahwa node terkini memiliki tipe sesuai yang telah ditentukan grammar dan hasil ini akan digunakan lebih lanjut oleh parent node dari node terkini sebagai referensi pembanding dengan sibling node. Hal ini terjadi secara berantai - penggunaan tipe dari ke dari node terdalam ke node teratas. Untuk setiap penentuan tipe dari *identifier* dan metadata yang dihasilkan untuk *array* dan *block*, akan dilakukan pencatatan metadata tersebut ke *tab, atab,* dan *btab* untuk dilakukan pengujian selanjutnya apabila dipanggil pada proses penelusuran selanjutnya. Jika node tersebut menyalahi aturan yang diberikan oleh production rule tambahan (semantic rule), parser akan melaporkan kesalahan sintaks dengan informasi yang menunjukkan bagian program mana yang tidak sesuai aturan.

Produk yang dihasilkan adalah *Decorated Abstract Syntax Tree (DAST)* yang berisi node sesuai *identifier* dan *keyword* dengan tambahan definisi mengenai *scope* dan *type.* Selain itu, produk lain yang dihasilkan adalah *tab, atab,* dan *btab* yang berisi metadata pada berbagai *identifier* dan *keyword* yang sesuai. Metode ini cocok digunakan ketika grammar berada dalam bentuk yang relatif sederhana dan tidak mengandung ambiguitas yang memerlukan backtracking. Dalam konteks Pascal-S, pendekatan L-Attributed Grammar dapat diterapkan secara langsung karena bentuk grammar-nya memungkinkan pemetaan non-terminal ke fungsi-fungsi parser tanpa penanganan konflik produksi yang rumit. Selain memvalidasi urutan token, parser juga dapat membangun parse tree dengan menambahkan node baru pada setiap pemanggilan fungsi non-terminal dan pendefinisian terhadap *scope* dan *type*, sehingga struktur sintaks dan semantic dapat direpresentasikan secara lengkap sesuai dengan grammar yang digunakan.

BAB II

PERANCANGAN & IMPLEMENTASI

2.1 Perancangan

Perancangan semantic analyzer dimulai dengan dekomposisi tanggung jawab menjadi tiga komponen utama sesuai spesifikasi Pascal-S: Symbol Table (TAB, BTAB, ATAB), Decorated AST Nodes, dan Semantic Visitor Functions. Symbol Table dirancang dengan tiga tabel terpisah yaitu TAB untuk menyimpan informasi identifier (konstanta, variabel, tipe, prosedur, fungsi), BTAB untuk menyimpan informasi blok prosedur dan definisi tipe record, serta ATAB untuk menyimpan detail array termasuk tipe indeks, batasan indeks, tipe elemen, dan ukuran array. Symbol Table menerapkan struktur linked list per blok menggunakan field Link untuk manajemen scope, serta Display register untuk akses cepat ke block pada setiap lexical level sesuai prinsip stack untuk scope management.

Decorated AST dirancang mengikuti konsep Attributed Grammar dengan pola inheritance menggunakan interface DecoratedNode yang mengharuskan setiap node menyimpan atribut semantik berupa TabIndex (referensi ke symbol table), Type (tipe data hasil analisis), Ref (pointer ke ATAB/BTAB untuk tipe komposit), Level (lexical level untuk scope), dan Address (offset memori). Terdapat 15 jenis node yang diimplementasikan mencakup literal nodes (NumberNode, RealNode, StringNode, BooleanNode, CharNode), expression nodes (BinOpNode, UnaryOpNode, VarNode), statement nodes (AssignNode, IfNode, WhileNode, ForNode, ProcCallNode), dan declaration nodes (VarDeclNode, ConstDeclNode, SubprogramDeclNode). Setiap node dilengkapi dengan field Errors dan Warnings untuk menyimpan hasil validasi semantik.

Semantic Analyzer dirancang menggunakan Syntax-Directed Translation Scheme dengan visitor pattern yang menerapkan algoritma L-Attributed Grammar. Setiap production rule dari grammar memiliki fungsi visitor (visit_<node>) yang bertugas: (1) memanggil visit pada child-nya secara depth-first traversal, (2) mengakses atau memperbarui symbol table dengan memasukkan deklarasi identifier dan melakukan lookup, (3) menghitung atribut/tipe semantik berdasarkan aturan kompatibilitas tipe, dan (4) menambahkan anotasi pada node dengan informasi tipe ekspresi dan referensi symbol table. Perancangan mencakup 42 semantic rules yang mengimplementasikan validasi untuk: Type Checking (konsistensi tipe data dengan aturan promosi tipe integer→real dan kompatibilitas integer↔char), Scope Checking (pencegahan forward reference, deteksi duplikasi deklarasi, validasi akses identifier melalui scope chain), Control Flow Validation (kondisi boolean untuk if/while, tipe integer untuk for loop), serta Parameter Matching (jumlah dan tipe argumen, constraint var parameter sebagai L-value).

2.2 Implementasi

Implementasi dilakukan dengan menggunakan bahasa Go. Penggunaan bahasa Go dimotivasi dari kecepatannya dalam melakukan kompilasi dibandingkan dengan bahasa lainnya dan penggunaan bahasa ini dapat memberikan kemampuan konkurensi yang cenderung sama cepatnya dengan bahasa C, tetapi memiliki *syntax* penulisan yang bersifat *high-level* sehingga dapat mendorong performa *compiler* dan memberikan kesempatan lebih untuk pengembangan selanjutnya.

Berikut adalah struktur folder dari program. Folder src menyimpan segala program yang akan dibuat dan folder test berisi semua test case. Terdapat pula folder doc yang akan berisi laporan ini dalam bentuk pdf dan diagram DFA. File dfa.txt merupakan file yang berisi aturan DFA.

```
TMP-TUBES-IF2224/
├── src/
│   ├── milestone1/
│   │   ├── dfa.go
│   │   ├── dfa.txt
│   │   ├── lexicalAnalyzer.go
│   │   └── tokenize.go
│   ├── milestone2/
│   │   ├── abstractSyntaxTree.go
│   │   ├── parser.go
│   │   └── syntaxAnalyzer.go
│   ├── milestone2/
│   │   ├── decoratedAst.go
│   │   ├── semanticAnalyzer.go
│   │   └── symbolTable.go
│   └── main.go
├── test/
│   ├── milestone-1/
│   │   ├── .gitkeep
│   │   ├── test_case1.pas
│   │   ├── test_case2.pas
│   │   ├── test_case3.pas
│   │   ├── test_case4.pas
│   │   ├── test_case5.pas
│   │   └── tester.pas
│   ├── milestone-2/
│   │   ├── .gitkeep
│   │   ├── test_case_unkown.txt
│   │   └── test_case_missing_semicolon.txt
│   └── output/
│       ├── tokens.txt
│       └── .gitkeep
├── doc/
│   ├── Laporan-1-TMP
│   ├── Diagram-1-TMP
│   └── Laporan-2-TMP
```

```
└── Laporan-3-TMP
└── README.md
```

2.2.1 Program

Program semantic analyzer memiliki alur eksekusi utama yang menerima input berupa Parse Tree hasil parsing, kemudian melakukan traversal secara depth-first untuk membangun symbol table dan decorated AST secara simultan menggunakan L-Attributed Grammar. Proses diawali dengan inisiasi symbol table yang mencakup pembuatan block 0 untuk global scope dan pengisian 29 reserved words pada TAB dengan indeks 0-28, sehingga identifier program dimulai dari indeks 29 sesuai spesifikasi. Ketika mengunjungi node <program>, analyzer memasukkan nama program ke symbol table dengan Obj=ObjProgram, lalu memproses <declaration-part> dengan mengunjungi setiap deklarasi secara berurutan menggunakan semantic visitor functions.

Untuk deklarasi variabel, program mengekstrak identifier list dan tipe, melakukan lookup pada symbol table untuk validasi forward reference pada user-defined types, kemudian memasukkan variabel ke symbol table (push scope) dengan menghitung offset berdasarkan ukuran tipe (integer/boolean/char = 1 byte, real = 8 bytes, array/record dihitung dari ATAB/BTAB) dan menyimpannya pada field Adr. Untuk deklarasi array, program membuat entry di ATAB dengan validasi bahwa bounds harus berupa konstanta compile-time, menghitung atribut tipe dengan rumus Size = Elsz × (High - Low + 1), dan mendukung multi-dimensional arrays melalui chaining field Eref. Untuk deklarasi record, program membuat block baru di BTAB melalui push scope, memproses field list dengan memasukkan setiap field sebagai ObjField dengan offset bertahap, dan menghitung total ukuran record yang disimpan pada field Vsze.

Pada deklarasi subprogram (prosedur/fungsi), program memasuki scope baru dengan membuat level dan block baru (push scope pada stack), memproses parameter list dengan mendeteksi keyword variabel untuk var parameter (Nrm=0), memasukkan parameter ke symbol table dengan ukuran disimpan pada Psze, lalu untuk fungsi menambahkan implicit return variable dengan nama sama dengan fungsi dan tipe sesuai return type. Program melakukan validasi duplikasi subprogram dengan memeriksa apakah identifier sudah ada di current scope melalui lookup pada symbol table dan memastikan objeknya adalah procedure/function. Setelah memproses body, analyzer memvalidasi bahwa fungsi memiliki minimal satu assignment ke nama fungsinya sendiri untuk memastikan return value.

Untuk statement processing, program mengunjungi setiap node AST secara depth-first dan melakukan kalkulasi tipe data serta validasi semantik: assignment statement memvalidasi kompatibilitas tipe antara target dan value dengan type promotion (integer → real), if/while statement memvalidasi tipe

ekspresi kondisi harus boolean, dan for loop memvalidasi loop variable bertipe integer serta start/end expressions bertipe integer. Untuk expression evaluation, program menghitung atribut tipe hasil operasi berdasarkan aturan semantik: operator relasional menghasilkan boolean, operator aritmetika dengan operand real menghasilkan real (type promotion), operator / selalu menghasilkan real (real division), operator bagi/div/mod memerlukan operand integer dan menghasilkan integer (integer division).

Variable access mendukung pola kompleks seperti arr[i].field (akses field dari elemen array) dan rec.field[i] (array indexing pada field record). Program memproses dengan mengikuti rantai referensi: untuk pattern pertama, resolusi array indexing dilakukan melalui ATAB chain mengikuti field Eref kemudian lookup field di BTAB; untuk pattern kedua, field access dilakukan terlebih dahulu melalui lookup di BTAB kemudian array indexing. Validasi mencakup pemeriksaan tipe index (harus integer), keberadaan field di record melalui traversal linked list Last→Link, dan bounds checking untuk multi-dimensional arrays.

Parameter validation pada procedure/function call dilakukan dengan mengambil parameter list dari BTAB entry, traverse linked list dari Lpar melalui field Link untuk mengumpulkan semua parameter, kemudian memvalidasi konsistensi tipe: (1) jumlah argument sama dengan jumlah parameter, (2) tipe setiap argument compatible dengan parameter type berdasarkan aturan promosi tipe, dan (3) untuk var parameter (Nrm=0), argument harus berupa L-value (variabel yang bisa di-assign). Error messages yang dihasilkan bersifat deskriptif dengan informasi posisi argument, nama procedure, expected type, dan actual type untuk memudahkan debugging.

Hasil akhir program berupa Decorated AST yang setiap nodenya telah didekorasi dengan informasi tipe ekspresi (diperoleh melalui kalkulasi atribut tipe dari operand), referensi ke symbol table (TabIndex), informasi scope (Level), dan address, serta Symbol Table lengkap (TAB, BTAB, ATAB) yang dapat digunakan untuk tahap code generation. Program juga mengakumulasi semantic errors dan warnings hasil validasi semantik yang ditampilkan dengan format terstruktur, memungkinkan compiler melaporkan multiple errors dalam satu run sesuai prinsip robust error handling.

Berikut adalah penjelasan mengenai fungsi yang terdapat pada program di berbagai file.

2.2.1.1 Symbol Table (symbolTable.go)

Nama Fungsi/Tipe	Penjelasan
type ObjectClass	Enumerasi untuk kelas objek: ObjConstant, ObjVariable, ObjType, ObjProcedure, ObjFunction, ObjProgram, ObjField

func (o ObjectClass) String()	Method dari ObjectClass. Mengkonversi ObjectClass ke string representasi. Tipe keluaran: string
type TypeKind	Enumerasi untuk jenis tipe data: TypeNone, TypeInteger, TypeBoolean, TypeChar, TypeReal, TypeArray, TypeRecord
func (t TypeKind) String()	Method dari TypeKind. Mengkonversi TypeKind ke string representasi. Tipe keluaran: string
type TabEntry	Struktur data untuk entry dalam TAB (symbol table). Fields: Identifier, Link, Obj, Type, Ref, Nrm, Lev, Adr
type AtabEntry	Struktur data untuk entry dalam ATAB (array table). Fields: Xtyp, Etyp, Eref, Low, High, Elsz, Size
type BtabEntry	Struktur data untuk entry dalam BTAB (block table). Fields: Last, Lpar, Psze, Vsze
type SymbolTable	Struktur data utama symbol table. Fields: Tab, Btab, Atab, TabIndex, BtabIndex, AtabIndex, CurrentLevel, CurrentBlock, Display, ReservedWordsCount
func NewSymbolTable()	Membuat symbol table baru dengan inisialisasi block 0 dan reserved words. Tipe keluaran: *SymbolTable
func (st *SymbolTable) initReservedWords()	Method dari SymbolTable. Menginisialisasi 29 reserved words Pascal-S ke dalam TAB (indeks 0-28). Tidak ada keluaran.
func (st *SymbolTable) enterBlock()	Method dari SymbolTable. Membuat block baru di BTAB dan mengupdate CurrentBlock. Tipe keluaran: int (block index)
func (st *SymbolTable) enterLevel()	Method dari SymbolTable. Masuk ke nested level baru (increment CurrentLevel) untuk scope management. Tidak ada keluaran.
func (st *SymbolTable)	Method dari SymbolTable. Masuk ke nested

enterLevelWithBlock()	level baru dengan membuat block baru dan mengupdate Display register. Tipe keluaran: int (block index)
func (st *SymbolTable) exitLevel()	Method dari SymbolTable. Keluar dari nested level (decrement CurrentLevel) dan restore CurrentBlock dari Display. Tidak ada keluaran.
func (st *SymbolTable) Enter(...)	Method dari SymbolTable. Menambahkan identifier baru ke TAB dengan parameter: identifier, obj, typ, ref, nrm, adr. Mengupdate linked list di block. Tipe keluaran: int (TAB index)
func (st *SymbolTable) EnterArray(...)	Method dari SymbolTable. Menambahkan entry ke ATAB dengan parameter: xtyp, etyp, eref, low, high, elsz. Menghitung total size array. Tipe keluaran: int (ATAB index)
func (st *SymbolTable) EnterBlock()	Method dari SymbolTable. Wrapper untuk enterBlock(). Tipe keluaran: int (block index)
func (st *SymbolTable) AddVariableSize(amount int)	Method dari SymbolTable. Menambahkan ukuran variabel lokal ke Vsze pada BTAB block saat ini. Tidak ada keluaran.
func (st *SymbolTable) AddParameterSize(amount int)	Method dari SymbolTable. Menambahkan ukuran parameter ke Psze pada BTAB block saat ini. Tidak ada keluaran.
func (st *SymbolTable) UpdateBlockLastParam(...)	Method dari SymbolTable. Mengupdate field Lpar pada BTAB block tertentu dengan index parameter terakhir. Parameter: blockIndex, lpar. Tidak ada keluaran.
func (st *SymbolTable) Lookup(identifier string)	Method dari SymbolTable. Mencari identifier di symbol table dari scope saat ini ke global (level 0) melalui Display register dan linked list. Tipe keluaran: int, bool (TAB index, found)
func (st SymbolTable) LookupInCurrentScope(...)	Method dari SymbolTable. Mencari identifier hanya di scope/block saat ini menggunakan linked list. Tipe keluaran: int,

	bool (TAB index, found)
func (st *SymbolTable) GetEntry(index int)	Method dari SymbolTable. Mengambil entry dari TAB berdasarkan index dengan bounds checking. Tipe keluaran: *TabEntry, error
func (st *SymbolTable) GetArrayEntry(index int)	Method dari SymbolTable. Mengambil entry dari ATAB berdasarkan index dengan bounds checking. Tipe keluaran: *AtabEntry, error
func (st *SymbolTable) GetBlockEntry(index int)	Method dari SymbolTable. Mengambil entry dari BTAB berdasarkan index dengan bounds checking. Tipe keluaran: *BtabEntry, error
func (st *SymbolTable) getTypeSize(typ, ref int)	Method dari SymbolTable. Menghitung ukuran tipe dalam byte/unit memori. Untuk tipe komposit (array/record), mengakses ATAB/BTAB via ref. Tipe keluaran: int
func (st *SymbolTable) IsDeclared(identifier)	Method dari SymbolTable. Memeriksa apakah identifier sudah dideklarasikan di scope manapun (wrapper untuk Lookup). Tipe keluaran: bool
func (st *SymbolTable) IsDeclaredInCurrentScope(...)	Method dari SymbolTable. Memeriksa apakah identifier sudah dideklarasikan di scope saat ini (wrapper untuk LookupInCurrentScope). Tipe keluaran: bool
func (st *SymbolTable) PrintSymbolTable()	Method dari SymbolTable. Mencetak seluruh isi TAB, BTAB, dan ATAB dalam format tabel untuk debugging. Tidak ada keluaran (print ke stdout).

2.2.1.2 Decorated AST (decoratedAst.go)

Nama Fungsi/Tipe	Penjelasan
type DecoratedNode	Interface untuk semua decorated AST nodes. Methods: GetTabIndex(), GetType(), GetLevel(), Accept(visitor)

type DecoratedNodeVisitor	Interface untuk visitor pattern. Methods untuk visit setiap jenis node (total 15 methods)
type BaseDecoratedNode	Struktur data dasar untuk semua decorated nodes. Fields: TabIndex, Type, Ref, Level, Address, Errors, Warnings
func (n *BaseDecoratedNode) GetTabIndex()	Method dari BaseDecoratedNode. Mengembalikan index di symbol table. Tipe keluaran: int
func (n *BaseDecoratedNode) GetType()	Method dari BaseDecoratedNode. Mengembalikan tipe data node. Tipe keluaran: TypeKind
func (n *BaseDecoratedNode) GetLevel()	Method dari BaseDecoratedNode. Mengembalikan lexical level node. Tipe keluaran: int
type ProgramNode	Node untuk keseluruhan program. Fields: BaseDecoratedNode, Name, Declarations, Block
func NewProgramNode(name string)	Membuat ProgramNode baru dengan inisialisasi fields default. Tipe keluaran: *ProgramNode
func (n *ProgramNode) Accept(visitor)	Method dari ProgramNode. Implementasi visitor pattern untuk traversal. Parameter: DecoratedNodeVisitor. Tidak ada keluaran.
type DeclarationListNode	Node untuk list deklarasi. Fields: BaseDecoratedNode, Declarations []DecoratedNode
func NewDeclarationListNode(declarations)	Membuat DeclarationListNode baru. Tipe keluaran: *DeclarationListNode
func (n *DeclarationListNode) Accept(visitor)	Method dari DeclarationListNode. Visit semua deklarasi secara iteratif. Tidak ada keluaran.
type VarDeclNode	Node untuk deklarasi variabel. Fields:

	BaseDecoratedNode, Name
func NewVarDeclNode(name, typ)	Membuat VarDeclNode baru dengan nama dan tipe. Tipe keluaran: *VarDeclNode
func (n *VarDeclNode) Accept(visitor)	Method dari VarDeclNode. Implementasi visitor pattern. Tidak ada keluaran.
type ConstDeclNode	Node untuk deklarasi konstanta. Fields: BaseDecoratedNode, Name, Value
func NewConstDeclNode(name, value, typ)	Membuat ConstDeclNode baru dengan nama, nilai, dan tipe. Tipe keluaran: *ConstDeclNode
func (n *ConstDeclNode) Accept(visitor)	Method dari ConstDeclNode. Implementasi visitor pattern. Tidak ada keluaran.
type TypeDeclNode	Node untuk deklarasi tipe. Fields: BaseDecoratedNode, Name
type SubprogramDeclNode	Node untuk deklarasi prosedur/fungsi. Fields: BaseDecoratedNode, Name, Parameters, ReturnType, Body, IsFunction
func NewSubprogramDeclNode(...)	Membuat SubprogramDeclNode baru dengan parameter lengkap. Tipe keluaran: *SubprogramDeclNode
func (n *SubprogramDeclNode) Accept(visitor)	Method dari SubprogramDeclNode. Implementasi visitor pattern (TODO). Tidak ada keluaran.
type BlockNode	Node untuk compound statement (mulai...selesai). Fields: BaseDecoratedNode, Statements []DecoratedNode, BlockIndex
func NewBlockNode(statements)	Membuat BlockNode baru dengan list statements. Tipe keluaran: *BlockNode
func (n *BlockNode) Accept(visitor)	Method dari BlockNode. Implementasi visitor pattern. Tidak ada keluaran.

type AssignNode	Node untuk assignment statement. Fields: BaseDecoratedNode, Target, Value
func NewAssignNode(target, value)	Membuat AssignNode baru dengan target dan value nodes. Tipe keluaran: *AssignNode
func (n *AssignNode) Accept(visitor)	Method dari AssignNode. Implementasi visitor pattern. Tidak ada keluaran.
type BinOpNode	Node untuk operasi binary. Fields: BaseDecoratedNode, Operator, Left, Right
func NewBinOpNode(operator, left, right)	Membuat BinOpNode baru dengan operator dan operand. Tipe keluaran: *BinOpNode
func (n *BinOpNode) Accept(visitor)	Method dari BinOpNode. Implementasi visitor pattern. Tidak ada keluaran.
type UnaryOpNode	Node untuk operasi unary. Fields: BaseDecoratedNode, Operator, Operand
func NewUnaryOpNode(operator, operand)	Membuat UnaryOpNode baru dengan operator dan operand. Tipe keluaran: *UnaryOpNode
func (n *UnaryOpNode) Accept(visitor)	Method dari UnaryOpNode. Implementasi visitor pattern. Tidak ada keluaran.
type VarNode	Node untuk referensi variabel. Fields: BaseDecoratedNode, Name, IsLValue, IsIndexed, Index
func NewVarNode(name)	Membuat VarNode baru dengan nama variabel. Tipe keluaran: *VarNode
func (n *VarNode) Accept(visitor)	Method dari VarNode. Implementasi visitor pattern. Tidak ada keluaran.
type NumberNode	Node untuk literal integer. Fields: BaseDecoratedNode, Value int
func	Membuat NumberNode baru dengan nilai

NewNumberNode(value)	integer. Tipe keluaran: *NumberNode
func (n *NumberNode) Accept(visitor)	Method dari NumberNode. Implementasi visitor pattern. Tidak ada keluaran.
type RealNode	Node untuk literal real number. Fields: BaseDecoratedNode, Value float64
func NewRealNode(value)	Membuat RealNode baru dengan nilai float64. Tipe keluaran: *RealNode
func (n *RealNode) Accept(visitor)	Method dari RealNode. Implementasi visitor pattern. Tidak ada keluaran.
type StringNode	Node untuk literal string. Fields: BaseDecoratedNode, Value string
func NewStringNode(value)	Membuat StringNode baru dengan nilai string. Tipe keluaran: *StringNode
func (n *StringNode) Accept(visitor)	Method dari StringNode. Implementasi visitor pattern. Tidak ada keluaran.
type BooleanNode	Node untuk literal boolean. Fields: BaseDecoratedNode, Value bool
func NewBooleanNode(value)	Membuat BooleanNode baru dengan nilai boolean. Tipe keluaran: *BooleanNode
func (n *BooleanNode) Accept(visitor)	Method dari BooleanNode. Implementasi visitor pattern. Tidak ada keluaran.
type CharNode	Node untuk literal character. Fields: BaseDecoratedNode, Value rune
func NewCharNode(value)	Membuat CharNode baru dengan nilai rune. Tipe keluaran: *CharNode
func (n *CharNode) Accept(visitor)	Method dari CharNode. Implementasi visitor pattern. Tidak ada keluaran.
type ProcCallNode	Node untuk pemanggilan prosedur/fungsi. Fields: BaseDecoratedNode, Name, Arguments []DecoratedNode, IsBuiltIn

func NewProcCallNode(name, arguments)	Membuat ProcCallNode baru dengan nama dan arguments. Tipe keluaran: *ProcCallNode
func (n *ProcCallNode) Accept(visitor)	Method dari ProcCallNode. Implementasi visitor pattern. Tidak ada keluaran.
type IfNode	Node untuk if statement. Fields: BaseDecoratedNode, Condition, ThenStmt, ElseStmt
func (n *IfNode) Accept(visitor)	Method dari IfNode. Implementasi visitor pattern. Tidak ada keluaran.
type WhileNode	Node untuk while loop. Fields: BaseDecoratedNode, Condition, Body
func (n *WhileNode) Accept(visitor)	Method dari WhileNode. Implementasi visitor pattern. Tidak ada keluaran.
type ForNode	Node untuk for loop. Fields: BaseDecoratedNode, Variable, StartValue, EndValue, Body, IsDownTo
func (n *ForNode) Accept(visitor)	Method dari ForNode. Implementasi visitor pattern. Tidak ada keluaran.
func PrintDecoratedAST(node, prefix, isLast)	Mencetak decorated AST dalam format tree dengan visual connectors. Parameter: DecoratedNode, string, bool. Tidak ada keluaran (print ke stdout).
func printDeclarationList(node, prefix)	Helper untuk mencetak declaration list dengan connectors. Tidak ada keluaran.
func printBlockStatements(node, prefix)	Helper untuk mencetak block statements dengan connectors. Tidak ada keluaran.
func formatNodeInline(node)	Helper untuk format node secara inline (single line). Tipe keluaran: string
func formatBinOpMultiline(node,	Helper untuk format BinOpNode secara

prefix)	multiline. Tipe keluaran: string
func isBinOp(node)	Helper untuk cek apakah node adalah BinOpNode. Tipe keluaran: bool
func getSpaces(n)	Helper untuk generate string berisi n spasi. Tipe keluaran: string

2.2.1.3 Semantic Analyzer (semanticAnalyzer.go)

Nama Fungsi/Tipe	Penjelasan
type SemanticAnalyzer	Struktur data untuk semantic analyzer. Fields: SymTable, CurrentOffset, Errors, Warnings
func NewSemanticAnalyzer()	Membuat semantic analyzer baru dengan symbol table terisi reserved words. Tipe keluaran: *SemanticAnalyzer
func (sa *SemanticAnalyzer) Analyze(parseTree)	Method dari SemanticAnalyzer. Melakukan analisis semantik pada parse tree dan membangun decorated AST + symbol table. Tipe keluaran: DecoratedNode, error
func (sa *SemanticAnalyzer) GetSymbolTable()	Method dari SemanticAnalyzer. Mengembalikan pointer ke symbol table. Tipe keluaran: *SymbolTable
func (sa *SemanticAnalyzer) GetErrors()	Method dari SemanticAnalyzer. Mengembalikan slice semua error yang terdeteksi. Tipe keluaran: []string
func (sa *SemanticAnalyzer) GetWarnings()	Method dari SemanticAnalyzer. Mengembalikan slice semua warning yang terdeteksi. Tipe keluaran: []string
func (sa *SemanticAnalyzer) visitProgram(node)	Method dari SemanticAnalyzer. Visit <program> node, extract nama program, masukkan ke symbol table, visit declarations dan block. Tipe keluaran: *ProgramNode
func (sa *SemanticAnalyzer)	Method dari SemanticAnalyzer. Visit <declaration-part> node, memproses semua

visitDeclarationPart(node)	deklarasi (const, type, var, subprogram) secara berurutan. Tipe keluaran: DecoratedNode
func (sa *SemanticAnalyzer) visitVarDeclaration(node)	Method dari SemanticAnalyzer. Visit <var-declaration> node, extract identifier list dan type, masukkan ke TAB dengan offset calculation, validasi forward reference. Tipe keluaran: DecoratedNode
func (sa *SemanticAnalyzer) visitConstDeclaration(node)	Method dari SemanticAnalyzer. Visit <const-declaration> node, extract identifier dan value, infer tipe, masukkan ke TAB. Tipe keluaran: DecoratedNode
func (sa *SemanticAnalyzer) visitTypeDeclaration(node)	Method dari SemanticAnalyzer. Visit <type-declaration> node, proses tipe dan masukkan ke TAB (tidak muncul di decorated AST). Tipe keluaran: DecoratedNode
func (sa *SemanticAnalyzer) visitSubprogramDeclaration(node)	Method dari SemanticAnalyzer. Visit <subprogram-declaration> node, create level/block baru, proses parameter, validasi duplikasi subprogram, validasi function return assignment. Tipe keluaran: DecoratedNode
func (sa *SemanticAnalyzer) visitCompoundStatement(node)	Method dari SemanticAnalyzer. Visit <compound-statement> node (mulai...selesai), extract statement list dan buat BlockNode. Tipe keluaran: *BlockNode
func (sa *SemanticAnalyzer) visitStatementList(node)	Method dari SemanticAnalyzer. Visit <statement-list> node, memproses semua statement secara berurutan. Tipe keluaran: []DecoratedNode
func (sa *SemanticAnalyzer) visitStatement(node)	Method dari SemanticAnalyzer. Router untuk visit berbagai jenis statement (assignment, procedure call, if, while, for, compound). Tipe keluaran: DecoratedNode
func (sa *SemanticAnalyzer) visitAssignmentStatement(n	Method dari SemanticAnalyzer. Visit <assignment-statement>, lookup target variable, visit value expression, validasi

ode)	type compatibility. Tipe keluaran: *AssignNode
func (sa *SemanticAnalyzer) visitExpression(node)	Method dari SemanticAnalyzer. Visit <expression> node, handle simple expression atau relational operation (expr relop expr). Tipe keluaran: DecoratedNode
func (sa *SemanticAnalyzer) visitSimpleExpression(node)	Method dari SemanticAnalyzer. Visit <simple-expression> node, handle term atau chained additive operations (term addop term ...). Tipe keluaran: DecoratedNode
func (sa *SemanticAnalyzer) visitTerm(node)	Method dari SemanticAnalyzer. Visit <term> node, handle factor atau chained multiplicative operations (factor mulop factor ...), validasi tipe untuk operator /, bagi, div, mod, dan. Tipe keluaran: DecoratedNode
func (sa *SemanticAnalyzer) visitFactor(node)	Method dari SemanticAnalyzer. Visit <factor> node, handle literals (NUMBER, REAL, STRING, CHAR, BOOLEAN), variable, function call, unary operators (tidak, +, -), parenthesized expression. Tipe keluaran: DecoratedNode
func (sa *SemanticAnalyzer) visitVariable(node)	Method dari SemanticAnalyzer. Visit <variable> node, handle simple variable, array indexing (multi-dimensional), record field access, complex combinations (arr[i].field, rec.field[i]). Tipe keluaran: DecoratedNode
func (sa *SemanticAnalyzer) processFieldAccess(varNode, field)	Method dari SemanticAnalyzer. Helper untuk memproses akses field pada record, lookup field di BTAB, validasi tipe record. Tipe keluaran: *VarNode
func (sa *SemanticAnalyzer) visitIdentifier(node)	Method dari SemanticAnalyzer. Visit IDENTIFIER node, lookup di symbol table, buat VarNode dengan informasi lengkap dari TAB entry. Tipe keluaran: *VarNode
func (sa *SemanticAnalyzer) visitProcedureCall(node)	Method dari SemanticAnalyzer. Visit <procedure-call> node, extract nama dan arguments, lookup di symbol table, validasi

	parameter (jumlah dan tipe), deteksi built-in procedures. Tipe keluaran: *ProcCallNode
func (sa *SemanticAnalyzer) visitFunctionCall(node)	Method dari SemanticAnalyzer. Visit <function-call> node (untuk function call dalam expression), extract nama dan arguments, lookup di symbol table, set return type, validasi parameter. Tipe keluaran: *ProcCallNode
func (sa *SemanticAnalyzer) visitIfStatement(node)	Method dari SemanticAnalyzer. Visit <if-statement> node, extract condition dan statements (then/else), validasi condition bertipe boolean. Tipe keluaran: *IfNode
func (sa *SemanticAnalyzer) visitWhileStatement(node)	Method dari SemanticAnalyzer. Visit <while-statement> node, extract condition dan body, validasi condition bertipe boolean. Tipe keluaran: *WhileNode
func (sa *SemanticAnalyzer) visitForStatement(node)	Method dari SemanticAnalyzer. Visit <for-statement> node, extract loop variable, start/end expressions, direction (ke/turun_ke), body. Validasi: loop variable declared dan integer, start/end bertipe integer. Tipe keluaran: *ForNode
func (sa *SemanticAnalyzer) checkProcedureArguments(...)	Method dari SemanticAnalyzer. Helper untuk validasi argumen procedure/function call: cek jumlah argument, cek type compatibility setiap argument, cek var parameter constraint (L-value). Parameter: name, entry, arguments. Tidak ada keluaran (add error jika ada masalah).
func (sa *SemanticAnalyzer) isLValue(node)	Method dari SemanticAnalyzer. Helper untuk cek apakah node adalah L-value (dapat di-assign). Tipe keluaran: bool
func (sa *SemanticAnalyzer) processType(node)	Method dari SemanticAnalyzer. Memproses <type> node, handle built-in types (integer, real, boolean, char), array type, record type, user-defined type (lookup). Tipe keluaran: TypeKind, int (type, ref)
func (sa *SemanticAnalyzer)	Method dari SemanticAnalyzer. Memproses <array-type> node, extract bounds

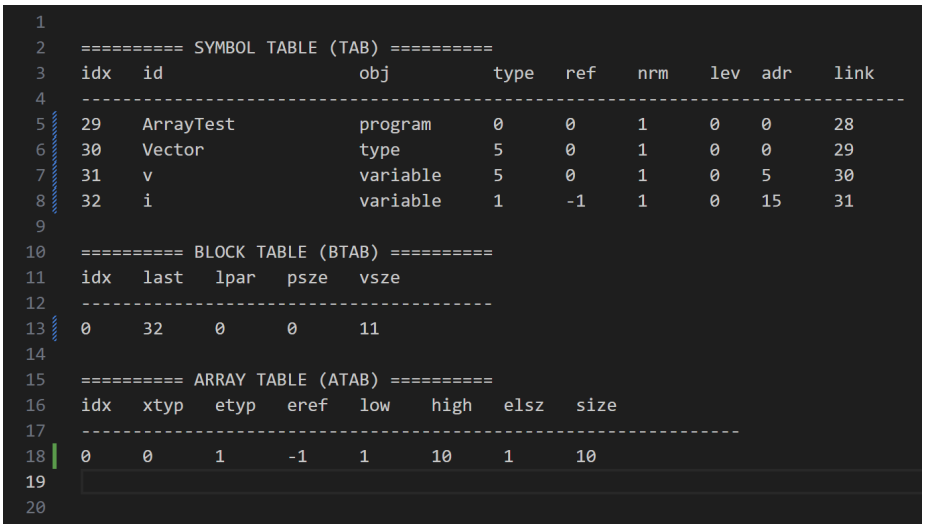
processArrayType(node)	(low/high), validasi bounds adalah konstanta compile-time, proses element type, hitung size, buat entry di ATAB. Tipe keluaran: TypeKind, int (TypeArray, ATAB index)
func (sa *SemanticAnalyzer) processRecordType(node)	Method dari SemanticAnalyzer. Memproses <record-type> node, buat block baru di BTAB, proses field list, hitung total size record. Tipe keluaran: TypeKind, int (TypeRecord, BTAB index)
func (sa *SemanticAnalyzer) processFieldList(node, blockIdx)	Method dari SemanticAnalyzer. Memproses <field-list> untuk record type, extract identifier list dan type, masukkan field ke TAB dengan ObjField, update offset. Parameter: node, blockIndex. Tidak ada keluaran.
func (sa *SemanticAnalyzer) extractIdentifierList(node)	Method dari SemanticAnalyzer. Extract list identifier dari <identifier-list> node. Tipe keluaran: []string
type Parameter	Struktur data helper untuk parameter. Fields: Name, Type, Ref, Nrm
func (sa *SemanticAnalyzer) extractParameters(node)	Method dari SemanticAnalyzer. Extract parameter list dari <parameter-list> node, deteksi var parameter (nrm=0) vs value parameter (nrm=1), extract nama, tipe, dan ref. Tipe keluaran: []Parameter
func (sa *SemanticAnalyzer) extractConstValue(tokenValue)	Method dari SemanticAnalyzer. Extract nilai dan tipe dari token konstanta (NUMBER, true/false, STRING_LITERAL). Tipe keluaran: int, TypeKind (value, type)
func (sa *SemanticAnalyzer) getNodeType(node)	Method dari SemanticAnalyzer. Mendapatkan tipe data dari decorated node (type inference). Tipe keluaran: TypeKind
func (sa *SemanticAnalyzer) typesCompatible(type1, type2)	Method dari SemanticAnalyzer. Cek kompatibilitas dua tipe: exact match, integer↔real promotion, integer↔char compatibility. Tipe keluaran: bool

func (sa *SemanticAnalyzer) isNumericType(typ)	Method dari SemanticAnalyzer. Cek apakah tipe adalah numeric (integer, char, real). Tipe keluaran: bool
func (sa *SemanticAnalyzer) addError(message)	Method dari SemanticAnalyzer. Menambahkan pesan error ke list errors. Parameter: string. Tidak ada keluaran.
func (sa *SemanticAnalyzer) addWarning(message)	Method dari SemanticAnalyzer. Menambahkan pesan warning ke list warnings. Parameter: string. Tidak ada keluaran.
func extractValue(tokenValue)	Function helper untuk extract value dari format token TYPE(value). Tipe keluaran: string
func (sa *SemanticAnalyzer) checkFunctionHasReturnAssignment(...)	Method dari SemanticAnalyzer. Cek apakah function body mengandung assignment ke nama fungsi (minimal satu kali). Parameter: body, funcName. Tipe keluaran: bool
func (sa *SemanticAnalyzer) checkStatementForReturnAssignment(...)	Method dari SemanticAnalyzer. Recursive helper untuk cek return assignment di dalam statement (termasuk nested blocks, if/else, loops). Parameter: stmt, funcName. Tipe keluaran: bool

BAB III

PENGUJIAN

3.1 Kasus 1

Input	program ArrayTest; tipe Vector = larik [1..10] dari integer; variabel v: Vector; i: integer; mulai v[1] := 100; i := v[1]; writeln(i) selesai.
Output	

```
========== SYMBOL TABLE (TAB) ==========
idx   id                  obj          type   ref    nrm    lev  adr    link
-------------------------------------------------------------------------------
29    ArrayTest           program      0      0      1      0    0      28
30    Vector              type         5      0      1      0    0      29
31    v                   variable     5      0      1      0    5      30
32    i                   variable     1      -1     1      0    15     31

========== BLOCK TABLE (BTAB) ==========
idx   last   lpar   psze   vsze
---------------------------------------
0     32     0      0      11

========== ARRAY TABLE (ATAB) ==========
idx   xtyp   etyp   eref   low    high   elsz   size
-------------------------------------------------------------
0     0      1      -1     1      10     1      10
```


```
ProgramNode(name: 'ArrayTest')
|
+-- Declarations
|   |
|   +-- VarDecl(name: 'v', type: 'array')
|   \-- VarDecl(name: 'i', type: 'integer')
|
\-- Block
    |
    +-- Assign(target: Var('v'), value: Num(100))
    |
    +-- Assign(target: Var('i'), value: Var('v'))
    |
    \-- ProcedureCall(name: 'writeln',
                      args: [Var('i')])
```

3.2 Kasus 2

Input	program TestRecord; tipe Mahasiswa = rekaman nim: integer; nama: integer; ipk: integer selesai; variabel mhs: Mahasiswa; mulai mhs := 1; writeln('Done') selesai.

Output

```
========== SYMBOL TABLE (TAB) ==========
idx   id                  obj          type   ref    nrm    lev  adr    link
------------------------------------------------------------------------------
29    TestRecord          program      0      0      1      0    0      28
30    nim                 field        1      -1     1      0    0      -1
31    nama                field        1      -1     1      0    1      30
32    ipk                 field        1      -1     1      0    2      31
33    Mahasiswa           type         6      1      1      0    0      29
34    mhs                 variable     6      1      1      0    5      33

========== BLOCK TABLE (BTAB) ==========
idx   last   lpar   psze   vsze
---------------------------------------
0     34     0      0      3
1     32     0      0      3

========== ARRAY TABLE (ATAB) ==========
idx   xtyp   etyp   eref   low    high   elsz   size
---------------------------------------------------------------
```

```
ProgramNode(name: 'TestRecord')
|
+-- Declarations
|   +-- VarDecl(name: 'mhs', type: 'record')
|
\-- Block
    |
    +-- Assign(target: Var('mhs'), value: Num(1))
    |
    \-- ProcedureCall(name: 'writeln',
                      args: [String('Done')])
```

3.3 Kasus 3

Input	program TestErrors; variabel a, b: integer; mulai a := 5; c := 10; b := a + c; selesai.

Output	========== MILESTONE 3: SEMANTIC ANALYSIS ========== Performing semantic analysis... Semantic analysis failed: semantic analysis failed with 3 error(s) Semantic errors: 1. Undefined variable 'c' 2. Undefined identifier 'c' 3. Arithmetic operator requires numeric operands ========== SYMBOL TABLE ========== ========== SYMBOL TABLE (TAB) ========== idx id obj type ref nrm lev adr link --- 29 TestErrors program 0 0 1 0 0 28 30 a variable 1 -1 1 0 5 29 31 b variable 1 -1 1 0 6 30 ========== BLOCK TABLE (BTAB) ========== idx last lpar psze vsze --- 0 31 0 0 2 ========== ARRAY TABLE (ATAB) ========== idx xtyp etyp eref low high elsz size --- ========== DECORATED AST ========== ProgramNode(name: 'TestErrors') | +-- Declarations | | | +-- VarDecl(name: 'a', type: 'integer') | \-- VarDecl(name: 'b', type: 'integer') | \-- Block | +-- Assign(target: Var('a'), value: Num(5)) | +-- Assign(target: Var('c'), value: Num(10)) | \-- Assign(target: Var('b'), value: BinOp(op: '+', left: Var('a'), right: Var('c'))) Symbol table saved to ../test/output/symbol-table.txt Decorated AST saved to ../test/output/decorated-ast.txt

3.4 Kasus 4

Input	program TestDuplicate; variabel x: integer; x: integer;

	y: integer; mulai x := 5; y := x + 10; selesai.
Output	========== MILESTONE 3: SEMANTIC ANALYSIS ========== Performing semantic analysis... Semantic analysis failed: semantic analysis failed with 1 error(s) Semantic errors: 1. Duplicate variable declaration: x ========== SYMBOL TABLE ========== ========== SYMBOL TABLE (TAB) ========== idx id obj type ref nrm lev adr link --- 29 TestDuplicate program 0 0 1 0 0 28 30 x variable 1 -1 1 0 5 29 31 y variable 1 -1 1 0 6 30 ========== BLOCK TABLE (BTAB) ========== idx last lpar psze vsze --- 0 31 0 0 2 ========== ARRAY TABLE (ATAB) ========== idx xtyp etyp eref low high elsz size --- ========== DECORATED AST ========== ProgramNode(name: 'TestDuplicate') | +-- Declarations | | | +-- VarDecl(name: 'x', type: 'integer') | \-- VarDecl(name: 'y', type: 'integer') | \-- Block | +-- Assign(target: Var('x'), value: Num(5)) | \-- Assign(target: Var('y'), value: BinOp(op: '+', left: Var('x'), right: Num(10))) Symbol table saved to ../test/output/symbol-table.txt Decorated AST saved to ../test/output/decorated-ast.txt

3.5 Kasus 5

Input	program StringTest; konstanta MSG = 'Hello';

	variabel name: larik [1..20] dari char; ch: char; mulai ch := 'A'; writeln(MSG) selesai.
Output	========== MILESTONE 3: SEMANTIC ANALYSIS ========== Performing semantic analysis... Semantic analysis failed: semantic analysis failed with 1 error(s) Semantic errors: 1. Type mismatch in assignment: cannot assign void to char ========== SYMBOL TABLE ========== ========== SYMBOL TABLE (TAB) ========== idx id obj type ref nrm lev adr link -- 29 StringTest program 0 0 1 0 0 28 30 MSG constant 3 -1 1 0 0 29 31 name variable 5 0 1 0 5 30 32 ch variable 3 -1 1 0 25 31 ========== BLOCK TABLE (BTAB) ========== idx last lpar psze vsze --------------------------------------- 0 32 0 0 21 ========== ARRAY TABLE (ATAB) ========== idx xtyp etyp eref low high elsz size --- 0 0 3 -1 1 20 1 20 ========== DECORATED AST ========== ProgramNode(name: 'StringTest') | +-- Declarations | | | +-- ConstDecl(name: 'MSG', value: 0, type: 'char') | +-- VarDecl(name: 'name', type: 'array') | \-- VarDecl(name: 'ch', type: 'char') | \-- Block | +-- Assign(target: Var('ch'), value: Char('A')) | \-- ProcedureCall(name: 'writeln', args: [Var('MSG')]) Symbol table saved to ../test/output/symbol-table.txt Decorated AST saved to ../test/output/decorated-ast.txt

3.6 Kasus 6

Input	program Complex;

```
konstanta
  MAX = 100;

variabel
  a, b, c: integer;
  result: integer;

mulai
  a := 10;
  b := 20;
  c := 30;
  result := a + b * c;
  writeln('A: ', a);
  writeln('B: ', b);
  writeln('C: ', c);
  writeln('Result: ', result);
selesai.
```

Output

```
========== MILESTONE 3: SEMANTIC ANALYSIS ==========
Performing semantic analysis...
Semantic analysis completed successfully

========== SYMBOL TABLE ==========

========== SYMBOL TABLE (TAB) ==========
idx   id                  obj          type   ref    nrm    lev  adr    link
------------------------------------------------------------------------------
29    Complex             program      0      0      1      0    0      28
30    MAX                 constant     1      -1     1      0    100    29
31    a                   variable     1      -1     1      0    5      30
32    b                   variable     1      -1     1      0    6      31
33    c                   variable     1      -1     1      0    7      32
34    result              variable     1      -1     1      0    8      33

========== BLOCK TABLE (BTAB) ==========
idx   last   lpar   psze   vsze
---------------------------------------
0     34     0      0      4

========== ARRAY TABLE (ATAB) ==========
idx   xtyp   etyp   eref   low    high   elsz   size
-----------------------------------------------------------------
```

	`` ========== DECORATED AST ========== ProgramNode(name: 'Complex') | +-- Declarations | | | +-- ConstDecl(name: 'MAX', value: 100, type: 'integer') | +-- VarDecl(name: 'a', type: 'integer') | +-- VarDecl(name: 'b', type: 'integer') | +-- VarDecl(name: 'c', type: 'integer') | \-- VarDecl(name: 'result', type: 'integer') | \-- Block | +-- Assign(target: Var('a'), value: Num(10)) | +-- Assign(target: Var('b'), value: Num(20)) | +-- Assign(target: Var('c'), value: Num(30)) | +-- Assign(target: Var('result'), value: BinOp(op: '+', left: Var('a'), right: BinOp(op: '*', left: Var('b'), right: Var('c')))) | +-- ProcedureCall(name: 'writeln', args: [String('A: '), Var('a')]) | +-- ProcedureCall(name: 'writeln', args: [String('B: '), Var('b')]) | +-- ProcedureCall(name: 'writeln', args: [String('C: '), Var('c')]) | \-- ProcedureCall(name: 'writeln', args: [String('Result: '), Var('result')]) Symbol table saved to ../test/output/symbol-table.txt Decorated AST saved to ../test/output/decorated-ast.txt ```

3.7 Kasus 7

Input	program TestLocalVar; variabel x: integer; fungsi hitung(a: integer; b: integer): integer; variabel hasil: integer; temp: integer; mulai hasil := a + b; temp := hasil * 2; hitung := temp selesai; mulai x := hitung(5, 10); writeln(x) selesai.

Output

```
========== MILESTONE 3: SEMANTIC ANALYSIS ==========
Performing semantic analysis...
Semantic analysis completed successfully

========== SYMBOL TABLE ==========

========== SYMBOL TABLE (TAB) ==========
idx   id                obj         type   ref    nrm    lev  adr    link
------------------------------------------------------------------------------
29    TestLocalVar      program     0      0      1      0    0      28
30    x                 variable    1      -1     1      0    5      29
31    a                 variable    1      -1     1      1    5      -1
32    b                 variable    1      -1     1      1    6      31
33    hitung            function    1      1      1      0    0      30
34    hitung            variable    1      -1     1      1    0      32
35    hasil             variable    1      -1     1      1    7      34
36    temp              variable    1      -1     1      1    8      35

========== BLOCK TABLE (BTAB) ==========
idx   last   lpar   psze   vsze
---------------------------------------
0     33     0      0      1
1     36     32     2      2

========== ARRAY TABLE (ATAB) ==========
idx   xtyp   etyp   eref   low    high   elsz   size
---------------------------------------------------------------
```

```
========== DECORATED AST ==========
ProgramNode(name: 'TestLocalVar')
|
+-- Declarations
|   |
|   +-- VarDecl(name: 'x', type: 'integer')
|   \-- FunctionDecl(name: 'hitung', return_type: 'integer')
|
\-- Block
    |
    +-- Assign(target: Var('x'), value: hitung(Num(5), Num(10)))
    |
    \-- ProcedureCall(name: 'writeln',
                      args: [Var('x')])
Symbol table saved to ../test/output/symbol-table.txt
Decorated AST saved to ../test/output/decorated-ast.txt
```

BAB IV

KESIMPULAN DAN SARAN

4.1 Kesimpulan

Berdasarkan hasil implementasi yang telah diperoleh, dapat dinyatakan bahwa implementasi telah berhasil dengan pendekatan algoritma *l-attributed grammar*. Pendekatan dari algoritma ini memiliki beberapa kekurangan dan kelebihan, yakni algoritma ini memiliki kelebihan berupa penyelesaian yang akan selalu mendapatkan terminal / mencapai akhir dari komputasi dengan identifikasi terhadap aspek *scope* dan *type* sehingga dapat dilakukan pengujian terhadap tipe dengan berbagai uji kasus, sementara itu kekurangan dari algoritma ini adalah pendekatan yang menggunakan algoritma DFS akan memiliki kompleksitas yang cenderung mahal karena membutuhkan waktu yang cukup besar apabila terdapat kompleksitas dan fitur bahasa baru yang akan diterapkan selanjutnya dan dengan adanya tambahan *production rule* (*semantic rule*) yang digunakan, terdapat peningkatan waktu dalam waktu eksekusi kode.

Milestone 3 mengimplementasikan *semantic analyzer* untuk bahasa Pascal-S menggunakan automata Context Free Grammar (CFG) dengan pendekatan semantic menggunakan tabel *tab, atab,* dan *btab*. Program mampu membaca file kode sumber dan menghasilkan *Decorated Abstract Syntax Tree* (AST) sesuai spesifikasi. Tahapan ini menjadi fondasi untuk milestone selanjutnya dalam pembuatan *compiler*, yakni *intermediate code generator*.

4.2 Saran

1. Melakukan perancangan dengan lebih baik sehingga tidak melakukan banyak refactor code.

REFERENSI

- https://www.geeksforgeeks.org/compiler-design/introduction-to-compilers/
- https://www.geeksforgeeks.org/compiler-design/semantic-analysis-in-compiler-design/
- https://people.montefiore.uliege.be/geurts/Cours/compil/2015/04-semantic-2015-2016.pdf
- http://pascal.hansotten.com/uploads/pascals/PASCAL-S%20A%20subset%20and%20its%20Implementation%20012.pdf
- https://www.tutorialspoint.com/compiler_design/compiler_design_semantic_analysis.htm
- https://www.tutorialspoint.com/compiler_design/compiler_design_symbol_table.htm
- Aho, et al. (2007). Compilers : Principles, Techniques, & Tools. New York : Pearson Education Inc.

LAMPIRAN

1. Tauran Repository Github
   https://github.com/BrianHadianSTEI23/TMP-Tubes-IF2224

2. Tautan Release Repository Github
   https://github.com/BrianHadianSTEI23/TMP-Tubes-IF2224/tree/v0.3.1

3. Pembagian Tugas

Nama	NIM	Tugas
Bertha Soliany Frandi	13523026	Program (25%)
Brian Albar Hadian	13523048	Laporan, Testing (25%)
Muhammad Izzat Jundy	13523092	Laporan, Testing (25%)
Michael Alexander Angkawijaya	13523102	Program, Laporan (25%)