LAPORAN MILESTONE 3: SEMANTIC ANALYSIS

IF2224 Teori Bahasa Formal dan Automata

Disusun oleh:

Kelompok JYP, K-02

Noumisyifa Nabila Nareswari	13523058
M. Ghifary Komara Putra	13523066
Sabilul Huda	13523072
Diyah Susan Nugrahani	13523080
Henry Filberto Shinelo	13523108

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2025

DAFTAR ISI

A. LANDASAN TEORI

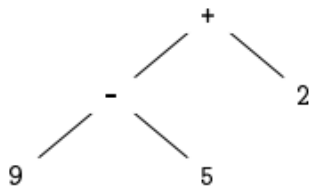
1. *Semantic Analysis Phase*

Semantic Analysis merupakan tahap ketiga dalam proses kerja sebuah compiler, setelah tahap *lexical analysis* dan *syntax analysis*. Pada tahap ini, *semantic analyzer* memanfaatkan syntax tree yang dihasilkan oleh *parser* serta informasi yang terdapat pada *symbol table* untuk memeriksa konsistensi semantik dari kode.

Fase *semantic analysis* memeriksa apakah konstruksi program memiliki makna yang benar secara semantik. Pemeriksaan ini meliputi validasi tipe data, kesesuaian deklarasi dan penggunaan variabel atau fungsi, pengecekan kompatibilitas operasi, pengecekan *scope*, serta aturan-aturan semantik lain yang tidak dapat diverifikasi oleh parser. Secara umum, hasil dari tahap ini adalah program yang sudah tervalidasi secara keseluruhan dan *syntax tree* yang telah diperkaya dengan informasi semantik yang akan digunakan oleh tahap berikutnya, yaitu *intermediate code generation*. Secara spesifik, pada tugas ini, hasil dari fase ini adalah *symbol table* dan *decorated abstract syntax tree*.

2. *Abstract Syntax Tree*

Abstract Syntax Tree (AST) berfungsi sebagai representasi hierarkis dari struktur program sumber dengan menekankan aspek-aspek sintaksis yang relevan untuk analisis lebih lanjut. Dalam AST untuk suatu ekspresi, setiap *interior node* merepresentasikan sebuah operator, sedangkan *children* dari node tersebut merepresentasikan operan-operannya. Dengan pendekatan ini, AST menghilangkan rincian *surface syntax* yang tidak penting seperti tanda kurung dan simbol tertentu lainnya, sehingga menghasilkan bentuk representasi yang lebih sederhana dan lebih mudah diproses oleh tahap *compiler* selanjutnya.



Gambar 1. Contoh AST untuk operasi 9-5+2

Sumber: Aho et al., 1986

AST umumnya dibangun selama fase *syntax analysis*, namun dalam implementasi tugas besar ini diimplementasi di fase jembatan antara fase *syntax analysis* dan sebelum masuk ke fase *semantic analysis*. Struktur data ini menjadi landasan analisis lanjutan pada tahap selanjutnya seperti *semantic analysis*, *intermediate code generation*, dan *optimization*. Karena sifatnya yang terstruktur dan bebas dari ambiguitas sintaks, AST memungkinkan compiler memahami hubungan semantik antar elemen program secara lebih jelas.

3. Symbol Table

Symbol table adalah struktur data yang digunakan oleh *compiler* untuk menyimpan informasi penting mengenai identitas *identifier* yang muncul dalam *source code*, seperti variabel, fungsi, parameter, tipe data, serta informasi atribut lain yang diperlukan selama proses kompilasi. *Symbol table* berfungsi sebagai pusat informasi yang dapat diakses oleh berbagai fase compiler untuk memastikan bahwa setiap penggunaan *identifier* konsisten secara semantik serta sesuai dengan aturan bahasa.

Secara umum, symbol table memungkinkan compiler untuk:

- Mencatat deklarasi *identifier*, termasuk tipe, lokasi memori, dan atribut terkait.
- Mendeteksi error semantik, seperti penggunaan variabel yang belum dideklarasikan atau re-deklarasi yang tidak valid.
- Mendukung *scoping* dan *block structure*, dengan mengelola ruang lingkup yang berbeda dalam program.

Pada tugas ini, terdapat tiga *symbol table* yang kami implementasikan:

1. tab: digunakan untuk menyimpan informasi mengenai identifier termasuk konstanta, variabel, prosedur, atau fungsi.
2. btab: digunakan untuk menyimpan informasi blok prosedur dan definisi tipe rekord.
3. atab: digunakan untuk menyimpan informasi array seperti tipe index, batasan index, tipe elemen, dan ukuran array.

Berikut merupakan daftar atribut dari symbol table

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

atab	
atribut	**deskripsi**
arrays	Indeks entri array
xtyp	Tipe indeks array (misalnya integer). Berupa kode tipe dari tabel tab.
etyp	Tipe elemen array (misalnya integer). Berupa kode tipe dari tabel tab.
eref	Pointer/indeks ke detail tipe elemen jika elemen adalah tipe komposit (misalnya array dalam array, atau record).

	Mengarah ke atab atau btab.
low	Batas bawah indeks array (misalnya 1 pada array[1..10] atau 0 pada array[0..15]).
high	Batas atas indeks array.
elsz	Ukuran satu elemen array (dalam byte/unit memori).
size	Total ukuran array

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

4. Decorated Abstract Syntax Tree

Decorated Abstract Syntax Tree (Decorated AST) merupakan pengembangan dari AST yang digunakan setelah proses *semantic analysis* selesai. Pada tahap ini, AST yang sebelumnya hanya merepresentasikan struktur sintaksis program diperluas dengan informasi-informasi semantik yang diperlukan agar tahap-tahap selanjutnya memiliki konteks yang cukup.Setiap node dalam *Decorated* AST setidaknya memiliki tipe ekspresi dan atribut-atribut tambahan lain yang relevan, seperti informasi deklarasi variabel, parameter fungsi, maupun properti khusus.

B. PERANCANGAN DAN IMPLEMENTASI

1. Deskripsi Umum

Semantic analyzer merupakan komponen ketiga dalam pengembangan *compiler* setelah *lexer* dan *parser*. Secara umum, komponen ini berperan dalam konstruksi *Abstract Syntax Tree* (AST), penyusunan *symbol table*, pengecekan tipe/semantik, serta melakukan dekorasi terhadap AST. Proses dimulai dengan mengembangkan AST berdasarkan *syntax tree* yang diberikan oleh *parser*. Dengan adanya AST, *compiler* dapat mengetahui struktur dari program yang sedang dikompilasi.

Setelah AST berhasil terbentuk, akan dilakukan pendataan *symbol table*, yaitu menelusuri seluruh isi AST untuk mengisi informasi terkait *identifier*, *array*, *block*, dan sebagainya ke dalam *symbol table* terkait. Dalam tahap ini, analisis terhadap semantik dilakukan pula, untuk menangani kasus-kasus seperti kesalahan tipe, deklarasi variabel berulang kali, dan lain-lain.

Setelah AST telah ditelusuri dan seluruh *symbol table* telah terisi, *compiler* akan melakukan dekorasi terhadap AST. Pada tahap ini, AST akan diberikan informasi relevan seperti lokasinya dalam tab, *scope* tempat ia berada, dan lain-lain. AST yang telah didekorasi akan dimanfaatkan untuk proses kompilasi berikutnya.

2. Abstract Syntax Tree

Struktur data utama dalam implementasi AST adalah ASTNode. *Node* ini akan jadi *abstract base type* untuk *node* lainnya yang dibuat untuk merepresentasikan berbagai kemungkinan semantik. Berikut rincian struktur data ASTNode.

```
struct ASTNode {
    string nodeType;              // Jenis node
    string dataType;              // Hasil type checking
    int symbolIndex = -1;         // Indeks ke tabel simbol
    int scopeLevel = -1;          // Scope tempat node ini berada

    vector<ASTNode*> children;

    ASTNode(const string &type) : nodeType(type) {}
    virtual ~ASTNode() {
        for (auto child : children) {
            delete child;
        }
    }
```

```
};
```

- `nodeType` mereprentasikan jenis *node* tersebut, apakah *assignment*, *if*, dan lain-lainnya.
- `dataType` merepresentasikan anotasi tipe dari *semantic analysis*.
- `symbolIndex` merepresentasikan index ke tabel simbol
- `scopeLevel` merepresentasikan *scope* posisi kemunculan *node*
- `children` merepresentasikan relasi *parent-child* di AST

Berikut adalah struktur data berbagai *node* yang merepresentasikan berbagai kemungkinan semantik di bahasa Pascal-S

Node	Fungsi
`ProgramNode`	Mewakili program utama, menyimpan nama program, deklarasi & block
`DeclarationsNode`	Menyimpan list seluruh deklarasi
`VarDeclNode`	Deklarasi variabel (nama + tipe)
`ConstDeclNode`	Deklarasi konstanta (nama + nilai)
`TypeDeclNode`	Deklarasi tipe baru
`ArrayTypeNode`	Menyimpan range indeks larik & tipe elemen
`ArrayAccessNode`	Ekspresi akses indeks array
`NumberNode, RealNode`	Literals: 5, 3.14
`StringNode, CharNode`	'a', "hello"
`BoolNode`	benar, salah
`VarNode`	Akses variabel
`BinOpNode`	a + b, x < y
`UnaryOpNode`	-x, tidak p
`AssignNode`	*Assignment*
`IfNode`	Percabangan dengan opsional else

`WhileNode`	Loop selama
`ForNode`	Perulangan naik / turun-ke
`ProcedureCallNode`	Pemanggilan prosedur
`BlockNode`	Berisi list deklarasi serta list statement

Pertama, `main.cpp` akan panggil `ASTMain` yang menerima *root node* dari fase *parsing*. Fungsi ini akan memanggil fungsi `buildAST`, *entry point* dari algoritma pembangunan AST, dan fungsi `printAST` untuk *print* struktur AST.

```
ASTNode* ASTMain(ParseNode* root) {
    ASTNode* ast = buildAST(root);

    if (!ast) {
        cerr << "Error: Failed to build AST\n";
        return nullptr;
    }

    cout << "\n========== AST Structure ==========\n";
    printAST(ast);

    return ast;
}
```

Selanjutnya, fungsi `buildAST` akan menerima *root node* dari `ParseNode` dan *pass* ke fungsi `convert` yang berisi algoritma inti pembangunan AST.

```
ASTNode* buildAST(ParseNode* root) {
    if (!root) {
        cerr << "Error: Parse tree root is null\n";
        return nullptr;
    }
    return convert(root);
}
```

Selanjutnya, fungsi convert akan membangun AST. Fungsi ini menerima *root node* dari ParseNode dan akan *traverse through the parse tree*, membaca label ParseNode dan memutuskan node AST apa yang harus dibuat, secara rekursif. Kurang lebih yang terjadi di dalam fungsi ini adalah sebagai berikut

```
ASTNode* convert(ParseNode* p) {

    if (has(p, "<assignment-statement>")) {
```

```
        ASTNode* left = convert(p->children[0]);
        ASTNode* right = convert(p->children[2]);
        return new AssignNode(left, right);
    }

    if (has(p, "<if-statement>")) {
        ASTNode* condition = convert(p->children[1]);
        ASTNode* thenStmt = convert(p->children[3]);
        ASTNode* ifNode = new IfNode(condition, thenStmt);

        if (ada_else) {
            ASTNode* elseStmt = convert(p->children[idxElse]);
            ifNode->children.push_back(elseStmt);
        }

        return ifNode;
    }

    if (has(p, "<while-statement>")) {
        ASTNode* condition = convert(p->children[1]);
        ASTNode* body = convert(p->children[3]);
        return new WhileNode(condition, body);
    }

    // dst. untuk node lainnya...


    // fallback
    if (!p->children.empty()) {
        return convert(p->children[0]); // rekursi ke child
    }

    return nullptr; // kasus base
}
```

- Algoritma akan mengenali suatu *rule grammar* dengan pencocokan label.
- Buat `NodeAST` yang sesuai.
- Secara terus-menerus mengulangi hal yang sama sampai *go through the whole parse tree* hingga selesai secara rekursif.
- *Return pointer* ke *root node* dari `NodeAST` yang sudah terbentuk.

Terdapat beberapa fungsi *helper* yang digunakan untuk kasus-kasus tertentu sekaligus untuk *print* AST yang dibentuk.

Helper	Fungsi
`buildExpression()`	Bangun ekspresi + precedence relasional
`buildSimpleExpression()`	Tangani operator + dan -

`buildTerm()`	Tangani operator *, /, mod
`buildFactor()`	Literals, Var, array access, parenthesis
`buildArrayTypeNode()`	Membentuk node tipe array [a..b] of T
`void printAST(ASTNode* node, const string& prefix, bool isLast)`	Print AST

3. *Symbol Table*

Secara umum, *symbol table* merupakan struktur data yang digunakan oleh *compiler* untuk menyimpan informasi mengenai *identifier* yang dideklarasikan dalam suatu program, misalnya variabel, konstanta, fungsi, dan sebagainya. Ketika *compiler* menemukan suatu *identifier*, ia harus mengetahui *identifier* tersebut merujuk pada hal dan informasi apa. Dengan adanya *symbol table*, informasi tersebut dapat tersimpan dan *compiler* dapat menangani kasus seperti memastikan variabel dideklarasikan sebelum digunakan, tidak ada deklarasi ganda terhadap suatu *identifier*, dan penanganan kasus terkait tipe dan *scope* lainnya. Dalam implementasi ini, *symbol table* terdefinisi ke dalam tiga struktur data, yaitu *tab, atab,* dan *btab*.

a. tab (*symbol table* umum)

Tab merupakan *symbol table* yang digunakan untuk menyimpan informasi mengenai *identifier*. Hal ini mencakup konstanta, variabel, prosedur, maupun fungsi. Hal ini berarti bahwa, dalam proses kompilasi, *compiler* (melalui tahap *semantic analyisis*) akan memasukkan entri baru ke dalam *tab* untuk setiap deklarasi yang ditemukan dalam program. Mengingat deklarasi mencakup deklarasi *array* dan bergantung pada *scope* program, *symbol table* ini terkait dengan *atab* maupun *btab* (dipaparkan pada poin b dan c). Entri pada tab direpresentasikan dalam struktur data dan spesifikasi sebagai berikut.

```
struct TabEntry {
    string name;
    int link;
    int obj;
    int type;
    int ref;
    int nrm;
    int lev;
    int adr;
    bool initialized;
```

```
};
```

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

Tab akan diinisialisasi pada awal proses *semantic analysis*. Dalam tahap ini, *tab* akan diisi dengan 29 entri yang masing-masing mewakili *reserved word* yang tersedia.

```
void initializeTab(){
    if(!tab.empty()) return;
    tab.reserve(256);

    vector<string> keywords = {
        "dan",          // AND
        "larik",        // ARRAY
        "mulai",        // BEGIN
        "kasus",        // CASE
        "konstanta",    // CONST
        "bagi",         // DIV
        "turun-ke",     // DOWNTO
```

```
        "lakukan",      // DO
        "selain-itu",   // ELSE
        "selesai",      // END
        "untuk",        // FOR
        "fungsi",       // FUNCTION
        "jika",         // IF
        "mod",          // MOD
        "tidak",        // NOT
        "dari",         // OF
        "atau",         // OR
        "prosedur",     // PROCEDURE
        "program",      // PROGRAM
        "rekaman",      // RECORD
        "ulangi",       // REPEAT
        "string",       // STRING
        "maka",         // THEN
        "ke",           // TO
        "tipe",         // TYPE
        "sampai",       // UNTIL
        "variabel",     // VAR
        "selama",       // WHILE
        "packed"        // PACKED
    };

    for (const string& kw : keywords) {
        TabEntry e;
        e.name = kw;
        e.link = 0;
        e.obj = 0;
        e.type = 0;
        e.ref = 0;
        e.nrm = 1;
        e.lev = 0;
        e.adr = 0;
        e.initialized = false;
        tab.push_back(e);
    }

    {
        TabEntry e;
        e.name = "false";
        e.link = 0;
        e.obj = OBJ_CONSTANT;
        e.type = 3;
        e.ref = 0;
        e.nrm = 1; e.lev = 0; e.adr = 0;
        tab.push_back(e);
    }

    {
        TabEntry e;
        e.name = "true";
        e.link = 0;
        e.obj = OBJ_CONSTANT;
        e.type = 3;
        e.ref = 1;
        e.nrm = 1; e.lev = 0; e.adr = 0;
        tab.push_back(e);
    }

    struct StdType { string name; int typeCode; int size; };
    vector<StdType> types = {
```

```
        {"integer", 1, 1},
        {"real",    2, 1},
        {"boolean", 3, 1},
        {"char",    4, 1}
    };

    for(const auto& t : types) {
        TabEntry e;
        e.name = t.name;
        e.link = 0;
        e.obj = OBJ_TYPE;
        e.type = t.typeCode;
        e.ref = t.size;
        e.nrm = 1; e.lev = 0; e.adr = 0;
        tab.push_back(e);
    }

    if (!btab.empty()) {
        btab[0].last = tab.size() - 1;
    }
}
```

Sebagai bentuk *error handling* terhadap *identifier* yang sudah dideklarasikan sebelumnya dalam *scope* yang sama, *tab* dapat memeriksa duplikasi entri melalui fungsi *isDuplicateInCurrentBlock()* di bawah ini.

```
bool isDuplicateInCurrentBlock(const string& name){
    if(display.empty()) return false;

    int curLevel = currentLevel;
    if(curLevel < 0 || curLevel >= (int)display.size()) return false;

    int curBlock = display[curLevel];
    if(curBlock < 0 || curBlock >= (int)btab.size()) return false;

    int idx = btab[curBlock].last;
    while(idx!=0){
        if(tab[idx].name == name) return true;
        idx = tab[idx].link;
    }
    return false;
}
```

Selebihnya, untuk mendata seluruh *identifier*, *tab* harus dapat mengetahui *scope/block* tempatnya berada saat ini, memasuki serta keluar dari *block* tersebut (dengan menyesuaikan *currentLevel* dan *btab*), menambahkan entri baru pada dirinya sendiri, serta melakukan *lookup* terhadap *identifier* secara lintas *scope* (dari *currentLevel* hingga 0, mengikuti *linked list* dari *block*).

```
int insertIdentifier(const string& name, int obj, int type, int ref,
int nrm, int adr){
```

```
    initializeTab();

    if(display.empty()){
        semanticError("insertIdentifier: no active display (no scope
opened).");
        return 0;
    }
    int curLevel = currentLevel;
    if(curLevel < 0 || curLevel >= (int)display.size()){
        semanticError("insertIdentifier: invalid current level.");
        return 0;
    }
    int curBlock = display[curLevel];

    // cek duplikat
    int head = btab[curBlock].last;
    int scan = head;
    while(scan !=0){
        if(tab[scan].name == name){
            semanticError("Redeclaration of identifier '" + name + "'
in the same block.");
            return scan;
        }
        scan = tab[scan].link;
    }

    // buat entry baru
    TabEntry e;
    e.name = name;
    e.link = head;
    e.obj = obj;
    e.type = type;
    e.ref = ref;
    e.nrm = nrm;
    e.lev = currentLevel;
    e.adr = adr;
    e.initialized = (obj == OBJ_CONSTANT);

    int newIndex = (int)tab.size();
    tab.push_back(e);

    btab[curBlock].last = newIndex;
    return newIndex;
}

int lookupIdentifier(const string& name){
    for (int lv = currentLevel; lv >= 0; --lv) {
        if (lv >= (int)display.size()) continue;
        int block = display[lv];
        if (block < 0 || block >= (int)btab.size()) continue;

        int idx = btab[block].last;
        while (idx > 0) {
            if (tab[idx].name == name) return idx;
            idx = tab[idx].link;
        }
    }

    for (size_t i = 0; i < tab.size(); i++) {
        if (tab[i].name == name) {
            return i;
        }
```

```
    }

    if (name == "writeln" || name == "write" || name == "readln" ||
name == "read") {
        for(size_t i = 29; i < tab.size(); i++){
            if(tab[i].name == name && tab[i].nrm == 1 && tab[i].obj ==
OBJ_PROCEDURE){
                return i;
            }
        }
        TabEntry e;
        e.name = name;
        e.link = 0;
        e.obj = OBJ_PROCEDURE;
        e.type = 0;
        e.ref = 0;
        e.nrm = 1;
        e.lev = 0;
        e.adr = 0;
        e.initialized = false;
        tab.push_back(e);
        return tab.size() - 1;
    }

    return 0; // Tidak ketemu
}

void openScope(){
    int newBlock = createNewBlock();
    enterBlock(newBlock);
}

void closeScope(){
    exitBlock();
}
```

b. atab (*symbol table* untuk *array*)

Array Table atau *atab* merupakan struktur pendukung yang digunakan khusus untuk menyimpan informasi mengenai tipe data array. *Table* utama *tab* didesain generik untuk menyimpan informasi identifier. *Atab* diperlukan karena array memiliki atribut spesifik yang perlu disimpan secara khusus. Setiap entri pada *atab* merepresentasikan satu tipe array unik dengan struktur data seperti AtabEntry di bawah.

```
struct AtabEntry {
    int xtyp;
    int etyp;
    int eref;
    int low;
    int high;
    int elsz;
```

```
    int size;
};
```

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

Proses pencatatan struktur data array ditangani secara spesifik oleh fungsi *processArrayDeclaration*. Mekanisme ini dimulai ketika parser mendeteksi adanya deklarasi tipe array, yang kemudian memicu ekstraksi nilai batas bawah dan batas atas dari Abstract Syntax Tree. Data tersebut langsung divalidasi melalui pemeriksaan semantik untuk memastikan konsistensi logika rentang indeks. Setelah validasi berhasil, sistem menghitung total kebutuhan alokasi memori berdasarkan ukuran tipe elemen yang dideklarasikan. Seluruh atribut array yang telah diolah ini kemudian disimpan ke dalam *atab*. Indeks lokasi penyimpanannya dikembalikan untuk dicatat sebagai referensi pada *tab*.

```
int processArrayDeclaration(ArrayTypeNode* arrayTypeNode) {
    if (!arrayTypeNode) {
        semanticError("Array type node is null");
        return -1;
    }
```

```
    int startType = determineIndexType(arrayTypeNode->rangeStart);
    int endType = determineIndexType(arrayTypeNode->rangeEnd);

    if (startType == 0 || endType == 0) {
        semanticError("Invalid index type. Array index must be Integer,
Char, or Boolean.");
        return -1;
    }

    if (startType != endType) {
        semanticError("Array range type mismatch. Start is type " +
to_string(startType) +
                      ", End is type " + to_string(endType));
        return -1;
    }

    AtabEntry entry;

    entry.xtyp = startType;

    entry.low = getOrdinalValue(arrayTypeNode->rangeStart);
    entry.high = getOrdinalValue(arrayTypeNode->rangeEnd);

    if (entry.low > entry.high) {
        semanticError("Array lower bound (" + to_string(entry.low) +
                      ") > upper bound (" + to_string(entry.high) +
")");
        return -1;
    }

    entry.etyp = getTypeCode(arrayTypeNode->elementType);
    if (entry.etyp == 0) {
        semanticError("Invalid array element type: " +
arrayTypeNode->elementType);
        return -1;
    }

    entry.elsz = getTypeSize(entry.etyp);
    entry.size = (entry.high - entry.low + 1) * entry.elsz;
    entry.eref = 0;

    atab.push_back(entry);

    return atab.size() - 1;
}
```

c. btab (*symbol table* untuk *block*/*scoping*)

Block Table atau btab merupakan struktur pendukung yang digunakan khusus untuk menyimpan informasi mengenai blok program, prosedur, fungsi, serta definisi tipe rekord. Table utama tab didesain generik untuk menyimpan informasi identifier, namun memerlukan struktur tambahan untuk mengelola informasi blok dan scope. Btab diperlukan karena setiap blok memiliki atribut spesifik terkait parameter dan variabel lokal yang perlu disimpan secara khusus. Setiap entri pada btab merepresentasikan satu blok program dengan struktur data seperti BtabEntry di bawah.

```
struct BtabEntry {
    int last;
    int lpar;
    int psze;
    int vsze;
};
```

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

Proses pengelolaan blok ditangani melalui beberapa fungsi utama. Mekanisme ini dimulai dengan inisialisasi btab menggunakan `initializeBtab()` yang membuat blok global (indeks 0) sebagai dasar hierarki scope program. Setiap kali compiler memasuki deklarasi prosedur atau fungsi, sistem membuat entri blok baru melalui `createNewBlock()` dan mengaktifkannya dengan `enterBlock()`. Display stack digunakan untuk melacak blok aktif pada setiap lexical level, memungkinkan lookup identifier yang efisien lintas scope. Saat keluar dari blok, `exitBlock()` menurunkan level dan mengembalikan konteks ke blok parent. Selama traversal AST, setiap deklarasi variabel atau parameter akan memperbarui atribut `last`, `lpar`, `psze`, dan `vsze` pada entri btab yang sesuai.

```
void initializeBtab() {
    btab.clear();
    display.clear();
```

```
    // Buat block global (indeks 0)
    BtabEntry globalBlock;
    globalBlock.last = 0;  // Akan diupdate saat ada deklarasi
    globalBlock.lpar = 0;  // Global block tidak punya parameter
    globalBlock.psze = 0;  // Tidak ada parameter
    globalBlock.vsze = 0;  // Akan diupdate saat ada variabel global

    btab.push_back(globalBlock);

    // Inisialisasi display untuk level 0
    display.push_back(0);  // display[0] = block index 0 (global)
    currentLevel = 0;
}

int createNewBlock() {
    BtabEntry newBlock;
    newBlock.last = 0;
    newBlock.lpar = 0;
    newBlock.psze = 0;
    newBlock.vsze = 0;

    btab.push_back(newBlock);
    return btab.size() - 1;
}

void enterBlock(int blockIndex) {
    currentLevel++;

    if ((int)display.size() <= currentLevel) {
        display.resize(currentLevel + 1);
    }

    display[currentLevel] = blockIndex;
}

void exitBlock() {
    if (currentLevel > 0) {
        currentLevel--;
    }
}

int getCurrentBlock() {
    if (currentLevel >= 0 && currentLevel < (int)display.size()) {
        return display[currentLevel];
    }
    return 0; // Default ke global block
}
```

Fungsi-fungsi pembaruan atribut btab dipanggil selama proses semantic analysis untuk mencatat informasi blok secara akurat:

```
void updateBlockLast(int blockIndex, int lastIdentifier) {
    if (blockIndex >= 0 && blockIndex < (int)btab.size()) {
        btab[blockIndex].last = lastIdentifier;
    }
}

void incrementBlockVsze(int blockIndex, int additionalSize) {
```

```
    if (blockIndex >= 0 && blockIndex < (int)btab.size()) {
        btab[blockIndex].vsze += additionalSize;
    }
}

void incrementBlockPsze(int blockIndex, int additionalSize) {
    if (blockIndex >= 0 && blockIndex < (int)btab.size()) {
        btab[blockIndex].psze += additionalSize;
    }
}
```

4. Decorated Abstract Syntax Tree

Decorated AST adalah struktur Abstract Syntax Tree yang telah ditambah dengan informasi semantik hasil dari analisis semantik. Setiap node dalam AST dianotasi dengan metadata tambahan yang mencakup informasi tipe data, scope, dan referensi ke tabel simbol.

Pertama, semantic analyzer akan memanggil fungsi `analyze`.Fungsi ini akan menerima root node dari AST yang telah dibangun (program). Kemudian, proses analisis semantik dilakukan mulai dari root tersebut.

```
bool SemanticAnalyzer::analyze(ASTNode* root) {
    if (!root) {
        cerr << "Error: AST root is null\n";
        return false;
    }

    if (root->nodeType != "Program") {
        cerr << "Error: Root must be Program\n";
        return false;
    }

    try {
        visitProgram(static_cast<ProgramNode*>(root));
        return !hasErrors;
    } catch (const exception& e) {
        cerr << "Semantic analysis failed: " << e.what() << endl;
        return false;
    }
}
```

Analisis semantik melakukan traversal melalui AST menggunakan fungsi visit, dimana setiap jenis node dikunjungi dan didekorasi:

```
void SemanticAnalyzer::visitProgram(ProgramNode* node) {
    // Dekorasi node program
    node->symbolIndex = progIdx;    // Index ke symbol table
    node->scopeLevel = 0;           // Scope level 0 (global)
    node->dataType = "program";     // Tipe data program

    // Rekursi ke children
    if (node->declarations) visitDeclarations(node->declarations);
    if (node->block) visitBlock(node->block);
}


void SemanticAnalyzer::visitDeclarations(DeclarationsNode* node) {
    if (!node) return;

    node->scopeLevel = currentLevel;

    for (ASTNode* decl : node->declarations) {
        if (!decl) continue;

        if (decl->nodeType == "VarDecl") {
            visitVarDecl(static_cast<VarDeclNode*>(decl));
        }
        else if (decl->nodeType == "ConstDecl") {
            visitConstDecl(static_cast<ConstDeclNode*>(decl));
        }
        else if (decl->nodeType == "TypeDecl") {
            visitTypeDecl(static_cast<TypeDeclNode*>(decl));
        }
        else if (decl->nodeType == "ProcedureDecl") {
            visitProcedureDecl(static_cast<ProcedureDeclNode*>(decl));
        }
        else if (decl->nodeType == "FunctionDecl") {
            visitFunctionDecl(static_cast<FunctionDeclNode*>(decl));
        }
    }
}

// dst. untuk visit node lainnya
```

Algoritma proses visit node.

- Algoritma akan mengenali jenis node dengan pencocokan nodeType.
- Panggil fungsi visit yang sesuai untuk node tersebut.
- Dekorasi node dengan informasi semantik
- Secara terus-menerus mengulangi proses visit untuk setiap child node, dan traverse seluruh AST hingga selesai secara rekursif.

Selama proses traversal, juga dilakukan pengisian pada symbol table sekaligus dilakukan analisis semantik.

```
void SemanticAnalyzer::visitAssign(AssignNode* node) {

// bagian yang relevan
...

// checking type mismatch
string targetType = inferType(node->target);
    string valueType = inferType(node->value);

    if (!isCompatibleType(targetType, valueType)) {
        semanticWarning("Type mismatch: " + targetType + " := " +
valueType);
    }

...
}



void SemanticAnalyzer::visitFunctionDecl(FunctionDeclNode* node) {

//bagian yang relevan
...

// mengisi symbol table dan decorate

int funcBlockIdx = createNewBlock();
    int retTypeCode = getTypeCode(node->returnType);

    int curBlock = getCurrentBlock();
    int previousId = btab[curBlock].last;

    TabEntry e;
    e.name = node->name;
    e.link = previousId;
    e.obj = OBJ_FUNCTION;
    e.type = retTypeCode;
    e.ref = funcBlockIdx;
    e.nrm = 1;
    e.lev = currentLevel;
    e.adr = 0;
    e.initialized = false;

    tab.push_back(e);
    int funcIdx = tab.size() - 1;

    btab[curBlock].last = funcIdx;

    node->symbolIndex = funcIdx;
    node->scopeLevel = currentLevel;
    node->dataType = node->returnType;
...
}
```

Terdapat beberapa fungsi *helper* yang digunakan untuk analisis semantik sekaligus print decorated AST yang dibentuk.

Helper	Fungsi
`inferType(ASTNode* node)`	Memberikan type dari AST Node
`isCompatibleType(const string& type1, const string& type2)`	Mengecek kesesuaian tipe pada assignment
`getOperatorResultType(const string& op, const string& leftType, const string& rightType)`	Mendapatkan hasil operasi dua variabel
`getTypeName(int typeCode)`	Konversi typeCode ke typeName
`getObjectName(int objCode)`	Konversi objCode ke objName
`printDecoratedAST(ASTNode* node, const string& prefix, bool isLast)`	Print decorated AST

C. PENGUJIAN

Pengujian 1: tc1.pas

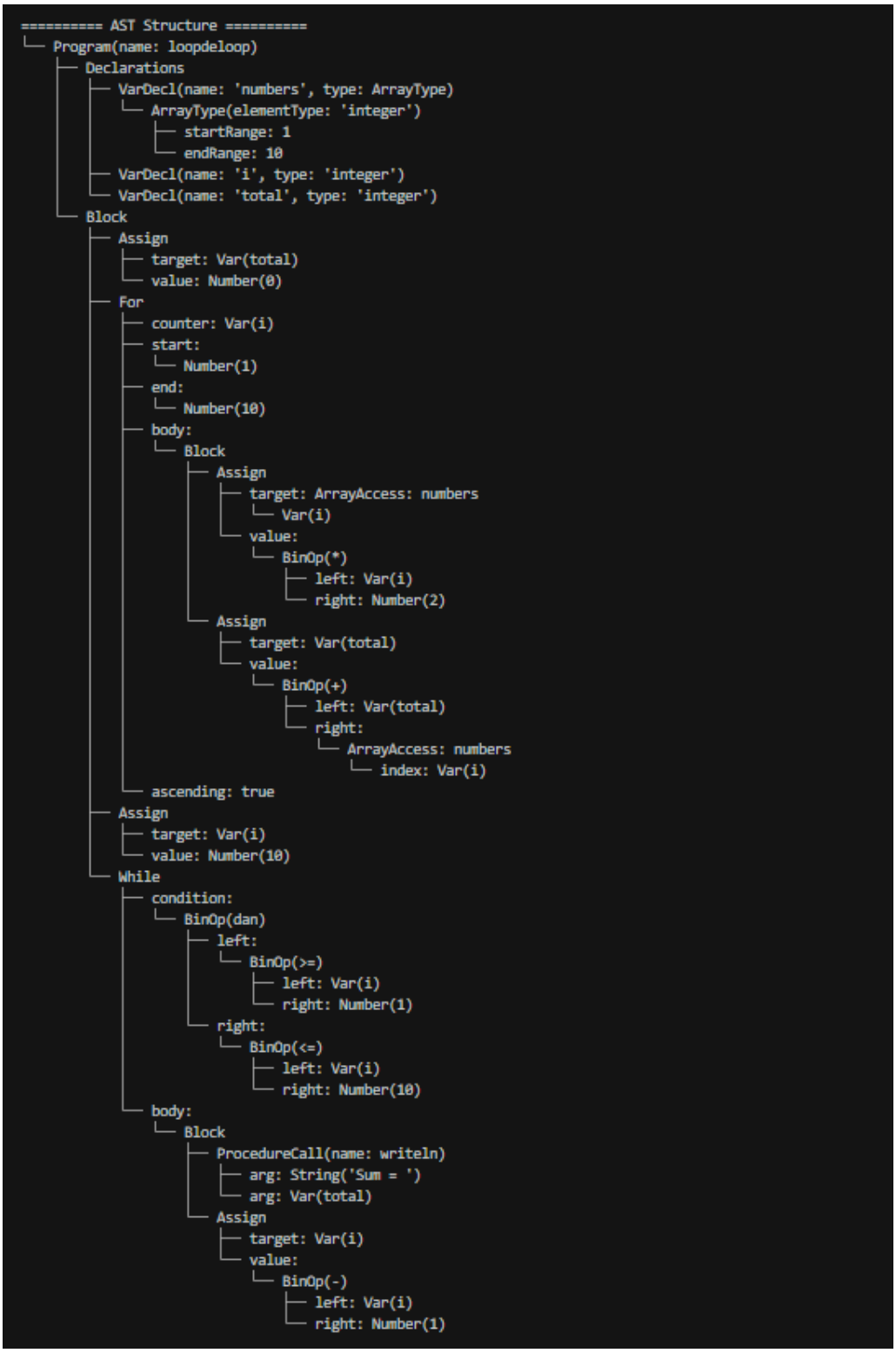
Test Case

```
program loopdeloop;
variabel
    numbers: larik[1 .. 10] dari integer;
    i, total: integer;
mulai
    total := 0;
    untuk i := 1 ke 10 lakukan
    mulai
        numbers[i] := i * 2;

        total := total + numbers[i];
    selesai;

    i := 10;
    selama (i >= 1) dan (i <= 10) lakukan
    mulai
        writeln('Sum = ', total);
        i := i - 1;
    selesai;
selesai.
```

Bukti Input

```
bill2247@bill2247-TECNO-MEGABOOK:~/TUBES/JYP-Tubes-IF2224$ make run ARGS=test/milestone-3/tc1.pas
```

Bukti Output AST



```
========== AST Structure ==========
└── Program(name: loopdeloop)
    ├── Declarations
    │   ├── VarDecl(name: 'numbers', type: ArrayType)
    │   │   └── ArrayType(elementType: 'integer')
    │   │       ├── startRange: 1
    │   │       └── endRange: 10
    │   ├── VarDecl(name: 'i', type: 'integer')
    │   └── VarDecl(name: 'total', type: 'integer')
    └── Block
        ├── Assign
        │   ├── target: Var(total)
        │   └── value: Number(0)
        ├── For
        │   ├── counter: Var(i)
        │   ├── start:
        │   │   └── Number(1)
        │   ├── end:
        │   │   └── Number(10)
        │   ├── body:
        │   │   └── Block
        │   │       ├── Assign
        │   │       │   ├── target: ArrayAccess: numbers
        │   │       │   │   └── Var(i)
        │   │       │   └── value:
        │   │       │       └── BinOp(*)
        │   │       │           ├── left: Var(i)
        │   │       │           └── right: Number(2)
        │   │       └── Assign
        │   │           ├── target: Var(total)
        │   │           └── value:
        │   │               └── BinOp(+)
        │   │                   ├── left: Var(total)
        │   │                   └── right:
        │   │                       └── ArrayAccess: numbers
        │   │                           └── index: Var(i)
        │   └── ascending: true
        ├── Assign
        │   ├── target: Var(i)
        │   └── value: Number(10)
        └── While
            ├── condition:
            │   └── BinOp(dan)
            │       ├── left:
            │       │   └── BinOp(>=)
            │       │       ├── left: Var(i)
            │       │       └── right: Number(1)
            │       └── right:
            │           └── BinOp(<=)
            │               ├── left: Var(i)
            │               └── right: Number(10)
            └── body:
                └── Block
                    ├── ProcedureCall(name: writeln)
                    │   ├── arg: String('Sum = ')
                    │   └── arg: Var(total)
                    └── Assign
                        ├── target: Var(i)
                        └── value:
                            └── BinOp(-)
                                ├── left: Var(i)
                                └── right: Number(1)
```

Bukti Output Symbol Table tab

```
================================ TAB (Symbol Table) ==================================
idx     id                      obj     type    ref     nrm     lev     adr     link
------------------------------------------------------------------------------------
0       dan                     0       0       0       1       0       0       0
1       larik                   0       0       0       1       0       0       0
2       mulai                   0       0       0       1       0       0       0
3       kasus                   0       0       0       1       0       0       0
4       konstanta               0       0       0       1       0       0       0
5       bagi                    0       0       0       1       0       0       0
6       turun-ke                0       0       0       1       0       0       0
7       lakukan                 0       0       0       1       0       0       0
8       selain-itu              0       0       0       1       0       0       0
9       selesai                 0       0       0       1       0       0       0
10      untuk                   0       0       0       1       0       0       0
11      fungsi                  0       0       0       1       0       0       0
12      jika                    0       0       0       1       0       0       0
13      mod                     0       0       0       1       0       0       0
14      tidak                   0       0       0       1       0       0       0
15      dari                    0       0       0       1       0       0       0
16      atau                    0       0       0       1       0       0       0
17      prosedur                0       0       0       1       0       0       0
18      program                 0       0       0       1       0       0       0
19      rekaman                 0       0       0       1       0       0       0
20      ulangi                  0       0       0       1       0       0       0
21      string                  0       0       0       1       0       0       0
22      maka                    0       0       0       1       0       0       0
23      ke                      0       0       0       1       0       0       0
24      tipe                    0       0       0       1       0       0       0
25      sampai                  0       0       0       1       0       0       0
26      variabel                0       0       0       1       0       0       0
27      selama                  0       0       0       1       0       0       0
28      packed                  0       0       0       1       0       0       0
29      false                   1       3       0       1       0       0       0
30      true                    1       3       1       1       0       0       0
31      integer                 3       1       1       1       0       0       0
32      real                    3       2       1       1       0       0       0
33      boolean                 3       3       1       1       0       0       0
34      char                    3       4       1       1       0       0       0
35      loopdeloop              0       0       0       1       0       0       0
36      numbers                 2       5       0       1       0       0       0
37      i                       2       1       0       1       0       10      36
38      total                   2       1       0       1       0       11      37
39      writeln                 4       0       0       1       0       0       0
====================================================================================
```

Bukti Output Symbol Table atab

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

Bukti Output Symbol Table btab

```
========== BTAB (Block Table) ==========
idx     last    lpar    psze    vsze
----------------------------------------
0       38      0       0       12
1       0       0       0       0
========================================
```

Bukti Output Decorated AST

```
========== Decorated AST ==========
└─ Program(name: 'loopdeloop') → tab_index:35, obj:program, type:'program', lev:0
    ├─ Declarations → type:void, lev:0
    │   ├─ VarDecl(name: 'numbers') → tab_index:36, obj:variable, type:'array', lev:0
    │   │   └─ ArrayType → type:void
    │   │       ├─ Number(1) → type:void
    │   │       └─ Number(10) → type:void
    │   ├─ VarDecl(name: 'i') → tab_index:37, obj:variable, type:'integer', lev:0
    │   └─ VarDecl(name: 'total') → tab_index:38, obj:variable, type:'integer', lev:0
    └─ Block → type:void, lev:1
        ├─ Assign → type:'integer', lev:1
        │   ├─ Var('total') → tab_index:38, obj:variable, type:'integer', lev:0
        │   └─ Number(0) → type:'integer', lev:1
        ├─ For → type:void, lev:1
        │   ├─ Var('i') → tab_index:37, obj:variable, type:'integer', lev:0
        │   ├─ Number(1) → type:'integer', lev:1
        │   ├─ Number(10) → type:'integer', lev:1
        │   └─ Block → type:void
        │       ├─ Assign → type:'integer', lev:1
        │       │   ├─ ArrayAccess: numbers → tab_index:36, obj:variable, type:'integer', lev:0
        │       │   │   └─ Var('i') → tab_index:37, obj:variable, type:'integer', lev:0
        │       │   └─ BinOp('*') → type:'integer', lev:1
        │       │       ├─ Var('i') → tab_index:37, obj:variable, type:'integer', lev:0
        │       │       └─ Number(2) → type:'integer', lev:1
        │       └─ Assign → type:'integer', lev:1
        │           ├─ Var('total') → tab_index:38, obj:variable, type:'integer', lev:0
        │           └─ BinOp('+') → type:'integer', lev:1
        │               ├─ Var('total') → tab_index:38, obj:variable, type:'integer', lev:0
        │               └─ ArrayAccess: numbers → tab_index:36, obj:variable, type:'integer', lev:0
        │                   └─ Var('i') → tab_index:37, obj:variable, type:'integer', lev:0
        ├─ Assign → type:'integer', lev:1
        │   ├─ Var('i') → tab_index:37, obj:variable, type:'integer', lev:0
        │   └─ Number(10) → type:'integer', lev:1
        └─ While → type:void, lev:1
            ├─ BinOp('dan') → type:'boolean', lev:1
            │   ├─ BinOp('>=') → type:'boolean', lev:1
            │   │   ├─ Var('i') → tab_index:37, obj:variable, type:'integer', lev:0
            │   │   └─ Number(1) → type:'integer', lev:1
            │   └─ BinOp('<=') → type:'boolean', lev:1
            │       ├─ Var('i') → tab_index:37, obj:variable, type:'integer', lev:0
            │       └─ Number(10) → type:'integer', lev:1
            └─ Block → type:void
                ├─ ProcedureCall(name: 'writeln') → tab_index:39, obj:procedure, type:'void', lev:1
                │   ├─ String("'Sum = '") → type:'string', lev:1
                │   └─ Var('total') → tab_index:38, obj:variable, type:'integer', lev:0
                └─ Assign → type:'integer', lev:1
                    ├─ Var('i') → tab_index:37, obj:variable, type:'integer', lev:0
                    └─ BinOp('-') → type:'integer', lev:1
                        ├─ Var('i') → tab_index:37, obj:variable, type:'integer', lev:0
                        └─ Number(1) → type:'integer', lev:1
==================================
```

Pengujian 2: tc2.pas

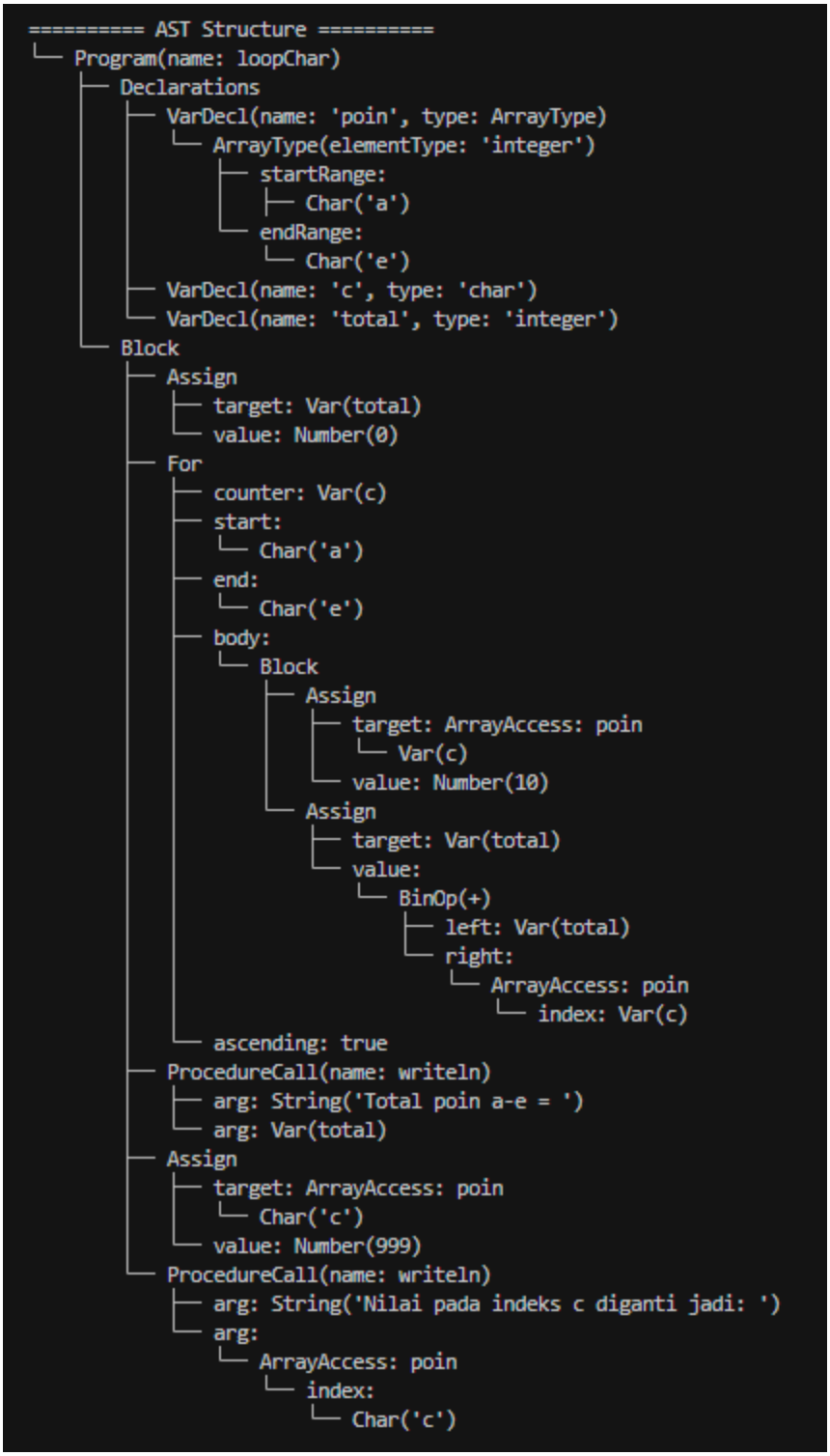
Test Case

```
program loopChar;
variabel
    poin: larik['a' .. 'e'] dari integer;
    c: char;
    total: integer;
mulai
    total := 0;

    untuk c := 'a' ke 'e' lakukan
    mulai
        poin[c] := 10;
        total := total + poin[c];
    selesai;

    writeln('Total poin a-e = ', total);

    poin['c'] := 999;
    writeln('Nilai pada indeks c diganti jadi: ', poin['c']);
selesai.
```

Bukti Input

```
bill2247@bill2247-TECNO-MEGABOOK:~/TUBES/JYP-Tubes-IF2224$ make run ARGS=test/milestone-3/tc2.pas
```

Bukti Output AST



Bukti Output Symbol Table tab

```
============================== TAB (Symbol Table) ==================================
idx     id                      obj     type    ref     nrm     lev     adr     link
--------------------------------------------------------------------------------
0       dan                     0       0       0       1       0       0       0
1       larik                   0       0       0       1       0       0       0
2       mulai                   0       0       0       1       0       0       0
3       kasus                   0       0       0       1       0       0       0
4       konstanta               0       0       0       1       0       0       0
5       bagi                    0       0       0       1       0       0       0
6       turun-ke                0       0       0       1       0       0       0
7       lakukan                 0       0       0       1       0       0       0
8       selain-itu              0       0       0       1       0       0       0
9       selesai                 0       0       0       1       0       0       0
10      untuk                   0       0       0       1       0       0       0
11      fungsi                  0       0       0       1       0       0       0
12      jika                    0       0       0       1       0       0       0
13      mod                     0       0       0       1       0       0       0
14      tidak                   0       0       0       1       0       0       0
15      dari                    0       0       0       1       0       0       0
16      atau                    0       0       0       1       0       0       0
17      prosedur                0       0       0       1       0       0       0
18      program                 0       0       0       1       0       0       0
19      rekaman                 0       0       0       1       0       0       0
20      ulangi                  0       0       0       1       0       0       0
21      string                  0       0       0       1       0       0       0
22      maka                    0       0       0       1       0       0       0
23      ke                      0       0       0       1       0       0       0
24      tipe                    0       0       0       1       0       0       0
25      sampai                  0       0       0       1       0       0       0
26      variabel                0       0       0       1       0       0       0
27      selama                  0       0       0       1       0       0       0
28      packed                  0       0       0       1       0       0       0
29      false                   1       3       0       1       0       0       0
30      true                    1       3       1       1       0       0       0
31      integer                 3       1       1       1       0       0       0
32      real                    3       2       1       1       0       0       0
33      boolean                 3       3       1       1       0       0       0
34      char                    3       4       1       1       0       0       0
35      loopChar                0       0       0       1       0       0       0
36      poin                    2       5       0       1       0       0       0
37      c                       2       4       0       1       0       5       36
38      total                   2       1       0       1       0       6       37
39      writeln                 4       0       0       1       0       0       0
================================================================================
```

Bukti Output Symbol Table atab

```
================== ATAB (Array Table) ======================
idx     xtyp    etyp    eref    low     high    elsz    size
------------------------------------------------------------
0       4       1       0       97      101     1       5
============================================================
```

Bukti Output Symbol Table btab

```
========== BTAB (Block Table) ==========
idx     last    lpar    psze    vsze
----------------------------------------
0       38      0       0       7
1       0       0       0       0
========================================
```

Bukti Output Decorated AST

```
========== Decorated AST ==========
└─ Program(name: 'loopChar') → tab_index:35, obj:program, type:'program', lev:0
    ├─ Declarations → type:void, lev:0
    │   ├─ VarDecl(name: 'poin') → tab_index:36, obj:variable, type:'array', lev:0
    │   │   └─ ArrayType → type:void
    │   │       ├─ Char('a') → type:void
    │   │       └─ Char('e') → type:void
    │   ├─ VarDecl(name: 'c') → tab_index:37, obj:variable, type:'char', lev:0
    │   └─ VarDecl(name: 'total') → tab_index:38, obj:variable, type:'integer', lev:0
    └─ Block → type:void, lev:1
        ├─ Assign → type:'integer', lev:1
        │   ├─ Var('total') → tab_index:38, obj:variable, type:'integer', lev:0
        │   └─ Number(0) → type:'integer', lev:1
        ├─ For → type:void, lev:1
        │   ├─ Var('c') → tab_index:37, obj:variable, type:'char', lev:0
        │   ├─ Char('a') → type:'char', lev:1
        │   ├─ Char('e') → type:'char', lev:1
        │   └─ Block → type:void
        │       ├─ Assign → type:'integer', lev:1
        │       │   ├─ ArrayAccess: poin → tab_index:36, obj:variable, type:'integer', lev:0
        │       │   │   └─ Var('c') → tab_index:37, obj:variable, type:'char', lev:0
        │       │   └─ Number(10) → type:'integer', lev:1
        │       └─ Assign → type:'integer', lev:1
        │           ├─ Var('total') → tab_index:38, obj:variable, type:'integer', lev:0
        │           └─ BinOp('+') → type:'integer', lev:1
        │               ├─ Var('total') → tab_index:38, obj:variable, type:'integer', lev:0
        │               └─ ArrayAccess: poin → tab_index:36, obj:variable, type:'integer', lev:0
        │                   └─ Var('c') → tab_index:37, obj:variable, type:'char', lev:0
        ├─ ProcedureCall(name: 'writeln') → tab_index:39, obj:procedure, type:'void', lev:1
        │   ├─ String("'Total poin a-e = '") → type:'string', lev:1
        │   └─ Var('total') → tab_index:38, obj:variable, type:'integer', lev:0
        ├─ Assign → type:'integer', lev:1
        │   ├─ ArrayAccess: poin → tab_index:36, obj:variable, type:'integer', lev:0
        │   │   └─ Char('c') → type:'char', lev:1
        │   └─ Number(999) → type:'integer', lev:1
        └─ ProcedureCall(name: 'writeln') → tab_index:39, obj:procedure, type:'void', lev:1
            ├─ String("'Nilai pada indeks c diganti jadi: '") → type:'string', lev:1
            └─ ArrayAccess: poin → tab_index:36, obj:variable, type:'integer', lev:0
                └─ Char('c') → type:'char', lev:1
==================================
```

Pengujian 3: tc3.pas

Test Case

```
program UjiFaktorial;
variabel
    hasil, n : integer;

fungsi faktorial(k : integer) : integer;
mulai
    jika (k <= 1) maka
        faktorial := 1;
    selain-itu
        faktorial := k * faktorial(k - 1);
selesai;

mulai
    n := 5;
    hasil := faktorial(n);
    writeln('Faktorial 5 = ', hasil);
selesai.
```

Bukti Input

```
bill2247@bill2247-TECNO-MEGABOOK:~/TUBES/JYP-Tubes-IF2224$ make run ARGS=test/milestone-3/tc3.pas
```

Bukti Output AST

```
========== AST Structure ==========
└── Program(name: UjiFaktorial)
    ├── Declarations
    │   ├── VarDecl(name: 'hasil', type: 'integer')
    │   ├── VarDecl(name: 'n', type: 'integer')
    │   └── FunctionDecl(name: 'faktorial', params: [(k : integer)], returnType: 'integer')
    └── Block
        ├── Assign
        │   ├── target: Var(n)
        │   └── value: Number(5)
        ├── Assign
        │   ├── target: Var(hasil)
        │   └── value:
        │       └── ProcedureCall(name: faktorial)
        │           └── arg: Var(n)
        └── ProcedureCall(name: writeln)
            ├── arg: String('Faktorial 5 = ')
            └── arg: Var(hasil)
```

Bukti Output Symbol Table tab

```
================================ TAB (Symbol Table) ==================================
idx     id                      obj     type    ref     nrm     lev     adr     link
------------------------------------------------------------------------------------
0       dan                     0       0       0       1       0       0       0
1       larik                   0       0       0       1       0       0       0
2       mulai                   0       0       0       1       0       0       0
3       kasus                   0       0       0       1       0       0       0
4       konstanta               0       0       0       1       0       0       0
5       bagi                    0       0       0       1       0       0       0
6       turun-ke                0       0       0       1       0       0       0
7       lakukan                 0       0       0       1       0       0       0
8       selain-itu              0       0       0       1       0       0       0
9       selesai                 0       0       0       1       0       0       0
10      untuk                   0       0       0       1       0       0       0
11      fungsi                  0       0       0       1       0       0       0
12      jika                    0       0       0       1       0       0       0
13      mod                     0       0       0       1       0       0       0
14      tidak                   0       0       0       1       0       0       0
15      dari                    0       0       0       1       0       0       0
16      atau                    0       0       0       1       0       0       0
17      prosedur                0       0       0       1       0       0       0
18      program                 0       0       0       1       0       0       0
19      rekaman                 0       0       0       1       0       0       0
20      ulangi                  0       0       0       1       0       0       0
21      string                  0       0       0       1       0       0       0
22      maka                    0       0       0       1       0       0       0
23      ke                      0       0       0       1       0       0       0
24      tipe                    0       0       0       1       0       0       0
25      sampai                  0       0       0       1       0       0       0
26      variabel                0       0       0       1       0       0       0
27      selama                  0       0       0       1       0       0       0
28      packed                  0       0       0       1       0       0       0
29      false                   1       3       0       1       0       0       0
30      true                    1       3       1       1       0       0       0
31      integer                 3       1       1       1       0       0       0
32      real                    3       2       1       1       0       0       0
33      boolean                 3       3       1       1       0       0       0
34      char                    3       4       1       1       0       0       0
35      UjiFaktorial            0       0       0       1       0       0       0
36      hasil                   2       1       0       1       0       0       0
37      n                       2       1       0       1       0       1       36
38      faktorial               5       1       1       1       0       0       37
39      k                       2       1       0       0       1       0       0
40      writeln                 4       0       0       1       0       0       0
====================================================================================
```

Bukti Output Symbol Table atab

```
================== ATAB (Array Table) ========================
idx     xtyp    etyp    eref    low     high    elsz    size
-------------------------------------------------------------
=============================================================
```

Bukti Output Symbol Table btab

```
========== BTAB (Block Table) ==========
idx     last    lpar    psze    vsze
---------------------------------------
0       38      0       0       2
1       39      39      1       0
2       0       0       0       0
=======================================
```

Bukti Output Decorated AST

```
========== Decorated AST ==========
└─ Program(name: 'UjiFaktorial') → tab_index:35, obj:program, type:'program', lev:0
    ├─ Declarations → type:void, lev:0
    │   ├─ VarDecl(name: 'hasil') → tab_index:36, obj:variable, type:'integer', lev:0
    │   ├─ VarDecl(name: 'n') → tab_index:37, obj:variable, type:'integer', lev:0
    │   └─ FunctionDecl(name: 'faktorial') → tab_index:38, obj:function, type:'integer', lev:0
    └─ Block → type:void, lev:1
        ├─ Assign → type:'integer', lev:1
        │   ├─ Var('n') → tab_index:37, obj:variable, type:'integer', lev:0
        │   └─ Number(5) → type:'integer', lev:1
        ├─ Assign → type:'integer', lev:1
        │   ├─ Var('hasil') → tab_index:36, obj:variable, type:'integer', lev:0
        │   └─ ProcedureCall(name: 'faktorial') → tab_index:38, obj:function, type:'integer', lev:1
        │       └─ Var('n') → tab_index:37, obj:variable, type:'integer', lev:0
        └─ ProcedureCall(name: 'writeln') → tab_index:40, obj:procedure, type:'void', lev:1
            ├─ String("'Faktorial 5 = '") → type:'string', lev:1
            └─ Var('hasil') → tab_index:36, obj:variable, type:'integer', lev:0
==================================
```

Pengujian 4: tc4.pas

Test Case

```
program TesScope;
variabel
    globalA, globalB : integer;

prosedur hitung(paramX : integer);
variabel
    lokalA : integer;
mulai
    lokalA := paramX + 10;
    globalA := lokalA;
selesai;

mulai
    globalA := 0;
    globalB := 5;

    hitung(globalB);
```

```
selesai.
```

Bukti Input

```
diyah@LAPTOP-6C00R5DE:/mnt/e/JYP-Tubes-IF2224-1$ make run ARGS=test/milestone-3/tc4.pas
./bin/main test/milestone-3/tc4.pas
```

Bukti Output AST

```
========== AST Structure ==========
└── Program(name: TesScope)
    ├── Declarations
    │   ├── VarDecl(name: 'globalA', type: 'integer')
    │   ├── VarDecl(name: 'globalB', type: 'integer')
    │   └── ProcedureDecl(name: 'hitung', params: [(paramX : integer)])
    │       └── Block
    │           ├── VarDecl(name: 'lokalA', type: 'integer')
    │           ├── Assign
    │           │   ├── target: Var(lokalA)
    │           │   └── value:
    │           │       └── BinOp(+)
    │           │           ├── left: Var(paramX)
    │           │           └── right: Number(10)
    │           └── Assign
    │               ├── target: Var(globalA)
    │               └── value: Var(lokalA)
    └── Block
        ├── Assign
        │   ├── target: Var(globalA)
        │   └── value: Number(0)
        ├── Assign
        │   ├── target: Var(globalB)
        │   └── value: Number(5)
        └── ProcedureCall(name: hitung)
            └── arg: Var(globalB)
```

Bukti Output Symbol Table tab

```
================================ TAB (Symbol Table) ==================================
idx     id                      obj     type    ref     nrm     lev     adr     link
------------------------------------------------------------------------------------
0       dan                     0       0       0       1       0       0       0
1       larik                   0       0       0       1       0       0       0
2       mulai                   0       0       0       1       0       0       0
3       kasus                   0       0       0       1       0       0       0
4       konstanta               0       0       0       1       0       0       0
5       bagi                    0       0       0       1       0       0       0
6       turun-ke                0       0       0       1       0       0       0
7       lakukan                 0       0       0       1       0       0       0
8       selain-itu              0       0       0       1       0       0       0
9       selesai                 0       0       0       1       0       0       0
10      untuk                   0       0       0       1       0       0       0
11      fungsi                  0       0       0       1       0       0       0
12      jika                    0       0       0       1       0       0       0
13      mod                     0       0       0       1       0       0       0
14      tidak                   0       0       0       1       0       0       0
15      dari                    0       0       0       1       0       0       0
16      atau                    0       0       0       1       0       0       0
17      prosedur                0       0       0       1       0       0       0
18      program                 0       0       0       1       0       0       0
19      rekaman                 0       0       0       1       0       0       0
20      ulangi                  0       0       0       1       0       0       0
21      string                  0       0       0       1       0       0       0
22      maka                    0       0       0       1       0       0       0
23      ke                      0       0       0       1       0       0       0
24      tipe                    0       0       0       1       0       0       0
25      sampai                  0       0       0       1       0       0       0
26      variabel                0       0       0       1       0       0       0
27      selama                  0       0       0       1       0       0       0
28      packed                  0       0       0       1       0       0       0
29      false                   1       3       0       1       0       0       0
30      true                    1       3       1       1       0       0       0
31      integer                 3       1       1       1       0       0       0
32      real                    3       2       1       1       0       0       0
33      boolean                 3       3       1       1       0       0       0
34      char                    3       4       1       1       0       0       0
35      TesScope                0       0       0       1       0       0       0
36      globalA                 2       1       0       1       0       0       0
37      globalB                 2       1       0       1       0       1       36
38      hitung                  4       0       1       1       0       0       37
39      paramX                  2       1       0       0       1       0       0
40      lokalA                  2       1       0       1       1       0       39
====================================================================================
```

Bukti Output Symbol Table atab

```
================= ATAB (Array Table) ======================
idx     xtyp    etyp    eref    low     high    elsz    size
-----------------------------------------------------------
===========================================================
```

Bukti Output Symbol Table btab

```
========== BTAB (Block Table) ==========
idx     last    lpar    psze    vsze
---------------------------------------
0       38      0       0       2
1       40      39      1       1
2       0       0       0       0
=======================================
```

Bukti Output Decorated AST

```
========== Decorated AST ==========
└─ Program(name: 'TesScope') → tab_index:35, obj:program, type:'program', lev:0
    ├─ Declarations → type:void, lev:0
    │   ├─ VarDecl(name: 'globalA') → tab_index:36, obj:variable, type:'integer', lev:0
    │   ├─ VarDecl(name: 'globalB') → tab_index:37, obj:variable, type:'integer', lev:0
    │   └─ ProcedureDecl(name: 'hitung') → tab_index:38, obj:procedure, type:'void', lev:0
    │       └─ Block → type:void
    │           ├─ VarDecl(name: 'lokalA') → tab_index:40, obj:variable, type:'integer', lev:1
    │           ├─ Assign → type:'integer', lev:1
    │           │   ├─ Var('lokalA') → tab_index:40, obj:variable, type:'integer', lev:1
    │           │   └─ BinOp('+') → type:'integer', lev:1
    │           │       ├─ Var('paramX') → tab_index:39, obj:variable, type:'integer', lev:1
    │           │       └─ Number(10) → type:'integer', lev:1
    │           └─ Assign → type:'integer', lev:1
    │               ├─ Var('globalA') → tab_index:36, obj:variable, type:'integer', lev:0
    │               └─ Var('lokalA') → tab_index:40, obj:variable, type:'integer', lev:1
    └─ Block → type:void, lev:1
        ├─ Assign → type:'integer', lev:1
        │   ├─ Var('globalA') → tab_index:36, obj:variable, type:'integer', lev:0
        │   └─ Number(0) → type:'integer', lev:1
        ├─ Assign → type:'integer', lev:1
        │   ├─ Var('globalB') → tab_index:37, obj:variable, type:'integer', lev:0
        │   └─ Number(5) → type:'integer', lev:1
        └─ ProcedureCall(name: 'hitung') → tab_index:38, obj:procedure, type:'void', lev:1
            └─ Var('globalB') → tab_index:37, obj:variable, type:'integer', lev:0
===================================
```

Pengujian 5: tc5.pas

Test Case

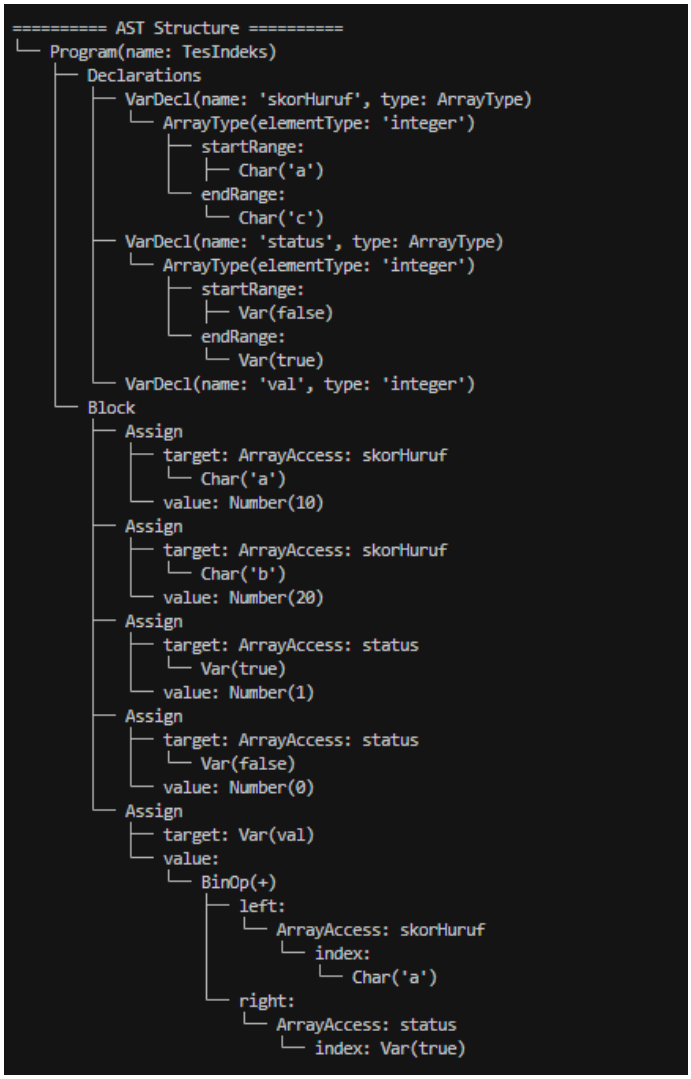
```
program TesIndeks;
variabel
    skorHuruf: larik['a' .. 'c'] dari integer;

    status: larik[false .. true] dari integer;

    val : integer;
mulai
```

```
    skorHuruf['a'] := 10;
    skorHuruf['b'] := 20;

    status[true] := 1;
    status[false] := 0;

    val := skorHuruf['a'] + status[true];
selesai.
```

Bukti Input

```
C:\Users\HP\Documents\Semester 5\TBFO\tubes\JYP-Tubes-IF2224>make run ARGS=test/milestone-3/tc5.pas
```

Bukti Output AST



Bukti Output Symbol Table tab

```
================================ TAB (Symbol Table) ==================================
idx     id                      obj     type    ref     nrm     lev     adr     link
------------------------------------------------------------------------------------
0       dan                     0       0       0       1       0       0       0
1       larik                   0       0       0       1       0       0       0
2       mulai                   0       0       0       1       0       0       0
3       kasus                   0       0       0       1       0       0       0
4       konstanta               0       0       0       1       0       0       0
5       bagi                    0       0       0       1       0       0       0
6       turun-ke                0       0       0       1       0       0       0
7       lakukan                 0       0       0       1       0       0       0
8       selain-itu              0       0       0       1       0       0       0
9       selesai                 0       0       0       1       0       0       0
10      untuk                   0       0       0       1       0       0       0
11      fungsi                  0       0       0       1       0       0       0
12      jika                    0       0       0       1       0       0       0
13      mod                     0       0       0       1       0       0       0
14      tidak                   0       0       0       1       0       0       0
15      dari                    0       0       0       1       0       0       0
16      atau                    0       0       0       1       0       0       0
17      prosedur                0       0       0       1       0       0       0
18      program                 0       0       0       1       0       0       0
19      rekaman                 0       0       0       1       0       0       0
20      ulangi                  0       0       0       1       0       0       0
21      string                  0       0       0       1       0       0       0
22      maka                    0       0       0       1       0       0       0
23      ke                      0       0       0       1       0       0       0
24      tipe                    0       0       0       1       0       0       0
25      sampai                  0       0       0       1       0       0       0
26      variabel                0       0       0       1       0       0       0
27      selama                  0       0       0       1       0       0       0
28      packed                  0       0       0       1       0       0       0
29      false                   1       3       0       1       0       0       0
30      true                    1       3       1       1       0       0       0
31      integer                 3       1       1       1       0       0       0
32      real                    3       2       1       1       0       0       0
33      boolean                 3       3       1       1       0       0       0
34      char                    3       4       1       1       0       0       0
35      TesIndeks               0       0       0       1       0       0       0
36      skorHuruf               2       5       0       1       0       0       0
37      status                  2       5       1       1       0       3       36
38      val                     2       1       0       1       0       5       37
====================================================================================
```

Bukti Output Symbol Table atab

```
================== ATAB (Array Table) ======================
idx     xtyp    etyp    eref    low     high    elsz    size
------------------------------------------------------------
0       4       1       0       97      99      1       3
1       3       1       0       0       1       1       2
============================================================
```

Bukti Output Symbol Table btab

```
========== BTAB (Block Table) ==========
idx     last    lpar    psze    vsze
---------------------------------------
0       38      0       0       6
1       0       0       0       0
=======================================
```

Bukti Output Decorated AST

```
========== Decorated AST ==========
└── Program(name: 'TesIndeks') → tab_index:35, obj:program, type:'program', lev:0
    ├── Declarations → type:void, lev:0
    │   ├── VarDecl(name: 'skorHuruf') → tab_index:36, obj:variable, type:'array', lev:0
    │   │   └── ArrayType → type:void
    │   │       ├── Char('a') → type:void
    │   │       └── Char('c') → type:void
    │   ├── VarDecl(name: 'status') → tab_index:37, obj:variable, type:'array', lev:0
    │   │   └── ArrayType → type:void
    │   │       ├── Var('false') → type:void
    │   │       └── Var('true') → type:void
    │   └── VarDecl(name: 'val') → tab_index:38, obj:variable, type:'integer', lev:0
    └── Block → type:void, lev:1
        ├── Assign → type:'integer', lev:1
        │   ├── ArrayAccess: skorHuruf → tab_index:36, obj:variable, type:'integer', lev:0
        │   │   └── Char('a') → type:'char', lev:1
        │   └── Number(10) → type:'integer', lev:1
        ├── Assign → type:'integer', lev:1
        │   ├── ArrayAccess: skorHuruf → tab_index:36, obj:variable, type:'integer', lev:0
        │   │   └── Char('b') → type:'char', lev:1
        │   └── Number(20) → type:'integer', lev:1
        ├── Assign → type:'integer', lev:1
        │   ├── ArrayAccess: status → tab_index:37, obj:variable, type:'integer', lev:0
        │   │   └── Var('true') → tab_index:30, obj:constant, type:'boolean', lev:0
        │   └── Number(1) → type:'integer', lev:1
        ├── Assign → type:'integer', lev:1
        │   ├── ArrayAccess: status → tab_index:37, obj:variable, type:'integer', lev:0
        │   │   └── Var('false') → tab_index:29, obj:constant, type:'boolean', lev:0
        │   └── Number(0) → type:'integer', lev:1
        └── Assign → type:'integer', lev:1
            ├── Var('val') → tab_index:38, obj:variable, type:'integer', lev:0
            └── BinOp('+') → type:'integer', lev:1
                ├── ArrayAccess: skorHuruf → tab_index:36, obj:variable, type:'integer', lev:0
                │   └── Char('a') → type:'char', lev:1
                └── ArrayAccess: status → tab_index:37, obj:variable, type:'integer', lev:0
                    └── Var('true') → tab_index:30, obj:constant, type:'boolean', lev:0
==================================
```

Pengujian 6: tc6.pas

Test Case

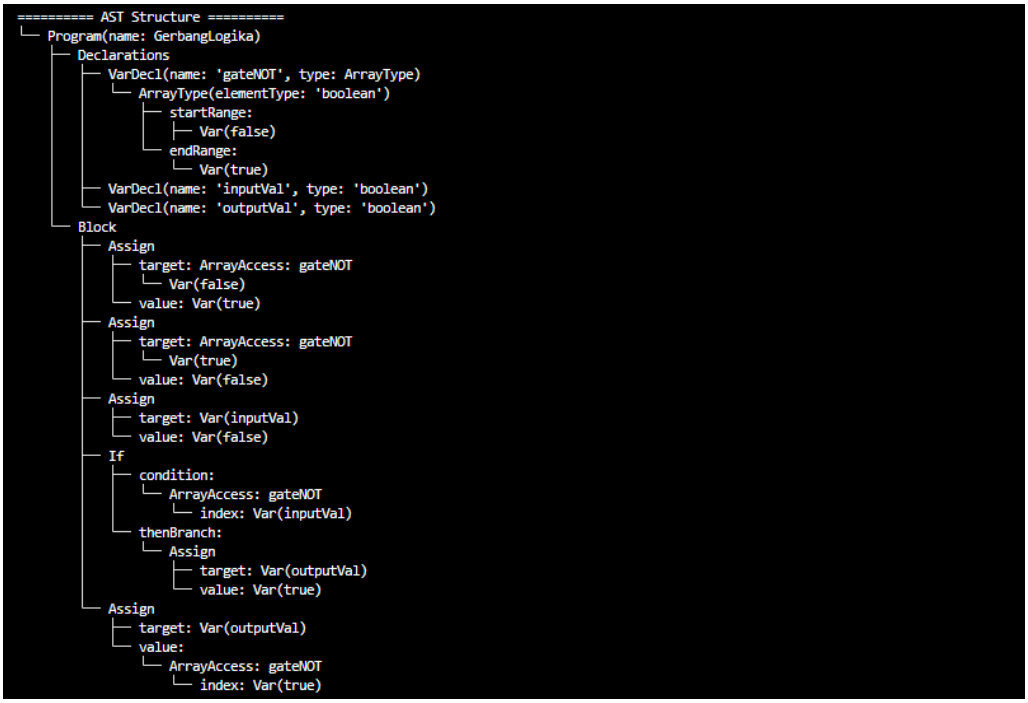
```
program GerbangLogika;
```

```
variabel
    gateNOT: larik[false .. true] dari boolean;
    inputVal, outputVal: boolean;
mulai
    gateNOT[false] := true;
    gateNOT[true] := false;

    inputVal := false;


    jika (gateNOT[inputVal]) maka
        outputVal := true;

    outputVal := gateNOT[true];
selesai.
```

Bukti Input

```
C:\Users\HP\Documents\Semester 5\TBFO\tubes\JYP-Tubes-IF2224>make run ARGS=test/milestone-3/tc6.pas
```

Bukti Output AST



Bukti Output Symbol Table tab

```
================================ TAB (Symbol Table) ==================================
idx     id                      obj     type    ref     nrm     lev     adr     link
------------------------------------------------------------------------------------
0       dan                     0       0       0       1       0       0       0
1       larik                   0       0       0       1       0       0       0
2       mulai                   0       0       0       1       0       0       0
3       kasus                   0       0       0       1       0       0       0
4       konstanta               0       0       0       1       0       0       0
5       bagi                    0       0       0       1       0       0       0
6       turun-ke                0       0       0       1       0       0       0
7       lakukan                 0       0       0       1       0       0       0
8       selain-itu              0       0       0       1       0       0       0
9       selesai                 0       0       0       1       0       0       0
10      untuk                   0       0       0       1       0       0       0
11      fungsi                  0       0       0       1       0       0       0
12      jika                    0       0       0       1       0       0       0
13      mod                     0       0       0       1       0       0       0
14      tidak                   0       0       0       1       0       0       0
15      dari                    0       0       0       1       0       0       0
16      atau                    0       0       0       1       0       0       0
17      prosedur                0       0       0       1       0       0       0
18      program                 0       0       0       1       0       0       0
19      rekaman                 0       0       0       1       0       0       0
20      ulangi                  0       0       0       1       0       0       0
21      string                  0       0       0       1       0       0       0
22      maka                    0       0       0       1       0       0       0
23      ke                      0       0       0       1       0       0       0
24      tipe                    0       0       0       1       0       0       0
25      sampai                  0       0       0       1       0       0       0
26      variabel                0       0       0       1       0       0       0
27      selama                  0       0       0       1       0       0       0
28      packed                  0       0       0       1       0       0       0
29      false                   1       3       0       1       0       0       0
30      true                    1       3       1       1       0       0       0
31      integer                 3       1       1       1       0       0       0
32      real                    3       2       1       1       0       0       0
33      boolean                 3       3       1       1       0       0       0
34      char                    3       4       1       1       0       0       0
35      GerbangLogika           0       0       0       1       0       0       0
36      gateNOT                 2       5       0       1       0       0       34
37      inputVal                2       3       0       1       0       2       36
38      outputVal               2       3       0       1       0       3       37
====================================================================================
```

Bukti Output Symbol Table atab

```
================== ATAB (Array Table) ========================
idx     xtyp    etyp    eref    low     high    elsz    size
-------------------------------------------------------------
0       3       3       0       0       1       1       2
=============================================================
```

Bukti Output Symbol Table btab

```
========== BTAB (Block Table) ==========
idx     last    lpar    psze    vsze
---------------------------------------
0       38      0       0       4
=======================================
```

Bukti Output Decorated AST

```
========== Decorated AST ==========
└── Program(name: 'GerbangLogika') → tab_index:35, obj:program, type:'program', lev:0
    ├── Declarations → type:void, lev:0
    │   ├── VarDecl(name: 'gateNOT') → tab_index:36, obj:variable, type:'array', lev:0
    │   │   └── ArrayType → type:void
    │   │       ├── Var('false') → type:void
    │   │       └── Var('true') → type:void
    │   ├── VarDecl(name: 'inputVal') → tab_index:37, obj:variable, type:'boolean', lev:0
    │   └── VarDecl(name: 'outputVal') → tab_index:38, obj:variable, type:'boolean', lev:0
    └── Block → type:void, lev:0
        ├── Assign → type:'boolean', lev:0
        │   ├── ArrayAccess: gateNOT → tab_index:36, obj:variable, type:'boolean', lev:0
        │   │   └── Var('false') → tab_index:29, obj:constant, type:'boolean', lev:0
        │   └── Var('true') → tab_index:30, obj:constant, type:'boolean', lev:0
        ├── Assign → type:'boolean', lev:0
        │   ├── ArrayAccess: gateNOT → tab_index:36, obj:variable, type:'boolean', lev:0
        │   │   └── Var('true') → tab_index:30, obj:constant, type:'boolean', lev:0
        │   └── Var('false') → tab_index:29, obj:constant, type:'boolean', lev:0
        ├── Assign → type:'boolean', lev:0
        │   ├── Var('inputVal') → tab_index:37, obj:variable, type:'boolean', lev:0
        │   └── Var('false') → tab_index:29, obj:constant, type:'boolean', lev:0
        ├── If → type:void, lev:0
        │   ├── ArrayAccess: gateNOT → tab_index:36, obj:variable, type:'boolean', lev:0
        │   │   └── Var('inputVal') → tab_index:37, obj:variable, type:'boolean', lev:0
        │   └── Assign → type:'boolean', lev:0
        │       ├── Var('outputVal') → tab_index:38, obj:variable, type:'boolean', lev:0
        │       └── Var('true') → tab_index:30, obj:constant, type:'boolean', lev:0
        └── Assign → type:'boolean', lev:0
            ├── Var('outputVal') → tab_index:38, obj:variable, type:'boolean', lev:0
            └── ArrayAccess: gateNOT → tab_index:36, obj:variable, type:'boolean', lev:0
                └── Var('true') → tab_index:30, obj:constant, type:'boolean', lev:0
==================================
```

Pengujian 7: tc7.pas

Test Case

```
program ShadowTest;
variabel
    data: integer;

prosedur proses(n: integer);
variabel
    listLokal: larik[1 .. 5] dari integer;
    i: integer;
mulai
```

```
    listLokal[1] := n * 10;

    data := listLokal[1];
selesai;

mulai
    data := 999;
    proses(5);
selesai.
```

Bukti Input

```
C:\Users\HP\Documents\Semester 5\TBFO\tubes\JYP-Tubes-IF2224>make run ARGS=test/milestone-3/tc7.pas
```

Bukti Output AST

Bukti Output Symbol Table tab

```
============================== TAB (Symbol Table) ==================================
idx     id                      obj     type    ref     nrm     lev     adr     link
----------------------------------------------------------------------------------
0       dan                     0       0       0       1       0       0       0
1       larik                   0       0       0       1       0       0       0
2       mulai                   0       0       0       1       0       0       0
3       kasus                   0       0       0       1       0       0       0
4       konstanta               0       0       0       1       0       0       0
5       bagi                    0       0       0       1       0       0       0
6       turun-ke                0       0       0       1       0       0       0
7       lakukan                 0       0       0       1       0       0       0
8       selain-itu              0       0       0       1       0       0       0
9       selesai                 0       0       0       1       0       0       0
10      untuk                   0       0       0       1       0       0       0
11      fungsi                  0       0       0       1       0       0       0
12      jika                    0       0       0       1       0       0       0
13      mod                     0       0       0       1       0       0       0
14      tidak                   0       0       0       1       0       0       0
15      dari                    0       0       0       1       0       0       0
16      atau                    0       0       0       1       0       0       0
17      prosedur                0       0       0       1       0       0       0
18      program                 0       0       0       1       0       0       0
19      rekaman                 0       0       0       1       0       0       0
20      ulangi                  0       0       0       1       0       0       0
21      string                  0       0       0       1       0       0       0
22      maka                    0       0       0       1       0       0       0
23      ke                      0       0       0       1       0       0       0
24      tipe                    0       0       0       1       0       0       0
25      sampai                  0       0       0       1       0       0       0
26      variabel                0       0       0       1       0       0       0
27      selama                  0       0       0       1       0       0       0
28      packed                  0       0       0       1       0       0       0
29      false                   1       3       0       1       0       0       0
30      true                    1       3       1       1       0       0       0
31      integer                 3       1       1       1       0       0       0
32      real                    3       2       1       1       0       0       0
33      boolean                 3       3       1       1       0       0       0
34      char                    3       4       1       1       0       0       0
35      ShadowTest              0       0       0       1       0       0       0
36      data                    2       1       0       1       0       0       0
37      proses                  4       0       1       1       0       0       36
38      n                       2       1       0       0       1       0       0
39      listLokal               2       5       0       1       1       0       38
40      i                       2       1       0       1       1       5       39
==================================================================================
```

Bukti Output Symbol Table atab

```
================== ATAB (Array Table) ======================
idx     xtyp    etyp    eref    low     high    elsz    size
------------------------------------------------------------
0       1       1       0       1       5       1       5
============================================================
```

Bukti Output Symbol Table btab

```
========== BTAB (Block Table) ==========
idx     last    lpar    psze    vsze
----------------------------------------
0       37      0       0       1
1       40      38      1       6
========================================
```

Bukti Output Decorated AST

```
========== Decorated AST ==========
└── Program(name: 'ShadowTest') → tab_index:35, obj:program, type:'program', lev:0
    ├── Declarations → type:void, lev:0
    │   ├── VarDecl(name: 'data') → tab_index:36, obj:variable, type:'integer', lev:0
    │   └── ProcedureDecl(name: 'proses') → tab_index:37, obj:procedure, type:'void', lev:0
    │       └── Block → type:void
    │           ├── VarDecl(name: 'listLokal') → tab_index:39, obj:variable, type:'array', lev:1
    │           │   └── ArrayType → type:void
    │           │       ├── Number(1) → type:void
    │           │       └── Number(5) → type:void
    │           ├── VarDecl(name: 'i') → tab_index:40, obj:variable, type:'integer', lev:1
    │           ├── Assign → type:'integer', lev:1
    │           │   ├── ArrayAccess: listLokal → tab_index:39, obj:variable, type:'integer', lev:1
    │           │   │   └── Number(1) → type:'integer', lev:1
    │           │   └── BinOp('*') → type:'integer', lev:1
    │           │       ├── Var('n') → tab_index:38, obj:variable, type:'integer', lev:1
    │           │       └── Number(10) → type:'integer', lev:1
    │           └── Assign → type:'integer', lev:1
    │               ├── Var('data') → tab_index:36, obj:variable, type:'integer', lev:0
    │               └── ArrayAccess: listLokal → tab_index:39, obj:variable, type:'integer', lev:1
    │                   └── Number(1) → type:'integer', lev:1
    └── Block → type:void, lev:0
        ├── Assign → type:'integer', lev:0
        │   ├── Var('data') → tab_index:36, obj:variable, type:'integer', lev:0
        │   └── Number(999) → type:'integer', lev:0
        └── ProcedureCall(name: 'proses') → tab_index:37, obj:procedure, type:'void', lev:0
            └── Number(5) → type:'integer', lev:0
==================================
```

D. KESIMPULAN DAN SARAN

1. Kesimpulan

Pada milestone 3, implementasi proses *semantic analysis* dengan *semantic analyzer* untuk bahasa Pascal-S versi bahasa Indonesia telah berhasil diselesaikan dengan baik. Dalam tahap ini, *compiler* mampu menciptakan *Abstract Syntax Tree* (AST) program masukan, melakukan pendataan *identifier, array,* dan *scope* program menggunakan beberapa jenis *symbol table (tab, atab, btab)*, serta memanfaatkan informasi tersebut untuk mendekorasi AST ke dalam format *decorated AST.* Tentu, penanganan *error* terkait semantik, seperti deklarasi *identifier* secara berulang pada suatu *scope*, turut ditangani dalam tahap *semantic analysis* ini.

Untuk memastikan bahwa program telah diimplementasikan secara optimal dan sebagaimana diharapkan, penulis melakukan pengujian dengan rangkaian *test case* yang

mewakili berbagai kasus analisis semantik. Berdasarkan pengujian, *semantic analyzer* dapat menghasilkan AST, mengisi informasi ke dalam *symtab*, mendekorasi AST, menangani kasus program dengan *array* dari tipe-tipe data tertentu, dan menangani kasus program dengan fungsi/prosedur, termasuk fungsi rekursif.

2. Saran

Dalam tahap *semantic analysis*, beberapa *error* yang disampaikan, serupa dengan *syntax analysis*, dapat ditulis dengan lebih informatif, misalnya menambahkan lokasi baris letak kesalahan semantik pada program. Kemudian, untuk memastikan lebih lanjut bahwa *compiler* yang dikembangkan telah menangani aturan dengan *robust*, pengujian lebih lanjut dengan program kasus uji berukuran lebih besar dan kompleks dapat dipertimbangkan.

E. LAMPIRAN

1. Link *Repository* Github

Berikut terlampir link *repository* github untuk tugas compiler Pascal-S kelompok JYP K-02:

https://github.com/numshv/JYP-Tubes-IF2224

2. Pembagian Tugas

Berikut terlampir pembagian tugas tiap anggota kelompok JYP K-02 pada *milestone 3*:

NIM	Nama	Tugas	Persentase
13523058	Noumisyifa Nabila Nareswari	Mengerjakan bagian AST dan print decorated AST	20%
13523066	M. Ghifary Komara Putra	Mengerjakan bagian *symbol table* tab	20%
13523072	Sabilul Huda	Mengerjakan bagian *symbol table* btab	20%
13523080	Diyah Susan Nugrahani	Mengerjakan bagian *symbol table* atab	20%
13523108	Henry Filberto Shinelo	Mengerjakan bagian AST dan Semantic Analysis	20%