IF2124 Teori Bahasa Formal dan Otomata

Milestone 3
Semantic Analysis

Laporan Tugas Besar

Disusun untuk memenuhi tugas besar mata kuliah IF 2124 Teori Bahasa Formal dan Otomata pada Semester I Tahun Akademik 2025/2026

Oleh

Raka Daffa Iftikhaar	13523018
Aliya Husna Fayyaza	13523062
Ahsan Malik Al Farisi	13523074
Bevinda Vivian	13523120

Kelompok Ahsan Et Al (NTB)

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

2025

Daftar Isi

BAB I

Landasan Teori

1.1. Compiler dan Fase-Fase Kompilasi

Compiler adalah sebuah program komputer yang menerjemahkan *source code* yang ditulis dalam bahasa pemrograman tingkat tinggi menjadi bahasa mesin atau *object code* yang dapat dieksekusi langsung oleh komputer. Proses kompilasi merupakan serangkaian transformasi yang kompleks dan terstruktur, yang secara umum dibagi menjadi enam fase utama.

1) Fase pertama adalah analisis leksikal (*lexical analysis*), di mana kode sumber dibaca karakter demi karakter dan dikelompokkan menjadi unit-unit leksikal yang disebut token.
2) Fase kedua adalah analisis sintaks (*syntax analysis atau parsing*), yang memeriksa apakah urutan token membentuk struktur yang sesuai dengan tata bahasa (*grammar*) bahasa pemrograman.
3) Fase ketiga adalah analisis semantik (*semantic analysis*), yang memeriksa konsistensi makna dari program, seperti pengecekan tipe data dan deklarasi variabel.
4) Fase keempat adalah pembangkitan kode antara (*intermediate code generation*), yang menghasilkan representasi abstrak dari program yang lebih mudah dioptimasi.
5) Fase kelima adalah optimasi kode (*code optimization*), yang bertujuan memperbaiki kode antara agar lebih efisien tanpa mengubah fungsionalitas program.
6) Fase terakhir adalah pembangkitan kode *(code generation*), yang menghasilkan kode mesin atau assembly yang dapat dieksekusi.

Pada Milestone 3 ini, fokus utama adalah pada fase ketiga, yaitu analisis semantik, yang merupakan tahap lanjutan dari analisis sintaks. Analisis semantik berfungsi untuk memvalidasi konsistensi deklarasi dan juga statement yang dibuat oleh program. Analisis semantik akan menghasilkan *Decorated Abstract Syntax Tree* menggunakan algoritma L-Attributed Grammar. Selain itu, analisis semantik juga akan menghasilkan symbol table yang dapat menyimpan informasi nama identifier, tipe datanya, dan juga scopenya.

1.2. Analisis Semantik

Analisis semantik adalah tahap kompilasi yang memeriksa kebenaran makna (meaning correctness) dari program. Jika analisis sintaks memastikan bahwa struktur program valid secara grammar, maka analisis semantik memastikan bahwa struktur tersebut masuk akal sesuai aturan bahasa. Beberapa aspek yang diperiksa pada tahap ini meliputi:

a. Pengecekan Deklarasi

   Identifier harus dideklarasikan sebelum digunakan dan duplikasi deklarasi pada scope yang sama akan dianggap error. Prosedur dan fungsi yang dipakai memiliki definisi yang valid.

b. Pengecekan Tipe (Type Checking)

   Operator harus digunakan dengan operand bertipe sesuai dan semua ekspresi aritmetika, boolean, assignment, dan relasi harus valid. Selain itu, tipe hasil ekspresi harus dapat ditentukan dan ditempelkan ke AST.

c. Pengecekan Scope (Scoping Rules)

   Setiap identifier hidup dalam lingkungan (*scope*) tertentu dengan blok program baru akan membuka scope baru. Selain itu, Identifier dalam scope lokal akan mendapatkan shadowing identifier global.

d. Konsistensi Prosedur/Fungsi

   Validasi konsistensi dilakukan dengan mengecek kesesuaian jumlah dan tipe parameter dengan deklarasinya dan fungsi harus memiliki nilai kembali (return value) yang valid.

Untuk melaksanakan seluruh proses di atas, analisis semantik akan menghasilkan dua struktur utama, yaitu Decorated AST dan Symbol Table.

1.3. Abstract Syntax Tree (AST) dan Decorated AST

Abstract Syntax Tree (AST) merupakan representasi pohon dari struktur sintaks program yang disederhanakan dari *Parse Tree* pada proses analisis sintaks. AST akan menghilangkan elemen sintaks yang tidak relevan, misalnya simbol terminal terentu, tetapi tetap mempertahankan susunan logis dari konstruksi bahasa. Misalnya, pada kode " a + b * c" akan dihasilkan AST seperti berikut:

```
     (+)
    /  \
  a    (*)
       /  \
      b    c
```

Ekstensi dari AST merupakan Decorated AST yang akan menambahkan berbagai detail informasi, seperti tipe dari setiap ekspresi, referensi ke deklarasi identifier, informasi scope, dan atribut yang diperlukan untuk tahap kompilasi berikutnya. Decorated AST ini akan dihasilkan dengan aturan L-Attributed Grammar, karena atributnya dapat dihitung menggunakan traversal top-down dan left-to-right, sehingga cocok dengan model recursive descent dari tahap analisis sintaks.

1.4. Symbol Table

Symbol Table merupakan struktur data yang menyimpan seluruh informasi terkait identifier dalam program. Symbol table merupakan komponen yang sangat penting pada analisis semantik. Informasi yang disimpan dalam symbol table meliputi nama identifier, tipe data, kategori, scope, parameter, dan lainnya. Symbol table umumnya dibangun dengan hirarkis atau menggunakan *chained environments* sehingga setiap blok memiliki tabel simbolnya sendiri, namun tetap dapat mengakses tabel simbol di level atasnya.

1.5. L-Attributed Grammar dan Atribut Semantik

Pada milestone ini analisis semantik menggunakan pendekatan L-Attributed Grammar, yaitu kelas grammar beratribut dengan nilai atribut setiap node dapat dihitung dengan traversal kiri ke kanan menggunakan informasi dari node parent dan *sibling* sebelumnya. Tipe atribut yang digunakan adalah Inherited Attributes (INH) yaitu atribut yang diturunkan dari parent ke child, seperti tipe variabel saat deklarasi dan juga Synthesized Attributes (SYN) yaitu atribut yang naik dari child ke parent, seperti tipe hasil ekspresi.

1.6. Teori Bahasa Formal dan Hirarki Chomsky

Dalam proses analisis semantik, grammar yang digunakan tetap Context-Free Grammar (CFG), karena CFG mendefinisikan bentuk sintaks yang membangun AST, kemudian struktur nested tidak dapat direpresentasikan dengan regular grammar. Dengan demikian, *semantic rules* berjalan di atas hasil struktur CFG dan bukan menggantikan grammar.

BAB 2

Perancangan & Implementasi

2.1 Gambaran Umum Implementasi

Semantic analyzer dalam compiler Pascal-S ini terdiri dari tiga komponen utama yang bekerja secara terintegrasi yaitu symbol tabel yang terdiri dari TAB, BTAB, ATAB, dan display, scope and type checker yang terdiri dari visitor pattern, scope resolution, dan type validation, terakhir decorated AST printer yang terdiri dari semantic annotation dan output formatting. Program ini mendapatkan input dari AST parser dan mengeluarkan output berupa decorated AST dan beberapa symbol table.

2.2 Perancangan Symbol Table

2.2.1 Struktur Tiga Tabel

Symbol table diimplementasikan menggunakan tiga tabel terpisah untuk efisiensi dan modularitas.

i. TAB

Menyimpan semua identifier yang muncul dalam program.

```
struct TabEntry {
   std::string name;    // Nama identifier
   int link;            // Pointer ke identifier sama di outer scope
   ObjectKind obj;      // VARIABLE, CONSTANT, TYPE_ID,
PROCEDURE, FUNCTION
   BaseType typ;        // INTS, REALS, BOOLS, CHARS, NOTYPE
   int ref;             // Index ke ATAB/BTAB (bergantung obj)
   bool normal;         // Parameter passing: true=value,
false=reference
   int lev;             // Nesting level (0=global, 1=lokal, ...)
   int adr;             // Offset dalam block
};
```

Penjelasan:
link: Membentuk linked list untuk identifier dengan nama sama di scope berbeda. Contoh: variabel lokal x bisa coexist dengan global x.
ref: Multi-purpose reference:
Array: Index ATAB (info dimensi)
Procedure/Function: index BTAB (info block)
Lainnya: 0

lev: Menentukan visibility (0=global, semakin besar semakin dalam nested)
adr: Offset relatif untuk alokasi memori runtime

ii. BTAB
Menyimpan informasi block (program, procedure, function).

```
struct BTabEntry {
    int last;     // Index TAB identifier terakhir di block ini
    int lastpar;  // Index TAB parameter terakhir
    int psize;    // Total ukuran parameter
    int vsize;    // Total ukuran variabel lokal
};
```

Berfungsi untuk memisahkan parameter dari variabel lokal, menghitung memory footprint block, dan menandai boundary identifier dalam scope.

iii. ATAB
Menyimpan informasi array untuk bounds checking.

```
struct ATabEntry {
    int inxtyp;   // Tipe index (INTS)
    int eltyp;    // Tipe elemen (INTS, REALS, ...)
    int elref;    // Reference untuk multidimensi
    int low;      // Lower bound
    int high;     // Upper bound
    int elsize;   // Ukuran per elemen
    int size;     // Total ukuran
};
```

2.2.2 *Display Stack Untuk Scope Management*

Display adalah stack yang menyimpan index BTAB aktif untuk setiap level nesting.

```
std::vector<int> display;
// display[0] → BTAB index untuk level 0 (global)
// display[1] → BTAB index untuk level 1 (lokal pertama)
```

Cara Kerja:

```
Program (level 0, BTAB[0])
├─ Procedure A (level 1, BTAB[1])
│   └─ Procedure B (level 2, BTAB[2])
│
```

```
Display = [0, 1, 2]

Lookup identifier: cari di BTAB[2] → BTAB[1] → BTAB[0]
```

Algoritma Scope Resolution

```
for lev from current_level down to 0:
  search in BTAB[display[lev]]
  if found: return
not found: error
```

2.2.3 *Reserved Word Indexing*

```
reserved_words = [
  "program", "variabel", "mulai", "selesai", "const",
  "tipe", "prosedur", "fungsi", "jika", "maka",
  "selainitu", "untuk", "ke", "turun", "lakukan",
  "selama", "ulangi", "sampai", "larik", "dari",
  "integer", "real", "boolean", "char",
  "and", "or", "not", "div", "mod"
]
```

Keuntungan dari penggunaan hal ini adalah user identifier konsisten dengan dimulai dari index 29, mudah membedakan antara keyword dan identifier, dan kompatibel dengan spesifikasi Pascal-S.

2.2.4 *Operasi Insert*

Insert menambahkan identifier baru dengan validasi duplikasi.

Algoritma

```
insert(name, obj, typ, ref, normal, adr):
  // 1. Cek duplikasi di scope saat ini
  current_block = btab[display[level]]
  i = current_block.last

  while i > 0:
    if tab[i].name == name:
      throw "Duplicate identifier: " + name
    i = tab[i].link

  // 2. Buat entry baru
  entry = TabEntry{
    name,
    link: current_block.last,  // Link ke entry sebelumnya
    obj, typ, ref, normal,
    lev: level,
    adr
  }

  // 3. Tambahkan ke TAB
  tab.push_back(entry)

  // 4. Update BTAB
  btab[display[level]].last = t

  return t++
```

Terdapat beberapa poin penting dari hal tersebut. Pertama, duplikasi hanya dicek dalam scope yang sama via linked list, kedua entry baru menjadi last di BTAB, ketiga link membentuk chain untuk traversal.

2.2.5 *Operasi Lookup*

Lookup mencari identifier dari scope terdalam ke global.

Algoritma

```
lookup(name):
  // 1. Cari dari level saat ini ke level 0
  for lev from level down to 0:
    i = btab[display[lev]].last

    // Traverse linked list
    while i > 0:
      if tab[i].name == name:
        return i  // Found
      i = tab[i].link

  // 2. Auto-insert standard procedures
  if name in ["write", "writeln", "read", "readln"]:
    return insert(name, PROCEDURE, NOTYPE, 0, false, 0)

  return -1  // Not found
```

Prosedur pada bagian ini adalah standard procedures tidak dideklarasikan eksplisit, ditambahkan secara lazy saat pertama kali digunakan, dan memiliki lev=0 untuk menandakan predefined.

2.2.6 *Operasi Enter/Exit Scope*

Enter Scope:

```
enter_scope():
   level++
   display.push_back(b)

   btab.push_back(BTabEntry{
      last: 0,
      lastpar: 0,
      psize: 0,
      vsize: 0
   })

   b++
```

Hal ini dipanggil saat memulai program dan masuk ke prosedur/fungsi.

Exit Scope

```
exit_scope():
   level--
   display.pop_back()
```

Hal ini dipanggil saat selesai memproses block dan kembali ke outer scope.

2.3 Implementasi Scope and Type Checker

2.3.1 Visitor Pattern

Menggunakan Visitor Pattern untuk traversal AST dengan separation of concerns

```
class ScopeTypeChecker : public ASTVisitor {
private:
   SymbolTable* symbolTable;

public:
   void visitProgram(ASTProgramNode* node) override;
   void visitVarDecl(ASTVarDeclNode* node) override;
   void visitTypeDecl(TypeDeclarationNode* node) override;
   void visitAssign(ASTAssignNode* node) override;
   void visitBinOp(ASTBinOpNode* node) override;
   void visitProcedureCall(ASTProcedureCallNode* node) override;
   // ... visitor methods lainnya
};
```

Keuntungan dari implementasi ini adalah semantic logic terpisah dari struktur AST, mudah extend dengan visitor baru (code generator, optimizer), serta type-safe dimana compiler akan memberikan warning jika ada node tidak ter-handle.

2.3.2 *Visit Program Node*

Proses yang dilakukan pada implementasi ini adalah insert nama program ke TAB (sebagai CONSTANT), kemudian enter scope global (BTAB[0]), lalu visit block (deklarasi + statement), terakhir exit scope.

Implementasi

```
visitProgram(node):
  symbolTable->insert(
    node->program_name,
    ObjectKind::CONSTANT,
    BaseType::NOTYPE,
    0, false, 0
  )

  symbolTable->enter_scope()  // level 0 → 1
  node->block->accept(this)
  symbolTable->exit_scope()   // level 1 → 0
```

2.3.3 *Visit Variable Declaration*

Proses yang dilakukan pada implementasi ini adalah resolve tipe variabel(built-in atau user-defined), insert setiap variabel ke TAB, assign alamat berdasarkan vsize, dan increment vsize.

Implementasi

```
visitVarDecl(node):
  // 1. Tentukan BaseType
  if node->type == "integer":
    varType = BaseType::INTS
  else if node->type == "real":
    varType = BaseType::REALS
  else if node->type == "boolean":
    varType = BaseType::BOOLS
  else if node->type == "char":
    varType = BaseType::CHARS
  else:
    // User-defined type
    typeIdx = symbolTable->lookup(node->type)
    if typeIdx == -1:
      throw "Undefined type: " + node->type

    typeEntry = tab[typeIdx]
```

```
        if typeEntry.obj != ObjectKind::TYPE_ID:
            throw node->type + " is not a type"

        varType = typeEntry.typ
        ref = typeEntry.ref

    // 2. Insert setiap variabel
    for varName in node->identifiers:
        adr = btab[display[level]].vsize

        symbolTable->insert(
            varName,
            ObjectKind::VARIABLE,
            varType,
            ref, false, adr
        )

        btab[display[level]].vsize++
```

2.3.4 *Visit Type Declaration*

Implementasi ini menyelesaikan masalah Range type seperti IntRange = 1..100 harus diperlakukan sebagai alias integer.

Implementasi

```
visitTypeDecl(node):
    for typeDef in node->type_defs:
        if typeDef.pars_type_definition adalah RangeNode:
            // Range type = integer alias
            symbolTable->insert(
                typeDef.identifier,
                ObjectKind::TYPE_ID,
                BaseType::INTS,    // Range → integer
                0,                 // ref = 0 (bukan array)
                false, 0
            )

        else if typeDef.pars_type_definition adalah ArrayTypeNode:
            // Create ATAB entry
            // Insert dengan ref = ATAB index
```

2.3.5 *Visit Assignment State*

Algoritma ini harus memiliki beberapa validasi diantaranya target identifier harus ada (defined) dan target harus bertipe VARIABLE (bukan CONSTANT).

Implementasi

```
visitAssign(node):
   // 1. Lookup target
   idx = symbolTable->lookup(node->identifier)
   if idx == -1:
      throw "Undefined variable: " + node->identifier

   // 2. Validasi obj type
   entry = tab[idx]
   if entry.obj != ObjectKind::VARIABLE:
      throw node->identifier + " is not a variable"

   // 3. Visit expression (rekursif)
   node->expression->accept(this)
```

2.3.6 *Visit Binary Operation*

Proses dari implementasi ini adalah visit operand kiri (rekursif) dan visit operand kanan (rekursif).

Implementasi

```
visitBinOp(node):
   node->left->accept(this)
   node->right->accept(this)
```

2.3.7 *Visit Procedure Call*

Implementasi ini memiliki dua validasi diantaranya prosedur harus ada atau merupakan standard procedure dan identifier harus bertipe PROCEDURE.

Implementasi

```
visitProcedureCall(node):
   // 1. Lookup (auto-insert jika standard procedure)
   idx = symbolTable->lookup(node->procedure_name)
   if idx == -1:
      throw "Undefined procedure: " + node->procedure_name

   // 2. Validasi object kind
   entry = tab[idx]
   if entry.obj != ObjectKind::PROCEDURE:
      throw node->procedure_name + " is not a procedure"

   // 3. Visit arguments
   for arg in node->arguments:
      arg->accept(this)
```

2.4 Decorated AST Printer

2.4.1 Konsep

Decorated AST memperkaya AST dengan anotasi hasil semantic analysis berupa tab_index yaitu posisi di symbol table, type untuk tipe data, lev untuk nesting level, dan predefined yaitu flag untuk standard procedures.

2.4.2 Visit Identifier

Implementasi

```
visitIdentifier(node):
  out << "'" << node->name << "'"

  idx = symTab->lookup(node->name)
  if idx != -1:
    entry = symTab->get_tab(idx)
    out << " → tab_index:" << idx
    out << ", type:" << getTypeString(entry.typ)
    out << ", lev:" << entry.lev
```

Output

```
'a' → tab_index:30, type:integer, lev:0
'b' → tab_index:31, type:integer, lev:0
```

2.4.3 Visit Assignment

Implementasi

```
visitAssign(node):
  out << "Assign("

  idx = symTab->lookup(node->identifier)
  if idx != -1:
    entry = symTab->get_tab(idx)
    out << "'" << node->identifier << "'"
    out << " → tab_index:" << idx
    out << ", type:" << getTypeString(entry.typ)

  out << " := "

  indent += 2
  node->expression->accept(this)  // Rekursif
  indent -= 2

  out << ")"
```

Output

```
Assign('a' → tab_index:30, type:integer := 5)
  └─ value: 5 → type:integer
```

2.4.4 *Visit Procedure Call*

Implementasi

```
visitProcedureCall(node):
    out << node->procedure_name << "(...)"

    idx = symTab->lookup(node->procedure_name)
    if idx != -1:
        entry = symTab->get_tab(idx)
        out << " → "

        // Flag predefined
        if entry.lev == 0 && entry.obj == PROCEDURE:
            out << "predefined, "

        out << "tab_index:" << idx
```

Output

```
writeln(...) → predefined, tab_index:32
myProc(...) → tab_index:33
```

Pada implementasi ini, predefined adalah standard procedure (write, writeln, read, readln) sementara untuk lev == 0 telah defined di global scope.

2.4.5 *Visit Binary Operation*

Implementasi

```
visitBinOp(node):
    out << "BinOp '" << node->op << "'"

    indent += 2
    node->left->accept(this)
    node->right->accept(this)
    indent -= 2
```

Output

```
BinOp '+'
   ├─ 'a' → tab_index:30, type:integer
   └─ 10 → type:integer
```

2.5 Integrasi dengan Main Program

2.5.1 Execution Flow

Implementasi

```
main(argc, argv):
  // ... Lexer dan Parser ...

  // 1. Build AST
  ASTBuilder astBuilder
  ast = astBuilder.build(parseTree)

  // 2. Semantic Analysis
  SymbolTable symTab
  ScopeTypeChecker typeChecker(&symTab)
  ast->accept(&typeChecker)

  // 3. Output (jika flag --decorated)
  if decorated:
    print_symbol_table(symTab)
    print_decorated_ast(ast, symTab)
```

Setelah lexer dan parser selesai, ASTBuilder mengonversi parse tree menjadi AST dengan memanggil build(parseTree). Kemudian dibuat objek SymbolTable untuk menyimpan informasi identifier (TAB, BTAB, ATAB) dan ScopeTypeChecker yang menerima pointer symbol table. Proses semantic analysis dimulai saat ast->accept(&typeChecker) dipanggil, yang membuat type checker mengunjungi setiap node AST secara rekursif untuk memvalidasi scope, mendeteksi undefined/duplicate identifier, dan mengisi symbol table. Setelah analisis selesai, jika flag --decorated aktif, program mencetak symbol table (TAB dan BTAB) dan decorated AST melalui ASTDecoratedPrinter yang menampilkan setiap node dengan anotasi semantik seperti tab_index, type, lev, dan flag predefined.

BAB III

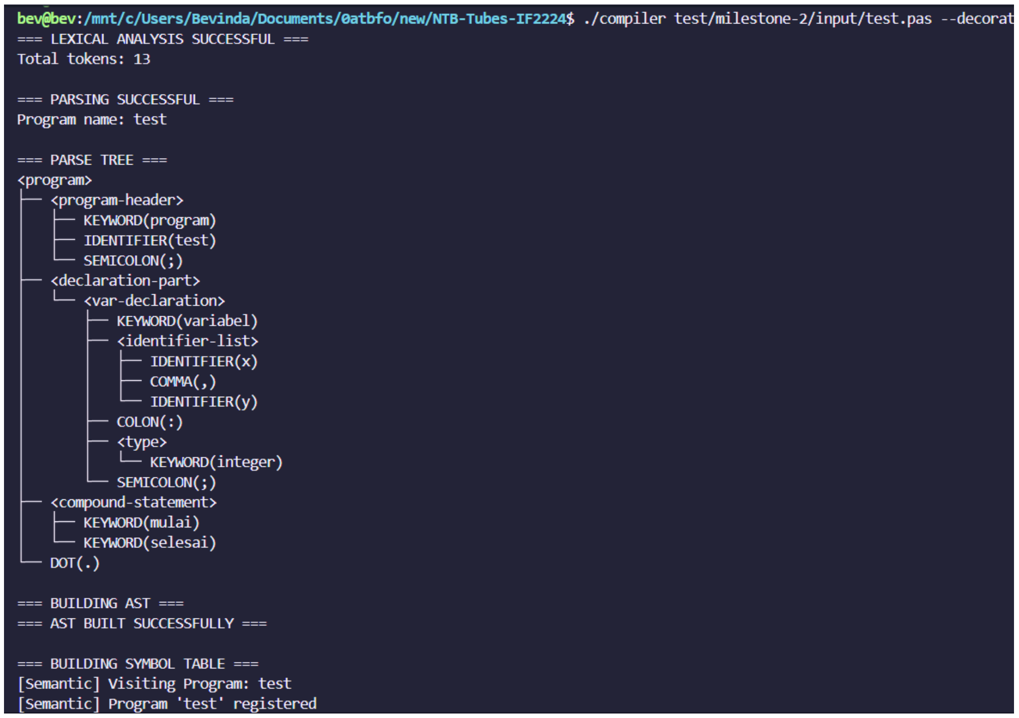
Pengujian

3.1. Test Case 1 - TestMinimal

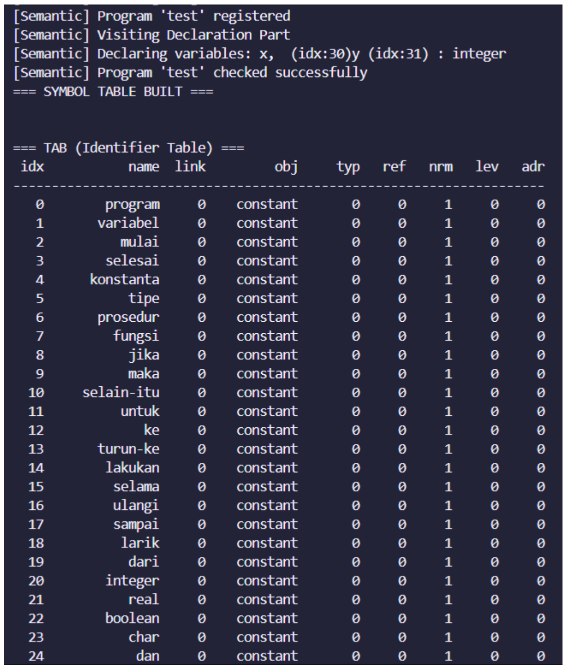
Test case ini bertujuan untuk memvalidasi bahwa semantic analyzer dapat membangun symbol table untuk deklarasi variabel sederhana, melakukan scope checking, dan menghasilkan decorated AST dengan informasi tipe untuk setiap node. Kasus ini ditujukan untuk memastikan struktur dasar semantic analysis berjalan dengan benar.

test.pas (input)
```
program TestMinimal;

variabel
  x, y: integer;

mulai
selesai.
```

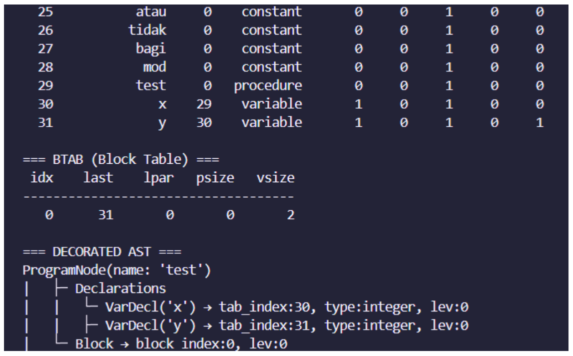
Output:



```
bev@bev:/mnt/c/Users/Bevinda/Documents/0atbfo/new/NTB-Tubes-IF2224$ ./compiler test/milestone-2/input/test.pas --decorat
=== LEXICAL ANALYSIS SUCCESSFUL ===
Total tokens: 13

=== PARSING SUCCESSFUL ===
Program name: test

=== PARSE TREE ===
<program>
├── <program-header>
│   ├── KEYWORD(program)
│   ├── IDENTIFIER(test)
│   └── SEMICOLON(;)
├── <declaration-part>
│   └── <var-declaration>
│       ├── KEYWORD(variabel)
│       ├── <identifier-list>
│       │   ├── IDENTIFIER(x)
│       │   ├── COMMA(,)
│       │   └── IDENTIFIER(y)
│       ├── COLON(:)
│       ├── <type>
│       │   └── KEYWORD(integer)
│       └── SEMICOLON(;)
├── <compound-statement>
│   ├── KEYWORD(mulai)
│   └── KEYWORD(selesai)
└── DOT(.)

=== BUILDING AST ===
=== AST BUILT SUCCESSFULLY ===

=== BUILDING SYMBOL TABLE ===
[Semantic] Visiting Program: test
[Semantic] Program 'test' registered
```



```
[Semantic] Program 'test' registered
[Semantic] Visiting Declaration Part
[Semantic] Declaring variables: x,  (idx:30)y (idx:31) : integer
[Semantic] Program 'test' checked successfully
=== SYMBOL TABLE BUILT ===


=== TAB (Identifier Table) ===
 idx          name  link        obj     typ   ref   nrm   lev   adr
--------------------------------------------------------------------
   0       program     0   constant       0     0     1     0     0
   1      variabel     0   constant       0     0     1     0     0
   2         mulai     0   constant       0     0     1     0     0
   3       selesai     0   constant       0     0     1     0     0
   4     konstanta     0   constant       0     0     1     0     0
   5          tipe     0   constant       0     0     1     0     0
   6      prosedur     0   constant       0     0     1     0     0
   7        fungsi     0   constant       0     0     1     0     0
   8          jika     0   constant       0     0     1     0     0
   9          maka     0   constant       0     0     1     0     0
  10    selain-itu     0   constant       0     0     1     0     0
  11         untuk     0   constant       0     0     1     0     0
  12            ke     0   constant       0     0     1     0     0
  13      turun-ke     0   constant       0     0     1     0     0
  14       lakukan     0   constant       0     0     1     0     0
  15        selama     0   constant       0     0     1     0     0
  16        ulangi     0   constant       0     0     1     0     0
  17        sampai     0   constant       0     0     1     0     0
  18         larik     0   constant       0     0     1     0     0
  19          dari     0   constant       0     0     1     0     0
  20       integer     0   constant       0     0     1     0     0
  21          real     0   constant       0     0     1     0     0
  22       boolean     0   constant       0     0     1     0     0
  23          char     0   constant       0     0     1     0     0
  24           dan     0   constant       0     0     1     0     0
```



```
  25          atau     0   constant       0     0     1     0     0
  26         tidak     0   constant       0     0     1     0     0
  27          bagi     0   constant       0     0     1     0     0
  28           mod     0   constant       0     0     1     0     0
  29          test     0  procedure       0     0     1     0     0
  30             x    29   variable       1     0     1     0     0
  31             y    30   variable       1     0     1     0     1

=== BTAB (Block Table) ===
 idx    last    lpar   psize   vsize
------------------------------------
   0      31       0       0       2

=== DECORATED AST ===
ProgramNode(name: 'test')
|   ├─ Declarations
|   |   └─ VarDecl('x') → tab_index:30, type:integer, lev:0
|   |   ├─ VarDecl('y') → tab_index:31, type:integer, lev:0
|   └─ Block → block index:0, lev:0
```

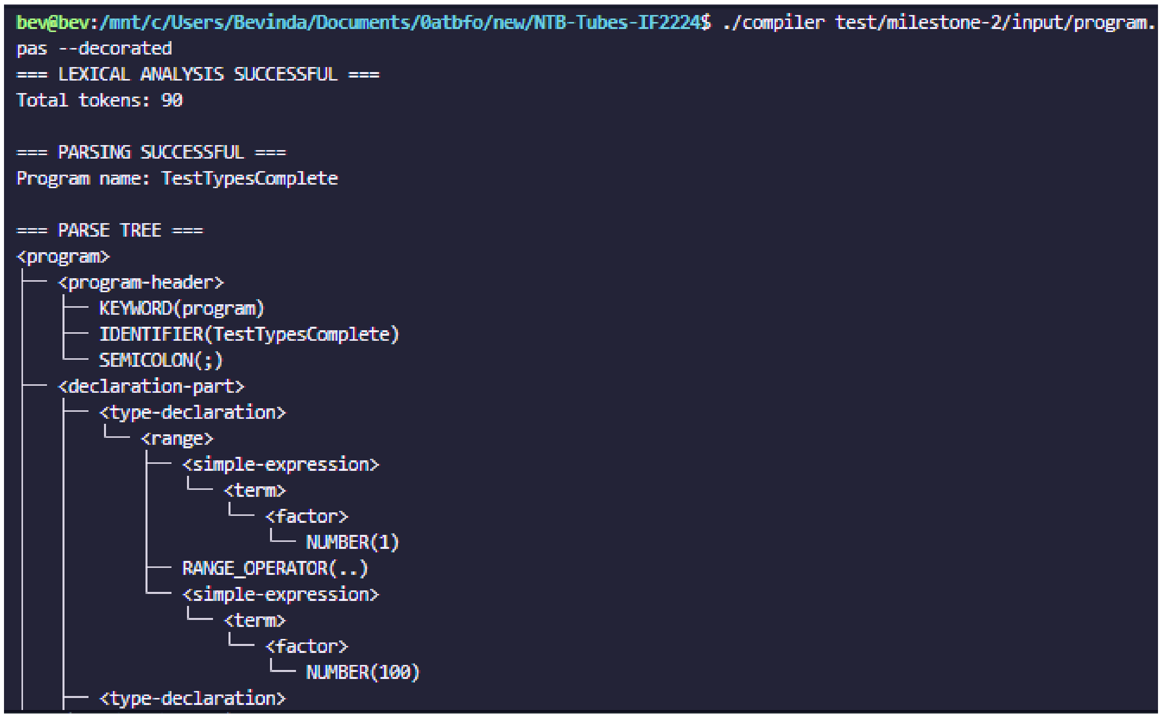
3.2. Test Case 2 - TestTypesComplete

Test case ini bertujuan untuk menguji kemampuan semantic analyzer dalam menangani deklarasi suatu tipe yang dibuat sendiri, array, serta type checking untuk assignment dan for-loop. Dan juga menguji pengisian array table untuk tipe array.

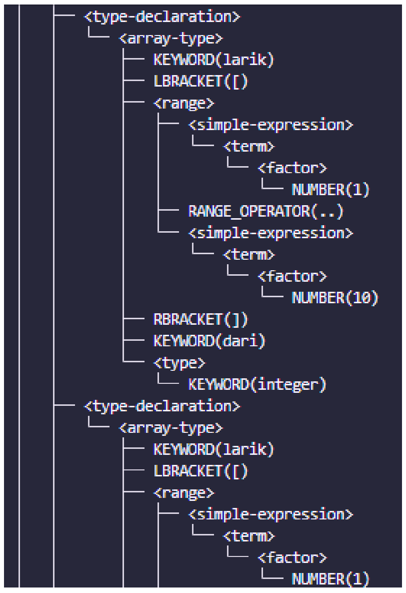
program.pas (Input)
```
program TestTypesComplete;

tipe
    IntRange = 1..100;
    MyArray = larik[1..10] dari integer;
    Matrix = larik[1..5] dari real;

variabel
    x, y: IntRange;
    arr: MyArray;
    mat: Matrix;
    total: integer;

mulai
    x := 10;
    y := 20;
    total := 0;
```

```
    untuk x := 1 ke 10 lakukan mulai
        total := total + x
    selesai;

    writeln('Total: ');
    writeln(total)
selesai.
```

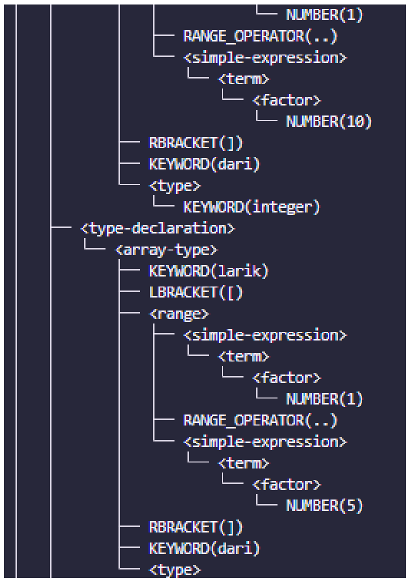
Output:



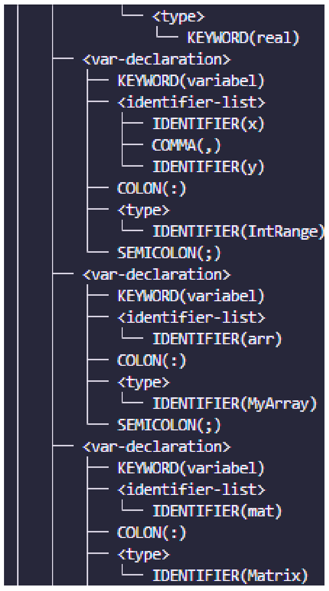
```
bev@bev:/mnt/c/Users/Bevinda/Documents/0atbfo/new/NTB-Tubes-IF2224$ ./compiler test/milestone-2/input/program.
pas --decorated
=== LEXICAL ANALYSIS SUCCESSFUL ===
Total tokens: 90

=== PARSING SUCCESSFUL ===
Program name: TestTypesComplete

=== PARSE TREE ===
<program>
├── <program-header>
│   ├── KEYWORD(program)
│   ├── IDENTIFIER(TestTypesComplete)
│   └── SEMICOLON(;)
├── <declaration-part>
│   ├── <type-declaration>
│   │   └── <range>
│   │       ├── <simple-expression>
│   │       │   └── <term>
│   │       │       └── <factor>
│   │       │           └── NUMBER(1)
│   │       ├── RANGE_OPERATOR(..)
│   │       └── <simple-expression>
│   │           └── <term>
│   │               └── <factor>
│   │                   └── NUMBER(100)
│   ├── <type-declaration>
```



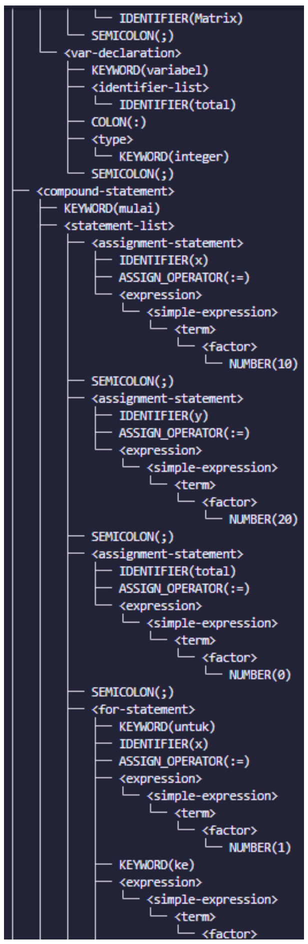
```
│   ├── <type-declaration>
│   │   └── <array-type>
│   │       ├── KEYWORD(larik)
│   │       ├── LBRACKET([)
│   │       ├── <range>
│   │       │   ├── <simple-expression>
│   │       │   │   └── <term>
│   │       │   │       └── <factor>
│   │       │   │           └── NUMBER(1)
│   │       │   ├── RANGE_OPERATOR(..)
│   │       │   └── <simple-expression>
│   │       │       └── <term>
│   │       │           └── <factor>
│   │       │               └── NUMBER(10)
│   │       ├── RBRACKET(])
│   │       ├── KEYWORD(dari)
│   │       └── <type>
│   │           └── KEYWORD(integer)
│   ├── <type-declaration>
│   │   └── <array-type>
│   │       ├── KEYWORD(larik)
│   │       ├── LBRACKET([)
│   │       ├── <range>
│   │       │   ├── <simple-expression>
│   │       │   │   └── <term>
│   │       │   │       └── <factor>
│   │       │   │           └── NUMBER(1)
```



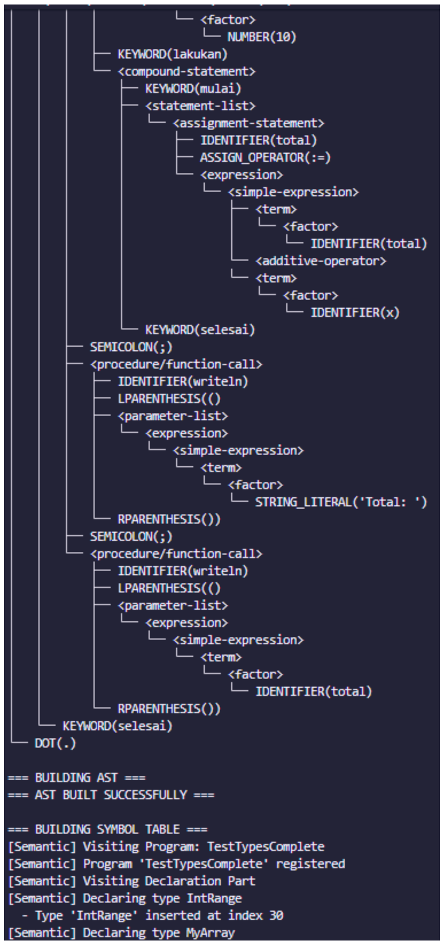
```
│   │       │   │           └── NUMBER(1)
│   │       │   ├── RANGE_OPERATOR(..)
│   │       │   └── <simple-expression>
│   │       │       └── <term>
│   │       │           └── <factor>
│   │       │               └── NUMBER(10)
│   │       ├── RBRACKET(])
│   │       ├── KEYWORD(dari)
│   │       └── <type>
│   │           └── KEYWORD(integer)
│   ├── <type-declaration>
│   │   └── <array-type>
│   │       ├── KEYWORD(larik)
│   │       ├── LBRACKET([)
│   │       ├── <range>
│   │       │   ├── <simple-expression>
│   │       │   │   └── <term>
│   │       │   │       └── <factor>
│   │       │   │           └── NUMBER(1)
│   │       │   ├── RANGE_OPERATOR(..)
│   │       │   └── <simple-expression>
│   │       │       └── <term>
│   │       │           └── <factor>
│   │       │               └── NUMBER(5)
│   │       ├── RBRACKET(])
│   │       ├── KEYWORD(dari)
│   │       └── <type>
```



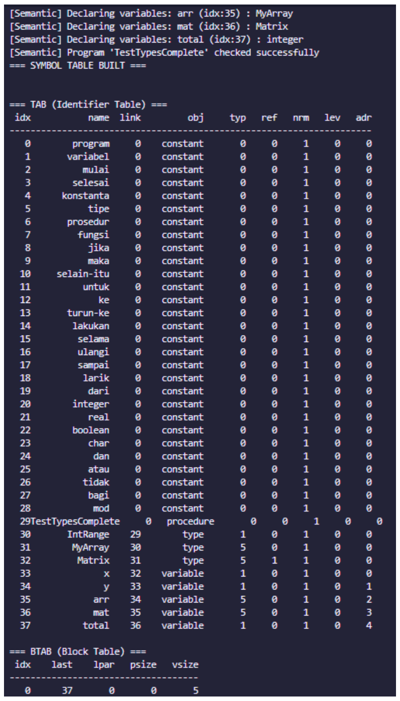
```
│   │       └── <type>
│   │           └── KEYWORD(real)
│   ├── <var-declaration>
│   │   ├── KEYWORD(variabel)
│   │   ├── <identifier-list>
│   │   │   ├── IDENTIFIER(x)
│   │   │   ├── COMMA(,)
│   │   │   └── IDENTIFIER(y)
│   │   ├── COLON(:)
│   │   ├── <type>
│   │   │   └── IDENTIFIER(IntRange)
│   │   └── SEMICOLON(;)
│   ├── <var-declaration>
│   │   ├── KEYWORD(variabel)
│   │   ├── <identifier-list>
│   │   │   └── IDENTIFIER(arr)
│   │   ├── COLON(:)
│   │   ├── <type>
│   │   │   └── IDENTIFIER(MyArray)
│   │   └── SEMICOLON(;)
│   ├── <var-declaration>
│   │   ├── KEYWORD(variabel)
│   │   ├── <identifier-list>
│   │   │   └── IDENTIFIER(mat)
│   │   ├── COLON(:)
│   │   ├── <type>
│   │   │   └── IDENTIFIER(Matrix)
```



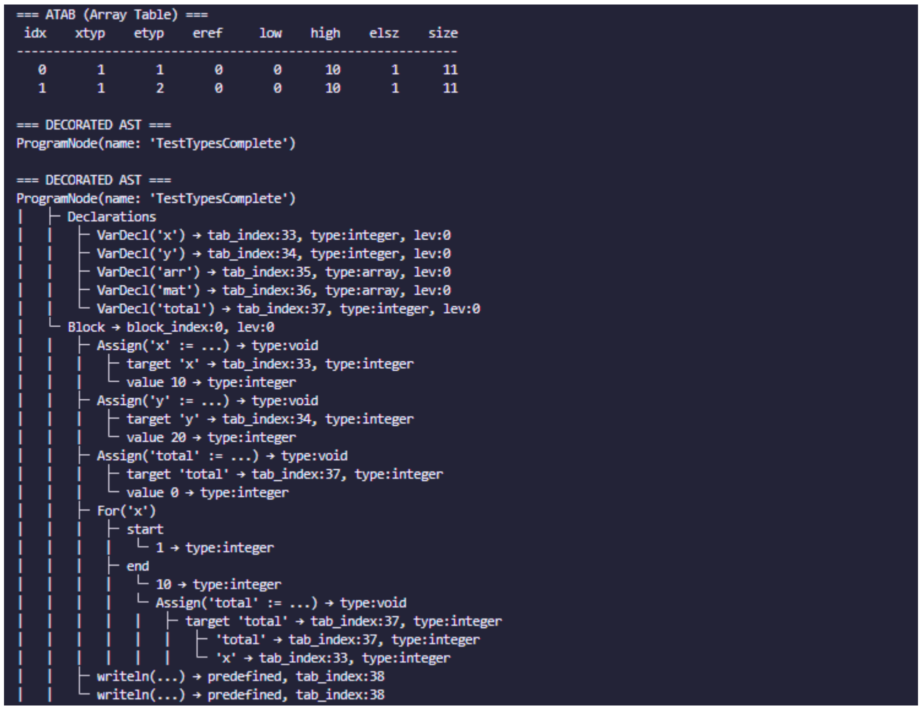
```
│   │   │   └── IDENTIFIER(Matrix)
│   │   └── SEMICOLON(;)
│   └── <var-declaration>
│       ├── KEYWORD(variabel)
│       ├── <identifier-list>
│       │   └── IDENTIFIER(total)
│       ├── COLON(:)
│       ├── <type>
│       │   └── KEYWORD(integer)
│       └── SEMICOLON(;)
├── <compound-statement>
│   ├── KEYWORD(mulai)
│   ├── <statement-list>
│   │   ├── <assignment-statement>
│   │   │   ├── IDENTIFIER(x)
│   │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │   └── <expression>
│   │   │       └── <simple-expression>
│   │   │           └── <term>
│   │   │               └── <factor>
│   │   │                   └── NUMBER(10)
│   │   ├── SEMICOLON(;)
│   │   ├── <assignment-statement>
│   │   │   ├── IDENTIFIER(y)
│   │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │   └── <expression>
│   │   │       └── <simple-expression>
│   │   │           └── <term>
│   │   │               └── <factor>
│   │   │                   └── NUMBER(20)
│   │   ├── SEMICOLON(;)
│   │   ├── <assignment-statement>
│   │   │   ├── IDENTIFIER(total)
│   │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │   └── <expression>
│   │   │       └── <simple-expression>
│   │   │           └── <term>
│   │   │               └── <factor>
│   │   │                   └── NUMBER(0)
│   │   ├── SEMICOLON(;)
│   │   ├── <for-statement>
│   │   │   ├── KEYWORD(untuk)
│   │   │   ├── IDENTIFIER(x)
│   │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │   ├── <expression>
│   │   │   │   └── <simple-expression>
│   │   │   │       └── <term>
│   │   │   │           └── <factor>
│   │   │   │               └── NUMBER(1)
│   │   │   ├── KEYWORD(ke)
│   │   │   ├── <expression>
│   │   │   │   └── <simple-expression>
│   │   │   │       └── <term>
│   │   │   │           └── <factor>
```



```
│   │   │   │           └── <factor>
│   │   │   │               └── NUMBER(10)
│   │   │   ├── KEYWORD(lakukan)
│   │   │   └── <compound-statement>
│   │   │       ├── KEYWORD(mulai)
│   │   │       ├── <statement-list>
│   │   │       │   └── <assignment-statement>
│   │   │       │       ├── IDENTIFIER(total)
│   │   │       │       ├── ASSIGN_OPERATOR(:=)
│   │   │       │       └── <expression>
│   │   │       │           └── <simple-expression>
│   │   │       │               ├── <term>
│   │   │       │               │   └── <factor>
│   │   │       │               │       └── IDENTIFIER(total)
│   │   │       │               ├── <additive-operator>
│   │   │       │               └── <term>
│   │   │       │                   └── <factor>
│   │   │       │                       └── IDENTIFIER(x)
│   │   │       └── KEYWORD(selesai)
│   │   ├── SEMICOLON(;)
│   │   ├── <procedure/function-call>
│   │   │   ├── IDENTIFIER(writeln)
│   │   │   ├── LPARENTHESIS(()
│   │   │   ├── <parameter-list>
│   │   │   │   └── <expression>
│   │   │   │       └── <simple-expression>
│   │   │   │           └── <term>
│   │   │   │               └── <factor>
│   │   │   │                   └── STRING_LITERAL('Total: ')
│   │   │   └── RPARENTHESIS())
│   │   ├── SEMICOLON(;)
│   │   └── <procedure/function-call>
│   │       ├── IDENTIFIER(writeln)
│   │       ├── LPARENTHESIS(()
│   │       ├── <parameter-list>
│   │       │   └── <expression>
│   │       │       └── <simple-expression>
│   │       │           └── <term>
│   │       │               └── <factor>
│   │       │                   └── IDENTIFIER(total)
│   │       └── RPARENTHESIS())
│   └── KEYWORD(selesai)
└── DOT(.)

=== BUILDING AST ===
=== AST BUILT SUCCESSFULLY ===

=== BUILDING SYMBOL TABLE ===
[Semantic] Visiting Program: TestTypesComplete
[Semantic] Program 'TestTypesComplete' registered
[Semantic] Visiting Declaration Part
[Semantic] Declaring type IntRange
  - Type 'IntRange' inserted at index 30
[Semantic] Declaring type MyArray
```



```
[Semantic] Declaring variables: arr (idx:35) : MyArray
[Semantic] Declaring variables: mat (idx:36) : Matrix
[Semantic] Declaring variables: total (idx:37) : integer
[Semantic] Program 'TestTypesComplete' checked successfully
=== SYMBOL TABLE BUILT ===


=== TAB (Identifier Table) ===
 idx          name  link        obj     typ   ref   nrm   lev   adr
-------------------------------------------------------------------
   0       program     0   constant       0     0     1     0     0
   1      variabel     0   constant       0     0     1     0     0
   2         mulai     0   constant       0     0     1     0     0
   3       selesai     0   constant       0     0     1     0     0
   4     konstanta     0   constant       0     0     1     0     0
   5          tipe     0   constant       0     0     1     0     0
   6      prosedur     0   constant       0     0     1     0     0
   7        fungsi     0   constant       0     0     1     0     0
   8          jika     0   constant       0     0     1     0     0
   9          maka     0   constant       0     0     1     0     0
  10    selain-itu     0   constant       0     0     1     0     0
  11         untuk     0   constant       0     0     1     0     0
  12            ke     0   constant       0     0     1     0     0
  13      turun-ke     0   constant       0     0     1     0     0
  14       lakukan     0   constant       0     0     1     0     0
  15        selama     0   constant       0     0     1     0     0
  16        ulangi     0   constant       0     0     1     0     0
  17        sampai     0   constant       0     0     1     0     0
  18         larik     0   constant       0     0     1     0     0
  19          dari     0   constant       0     0     1     0     0
  20       integer     0   constant       0     0     1     0     0
  21          real     0   constant       0     0     1     0     0
  22       boolean     0   constant       0     0     1     0     0
  23          char     0   constant       0     0     1     0     0
  24           dan     0   constant       0     0     1     0     0
  25          atau     0   constant       0     0     1     0     0
  26         tidak     0   constant       0     0     1     0     0
  27          bagi     0   constant       0     0     1     0     0
  28           mod     0   constant       0     0     1     0     0
  29TestTypesComplete     0   procedure       0     0     1     0     0
  30      IntRange    29       type       1     0     1     0     0
  31       MyArray    30       type       5     0     1     0     0
  32        Matrix    31       type       5     1     1     0     0
  33             x    32   variable       1     0     1     0     0
  34             y    33   variable       1     0     1     0     1
  35           arr    34   variable       5     0     1     0     2
  36           mat    35   variable       5     0     1     0     3
  37         total    36   variable       1     0     1     0     4

=== BTAB (Block Table) ===
 idx    last    lpar   psize   vsize
------------------------------------
   0      37       0       0       5
```



```
=== ATAB (Array Table) ===
 idx    xtyp    etyp    eref     low    high    elsz    size
-------------------------------------------------------------
   0       1       1       0       0      10       1      11
   1       1       2       0       0      10       1      11

=== DECORATED AST ===
ProgramNode(name: 'TestTypesComplete')

=== DECORATED AST ===
ProgramNode(name: 'TestTypesComplete')
│   ├─ Declarations
│   │   ├─ VarDecl('x') → tab_index:33, type:integer, lev:0
│   │   ├─ VarDecl('y') → tab_index:34, type:integer, lev:0
│   │   ├─ VarDecl('arr') → tab_index:35, type:array, lev:0
│   │   ├─ VarDecl('mat') → tab_index:36, type:array, lev:0
│   │   ├─ VarDecl('total') → tab_index:37, type:integer, lev:0
│   └─ Block → block_index:0, lev:0
│   │   ├─ Assign('x' := ...) → type:void
│   │   │   ├─ target 'x' → tab_index:33, type:integer
│   │   │   └─ value 10 → type:integer
│   │   ├─ Assign('y' := ...) → type:void
│   │   │   ├─ target 'y' → tab_index:34, type:integer
│   │   │   └─ value 20 → type:integer
│   │   ├─ Assign('total' := ...) → type:void
│   │   │   ├─ target 'total' → tab_index:37, type:integer
│   │   │   └─ value 0 → type:integer
│   │   ├─ For('x')
│   │   │   ├─ start
│   │   │   │   └─ 1 → type:integer
│   │   │   ├─ end
│   │   │   │   └─ 10 → type:integer
│   │   │   │   └─ Assign('total' := ...) → type:void
│   │   │   │   │   ├─ target 'total' → tab_index:37, type:integer
│   │   │   │   │   │   ├─ 'total' → tab_index:37, type:integer
│   │   │   │   │   │   └─ 'x' → tab_index:33, type:integer
│   │   ├─ writeln(...) → predefined, tab_index:38
│   │   └─ writeln(...) → predefined, tab_index:38
```

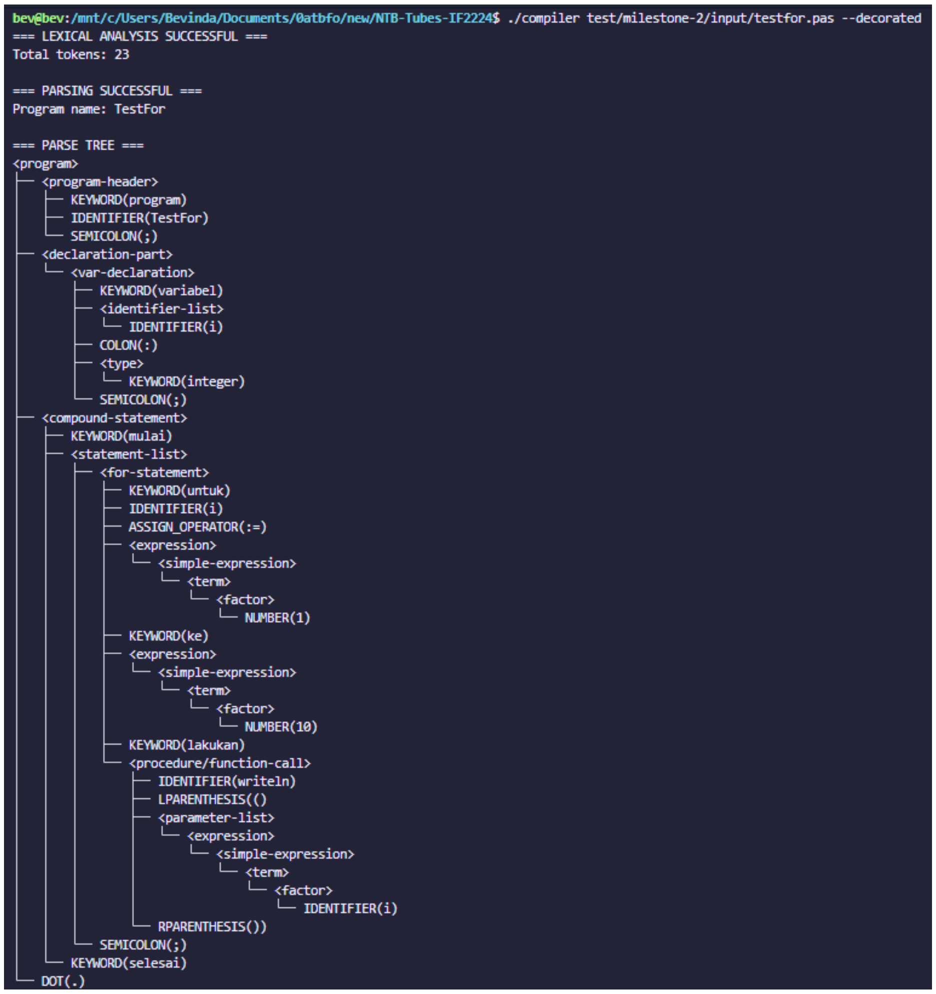
3.3. Test Case 3 - TestFor

Test case ini bertujuan untuk menguji validasi for-loop dengan direction ke (to) dan body berupa single statement (procedure call). Memastikan parser dapat menangani for-loop tanpa compound statement dan parameter passing ke procedure built-in.

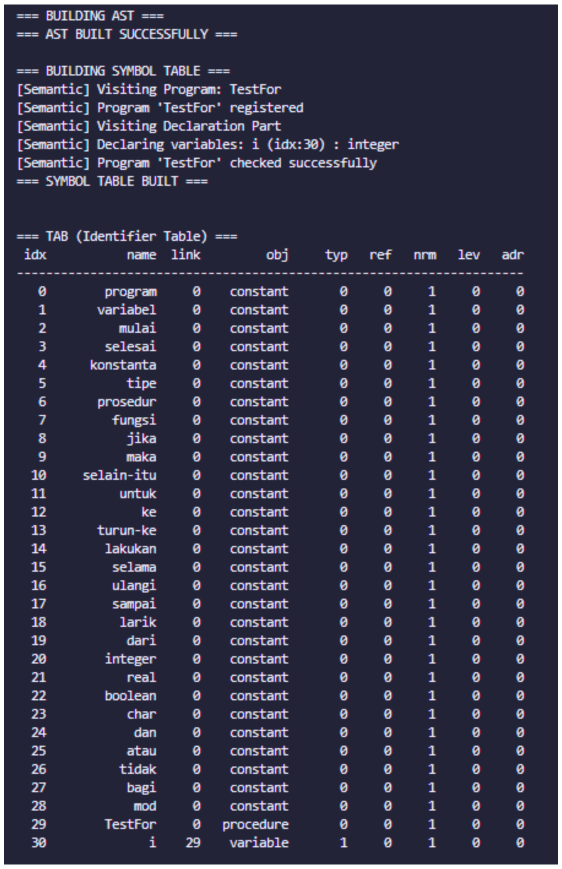
testfor.pas (Input)

`program TestFor;`
`variabel i: integer;`
`mulai`
`  untuk i := 1 ke 10 lakukan`
`    writeln(i);`
`selesai.`

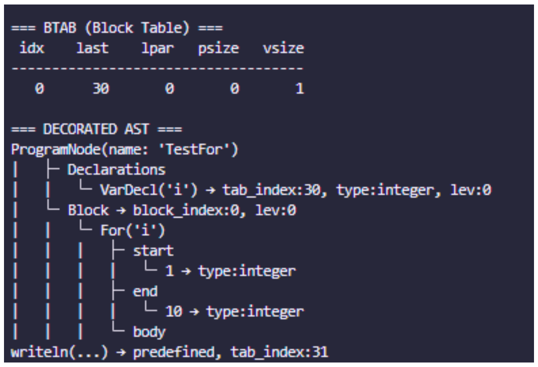
Output:



```
bev@bev:/mnt/c/Users/Bevinda/Documents/0atbfo/new/NTB-Tubes-IF2224$ ./compiler test/milestone-2/input/testfor.pas --decorated
=== LEXICAL ANALYSIS SUCCESSFUL ===
Total tokens: 23

=== PARSING SUCCESSFUL ===
Program name: TestFor

=== PARSE TREE ===
<program>
├── <program-header>
│   ├── KEYWORD(program)
│   ├── IDENTIFIER(TestFor)
│   └── SEMICOLON(;)
├── <declaration-part>
│   └── <var-declaration>
│       ├── KEYWORD(variabel)
│       ├── <identifier-list>
│       │   └── IDENTIFIER(i)
│       ├── COLON(:)
│       ├── <type>
│       │   └── KEYWORD(integer)
│       └── SEMICOLON(;)
├── <compound-statement>
│   ├── KEYWORD(mulai)
│   ├── <statement-list>
│   │   ├── <for-statement>
│   │   │   ├── KEYWORD(untuk)
│   │   │   ├── IDENTIFIER(i)
│   │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │   ├── <expression>
│   │   │   │   └── <simple-expression>
│   │   │   │       └── <term>
│   │   │   │           └── <factor>
│   │   │   │               └── NUMBER(1)
│   │   │   ├── KEYWORD(ke)
│   │   │   ├── <expression>
│   │   │   │   └── <simple-expression>
│   │   │   │       └── <term>
│   │   │   │           └── <factor>
│   │   │   │               └── NUMBER(10)
│   │   │   ├── KEYWORD(lakukan)
│   │   │   └── <procedure/function-call>
│   │   │       ├── IDENTIFIER(writeln)
│   │   │       ├── LPARENTHESIS(()
│   │   │       ├── <parameter-list>
│   │   │       │   └── <expression>
│   │   │       │       └── <simple-expression>
│   │   │       │           └── <term>
│   │   │       │               └── <factor>
│   │   │       │                   └── IDENTIFIER(i)
│   │   │       └── RPARENTHESIS())
│   │   └── SEMICOLON(;)
│   └── KEYWORD(selesai)
└── DOT(.)
```



```
=== BUILDING AST ===
=== AST BUILT SUCCESSFULLY ===

=== BUILDING SYMBOL TABLE ===
[Semantic] Visiting Program: TestFor
[Semantic] Program 'TestFor' registered
[Semantic] Visiting Declaration Part
[Semantic] Declaring variables: i (idx:30) : integer
[Semantic] Program 'TestFor' checked successfully
=== SYMBOL TABLE BUILT ===


=== TAB (Identifier Table) ===
 idx           name  link        obj     typ   ref   nrm   lev   adr
--------------------------------------------------------------------
   0        program    0   constant       0     0     1     0     0
   1       variabel    0   constant       0     0     1     0     0
   2          mulai    0   constant       0     0     1     0     0
   3        selesai    0   constant       0     0     1     0     0
   4      konstanta    0   constant       0     0     1     0     0
   5           tipe    0   constant       0     0     1     0     0
   6       prosedur    0   constant       0     0     1     0     0
   7         fungsi    0   constant       0     0     1     0     0
   8           jika    0   constant       0     0     1     0     0
   9           maka    0   constant       0     0     1     0     0
  10      selain-itu    0   constant       0     0     1     0     0
  11          untuk    0   constant       0     0     1     0     0
  12             ke    0   constant       0     0     1     0     0
  13        turun-ke    0   constant       0     0     1     0     0
  14        lakukan    0   constant       0     0     1     0     0
  15         selama    0   constant       0     0     1     0     0
  16         ulangi    0   constant       0     0     1     0     0
  17         sampai    0   constant       0     0     1     0     0
  18          larik    0   constant       0     0     1     0     0
  19           dari    0   constant       0     0     1     0     0
  20        integer    0   constant       0     0     1     0     0
  21           real    0   constant       0     0     1     0     0
  22        boolean    0   constant       0     0     1     0     0
  23           char    0   constant       0     0     1     0     0
  24            dan    0   constant       0     0     1     0     0
  25           atau    0   constant       0     0     1     0     0
  26          tidak    0   constant       0     0     1     0     0
  27           bagi    0   constant       0     0     1     0     0
  28            mod    0   constant       0     0     1     0     0
  29        TestFor    0  procedure       0     0     1     0     0
  30              i   29   variable       1     0     1     0     0
```



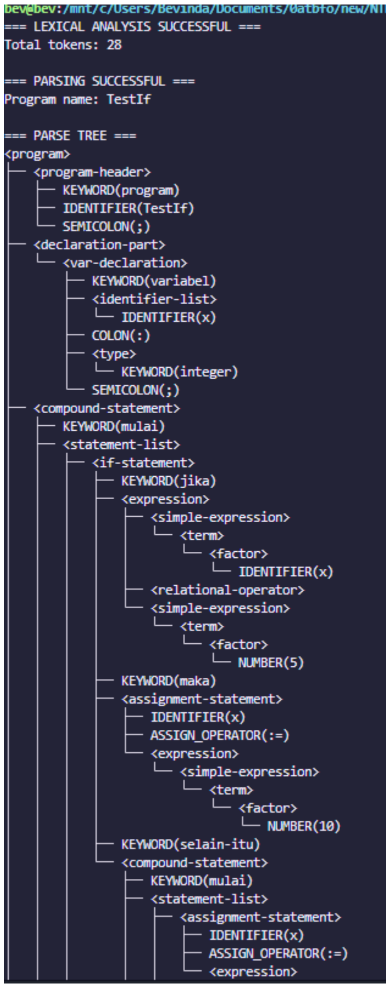
```
=== BTAB (Block Table) ===
 idx    last    lpar   psize   vsize
------------------------------------
   0      30       0       0       1

=== DECORATED AST ===
ProgramNode(name: 'TestFor')
|   ├─ Declarations
|   |   └─ VarDecl('i') → tab_index:30, type:integer, lev:0
|   └─ Block → block_index:0, lev:0
|   |   └─ For('i')
|   |   |   ├─ start
|   |   |   |   └─ 1 → type:integer
|   |   |   ├─ end
|   |   |   |   └─ 10 → type:integer
|   |   |   └─ body
writeln(...) → predefined, tab_index:31
```

3.4. Test Case 4 - TestIf

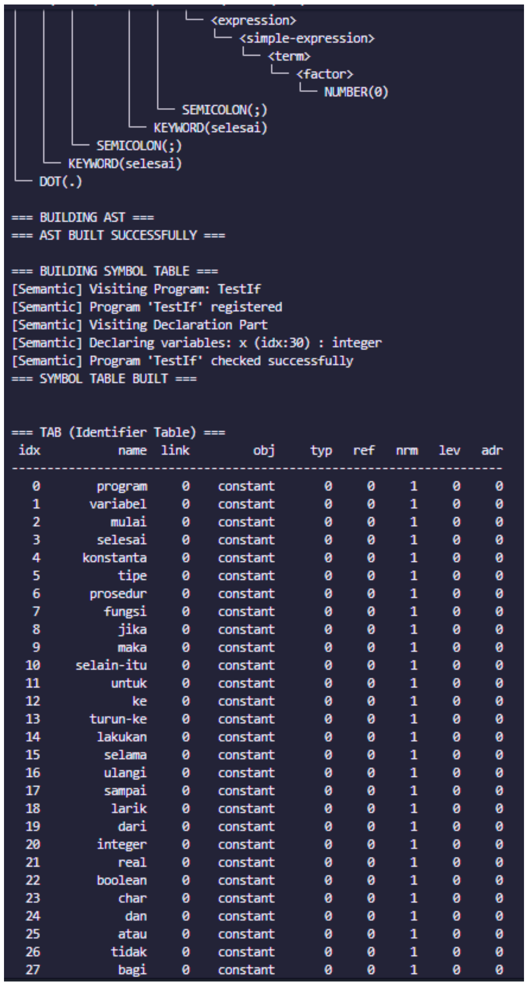
Test case ini bertujuan untuk memvalidasi if-statement dengan kondisi relasional, then-branch berupa single statement, dan else-branch berupa compound statement. Menguji kemampuan parser mengenali keyword selain-itu (else).

testif.pas
```
program TestIf;
variabel x: integer;
mulai
  jika x > 5 maka
      x := 10;
  selain-itu
    mulai
      x := 0;
    selesai;
selesai.
```

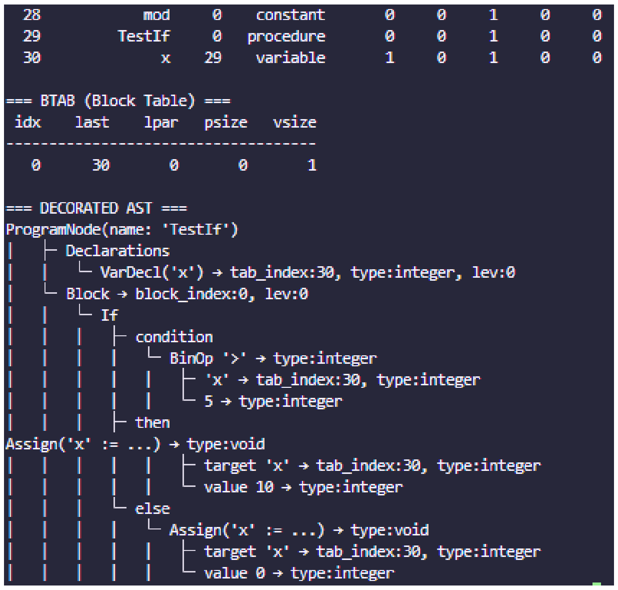
Output:



```
bev@bev:/mnt/c/Users/Bevinda/Documents/0atbfo/new/NI
=== LEXICAL ANALYSIS SUCCESSFUL ===
Total tokens: 28

=== PARSING SUCCESSFUL ===
Program name: TestIf

=== PARSE TREE ===
<program>
├── <program-header>
│   ├── KEYWORD(program)
│   ├── IDENTIFIER(TestIf)
│   └── SEMICOLON(;)
├── <declaration-part>
│   └── <var-declaration>
│       ├── KEYWORD(variabel)
│       ├── <identifier-list>
│       │   └── IDENTIFIER(x)
│       ├── COLON(:)
│       ├── <type>
│       │   └── KEYWORD(integer)
│       └── SEMICOLON(;)
├── <compound-statement>
│   ├── KEYWORD(mulai)
│   ├── <statement-list>
│   │   ├── <if-statement>
│   │   │   ├── KEYWORD(jika)
│   │   │   ├── <expression>
│   │   │   │   ├── <simple-expression>
│   │   │   │   │   └── <term>
│   │   │   │   │       └── <factor>
│   │   │   │   │           └── IDENTIFIER(x)
│   │   │   │   ├── <relational-operator>
│   │   │   │   └── <simple-expression>
│   │   │   │       └── <term>
│   │   │   │           └── <factor>
│   │   │   │               └── NUMBER(5)
│   │   │   ├── KEYWORD(maka)
│   │   │   ├── <assignment-statement>
│   │   │   │   ├── IDENTIFIER(x)
│   │   │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │   │   └── <expression>
│   │   │   │       └── <simple-expression>
│   │   │   │           └── <term>
│   │   │   │               └── <factor>
│   │   │   │                   └── NUMBER(10)
│   │   │   ├── KEYWORD(selain-itu)
│   │   │   └── <compound-statement>
│   │   │       ├── KEYWORD(mulai)
│   │   │       ├── <statement-list>
│   │   │       │   ├── <assignment-statement>
│   │   │       │   │   ├── IDENTIFIER(x)
│   │   │       │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │       │   │   └── <expression>
```



```
│   │   │       │   │   └── <expression>
│   │   │       │   │       └── <simple-expression>
│   │   │       │   │           └── <term>
│   │   │       │   │               └── <factor>
│   │   │       │   │                   └── NUMBER(0)
│   │   │       │   └── SEMICOLON(;)
│   │   │       └── KEYWORD(selesai)
│   │   └── SEMICOLON(;)
│   └── KEYWORD(selesai)
└── DOT(.)

=== BUILDING AST ===
=== AST BUILT SUCCESSFULLY ===

=== BUILDING SYMBOL TABLE ===
[Semantic] Visiting Program: TestIf
[Semantic] Program 'TestIf' registered
[Semantic] Visiting Declaration Part
[Semantic] Declaring variables: x (idx:30) : integer
[Semantic] Program 'TestIf' checked successfully
=== SYMBOL TABLE BUILT ===


=== TAB (Identifier Table) ===
 idx          name  link        obj     typ   ref   nrm   lev   adr
---------------------------------------------------------------------
   0       program     0   constant       0     0     1     0     0
   1      variabel     0   constant       0     0     1     0     0
   2         mulai     0   constant       0     0     1     0     0
   3       selesai     0   constant       0     0     1     0     0
   4     konstanta     0   constant       0     0     1     0     0
   5          tipe     0   constant       0     0     1     0     0
   6      prosedur     0   constant       0     0     1     0     0
   7        fungsi     0   constant       0     0     1     0     0
   8          jika     0   constant       0     0     1     0     0
   9          maka     0   constant       0     0     1     0     0
  10    selain-itu     0   constant       0     0     1     0     0
  11         untuk     0   constant       0     0     1     0     0
  12            ke     0   constant       0     0     1     0     0
  13      turun-ke     0   constant       0     0     1     0     0
  14       lakukan     0   constant       0     0     1     0     0
  15        selama     0   constant       0     0     1     0     0
  16        ulangi     0   constant       0     0     1     0     0
  17        sampai     0   constant       0     0     1     0     0
  18         larik     0   constant       0     0     1     0     0
  19          dari     0   constant       0     0     1     0     0
  20       integer     0   constant       0     0     1     0     0
  21          real     0   constant       0     0     1     0     0
  22       boolean     0   constant       0     0     1     0     0
  23          char     0   constant       0     0     1     0     0
  24           dan     0   constant       0     0     1     0     0
  25          atau     0   constant       0     0     1     0     0
  26         tidak     0   constant       0     0     1     0     0
  27          bagi     0   constant       0     0     1     0     0
```



```
  28           mod     0   constant       0     0     1     0     0
  29        TestIf     0  procedure       0     0     1     0     0
  30             x    29   variable       1     0     1     0     0

=== BTAB (Block Table) ===
 idx    last    lpar   psize   vsize
------------------------------------
   0      30       0       0       1

=== DECORATED AST ===
ProgramNode(name: 'TestIf')
|    ├ Declarations
|    |    └ VarDecl('x') → tab_index:30, type:integer, lev:0
|    └ Block → block_index:0, lev:0
|    |    └ If
|    |    |    ├ condition
|    |    |    |    └ BinOp '>' → type:integer
|    |    |    |    |    ├ 'x' → tab_index:30, type:integer
|    |    |    |    |    └ 5 → type:integer
|    |    |    ├ then
Assign('x' := ...) → type:void
|    |    |    |    |    ├ target 'x' → tab_index:30, type:integer
|    |    |    |    |    └ value 10 → type:integer
|    |    |    └ else
|    |    |    |    └ Assign('x' := ...) → type:void
|    |    |    |    |    ├ target 'x' → tab_index:30, type:integer
|    |    |    |    |    └ value 0 → type:integer
```

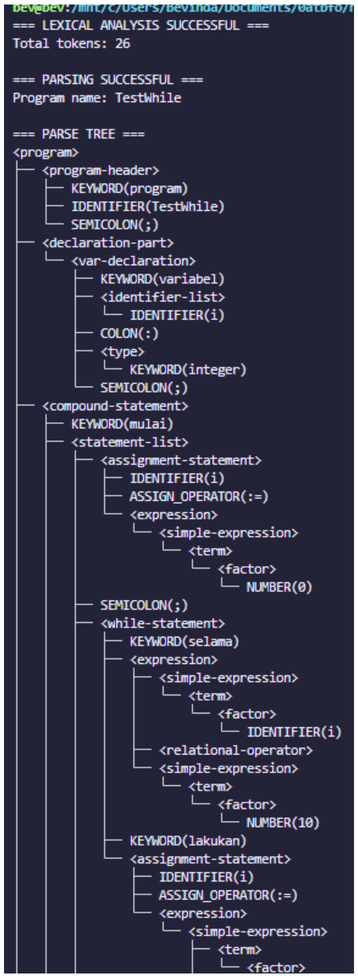
3.5. Test Case 5 - TestWhile

Test case ini bertujuan untuk memvalidasi while-loop dengan keyword selama dan lakukan, termasuk kondisi relasional dan body berupa single assignment statement dengan ekspresi aritmatika.

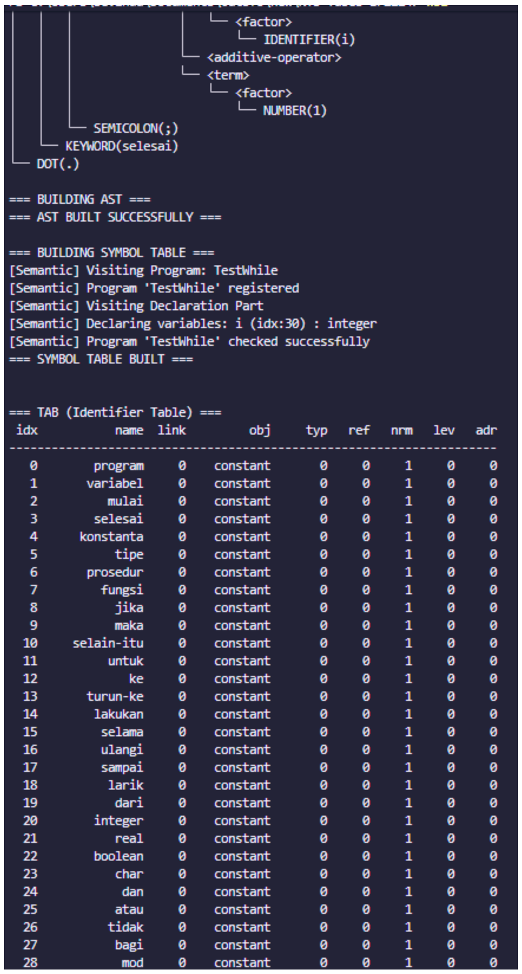
testwhile.pas

```
program TestWhile;
variabel i: integer;
mulai
  i := 0;
  selama i < 10 lakukan
    i := i + 1;
selesai.
```

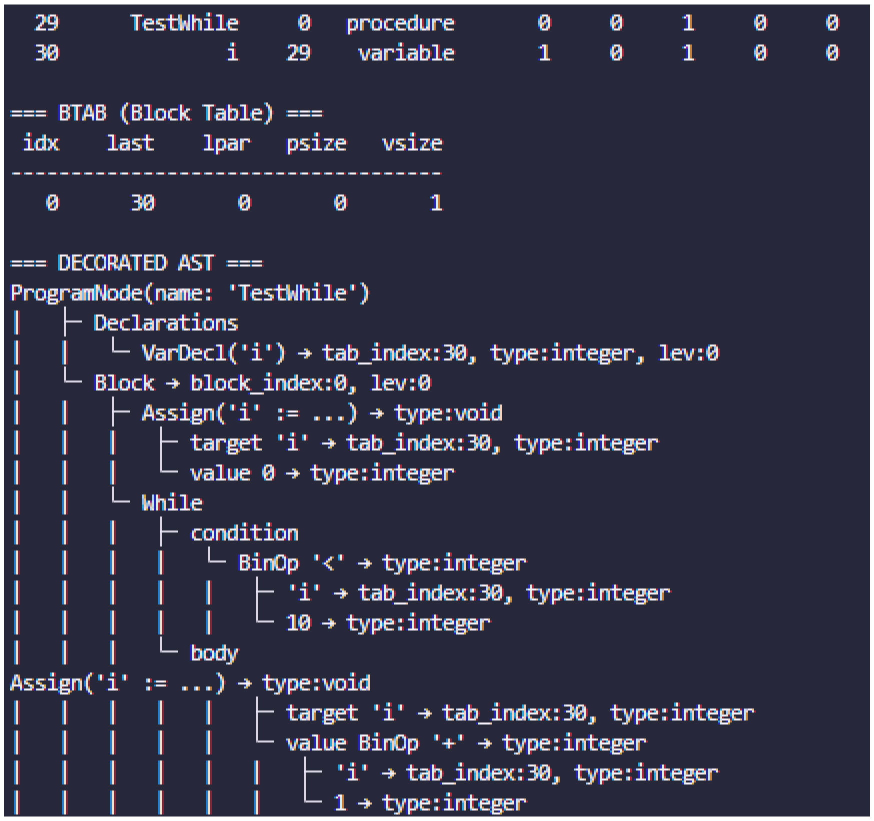
Output:



```
bev@bev:/mnt/c/Users/Bevinda/Documents/0atbfo/
=== LEXICAL ANALYSIS SUCCESSFUL ===
Total tokens: 26

=== PARSING SUCCESSFUL ===
Program name: TestWhile

=== PARSE TREE ===
<program>
├── <program-header>
│   ├── KEYWORD(program)
│   ├── IDENTIFIER(TestWhile)
│   └── SEMICOLON(;)
├── <declaration-part>
│   └── <var-declaration>
│       ├── KEYWORD(variabel)
│       ├── <identifier-list>
│       │   └── IDENTIFIER(i)
│       ├── COLON(:)
│       ├── <type>
│       │   └── KEYWORD(integer)
│       └── SEMICOLON(;)
├── <compound-statement>
│   ├── KEYWORD(mulai)
│   ├── <statement-list>
│   │   ├── <assignment-statement>
│   │   │   ├── IDENTIFIER(i)
│   │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │   └── <expression>
│   │   │       └── <simple-expression>
│   │   │           └── <term>
│   │   │               └── <factor>
│   │   │                   └── NUMBER(0)
│   │   ├── SEMICOLON(;)
│   │   ├── <while-statement>
│   │   │   ├── KEYWORD(selama)
│   │   │   ├── <expression>
│   │   │   │   ├── <simple-expression>
│   │   │   │   │   └── <term>
│   │   │   │   │       └── <factor>
│   │   │   │   │           └── IDENTIFIER(i)
│   │   │   │   ├── <relational-operator>
│   │   │   │   └── <simple-expression>
│   │   │   │       └── <term>
│   │   │   │           └── <factor>
│   │   │   │               └── NUMBER(10)
│   │   │   ├── KEYWORD(lakukan)
│   │   │   └── <assignment-statement>
│   │   │       ├── IDENTIFIER(i)
│   │   │       ├── ASSIGN_OPERATOR(:=)
│   │   │       └── <expression>
│   │   │           └── <simple-expression>
│   │   │               ├── <term>
│   │   │               │   └── <factor>
```



```
│   │   │               │   └── <factor>
│   │   │               │       └── IDENTIFIER(i)
│   │   │               ├── <additive-operator>
│   │   │               └── <term>
│   │   │                   └── <factor>
│   │   │                       └── NUMBER(1)
│   │   └── SEMICOLON(;)
│   └── KEYWORD(selesai)
└── DOT(.)

=== BUILDING AST ===
=== AST BUILT SUCCESSFULLY ===

=== BUILDING SYMBOL TABLE ===
[Semantic] Visiting Program: TestWhile
[Semantic] Program 'TestWhile' registered
[Semantic] Visiting Declaration Part
[Semantic] Declaring variables: i (idx:30) : integer
[Semantic] Program 'TestWhile' checked successfully
=== SYMBOL TABLE BUILT ===


=== TAB (Identifier Table) ===
 idx          name  link        obj     typ   ref   nrm   lev   adr
--------------------------------------------------------------------
   0       program     0   constant       0     0     1     0     0
   1      variabel     0   constant       0     0     1     0     0
   2         mulai     0   constant       0     0     1     0     0
   3       selesai     0   constant       0     0     1     0     0
   4     konstanta     0   constant       0     0     1     0     0
   5          tipe     0   constant       0     0     1     0     0
   6      prosedur     0   constant       0     0     1     0     0
   7        fungsi     0   constant       0     0     1     0     0
   8          jika     0   constant       0     0     1     0     0
   9          maka     0   constant       0     0     1     0     0
  10    selain-itu     0   constant       0     0     1     0     0
  11         untuk     0   constant       0     0     1     0     0
  12            ke     0   constant       0     0     1     0     0
  13      turun-ke     0   constant       0     0     1     0     0
  14       lakukan     0   constant       0     0     1     0     0
  15        selama     0   constant       0     0     1     0     0
  16        ulangi     0   constant       0     0     1     0     0
  17        sampai     0   constant       0     0     1     0     0
  18         larik     0   constant       0     0     1     0     0
  19          dari     0   constant       0     0     1     0     0
  20       integer     0   constant       0     0     1     0     0
  21          real     0   constant       0     0     1     0     0
  22       boolean     0   constant       0     0     1     0     0
  23          char     0   constant       0     0     1     0     0
  24           dan     0   constant       0     0     1     0     0
  25          atau     0   constant       0     0     1     0     0
  26         tidak     0   constant       0     0     1     0     0
  27          bagi     0   constant       0     0     1     0     0
  28           mod     0   constant       0     0     1     0     0
```


```
  29      TestWhile     0   procedure       0     0     1     0     0
  30              i    29    variable       1     0     1     0     0

=== BTAB (Block Table) ===
 idx    last    lpar   psize   vsize
------------------------------------
   0      30       0       0       1

=== DECORATED AST ===
ProgramNode(name: 'TestWhile')
|   ├ Declarations
|   |   └ VarDecl('i') → tab_index:30, type:integer, lev:0
|   └ Block → block_index:0, lev:0
|   |   ├ Assign('i' := ...) → type:void
|   |   |   ├ target 'i' → tab_index:30, type:integer
|   |   |   └ value 0 → type:integer
|   |   └ While
|   |   |   ├ condition
|   |   |   |   └ BinOp '<' → type:integer
|   |   |   |   |   ├ 'i' → tab_index:30, type:integer
|   |   |   |   |   └ 10 → type:integer
|   |   |   └ body
Assign('i' := ...) → type:void
|   |   |   |   |   ├ target 'i' → tab_index:30, type:integer
|   |   |   |   |   └ value BinOp '+' → type:integer
|   |   |   |   |   |   ├ 'i' → tab_index:30, type:integer
|   |   |   |   |   |   └ 1 → type:integer
```

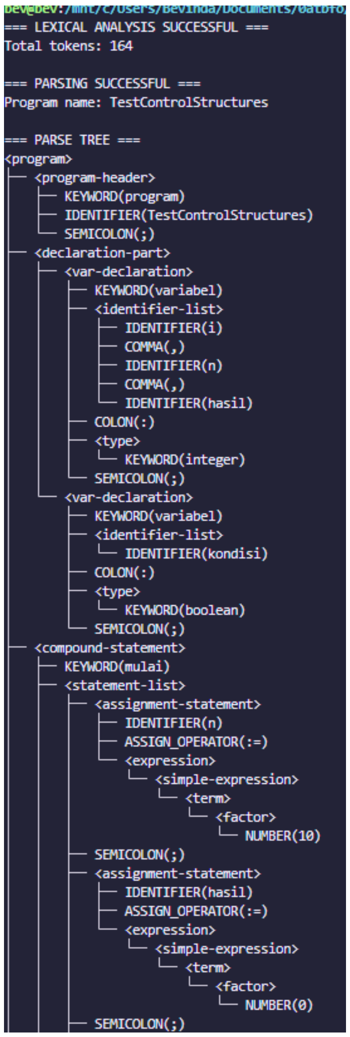
3.6. Test Case 6 - TestControlStructures

Test case ini bertujuan untuk menguji manajemen multiple scopes dengan nested if, while, dan for loops. Memvalidasi bahwa semantic analyzer dapat menangani push/pop scope dengan benar dan melakukan lookup identifier di multiple levels.

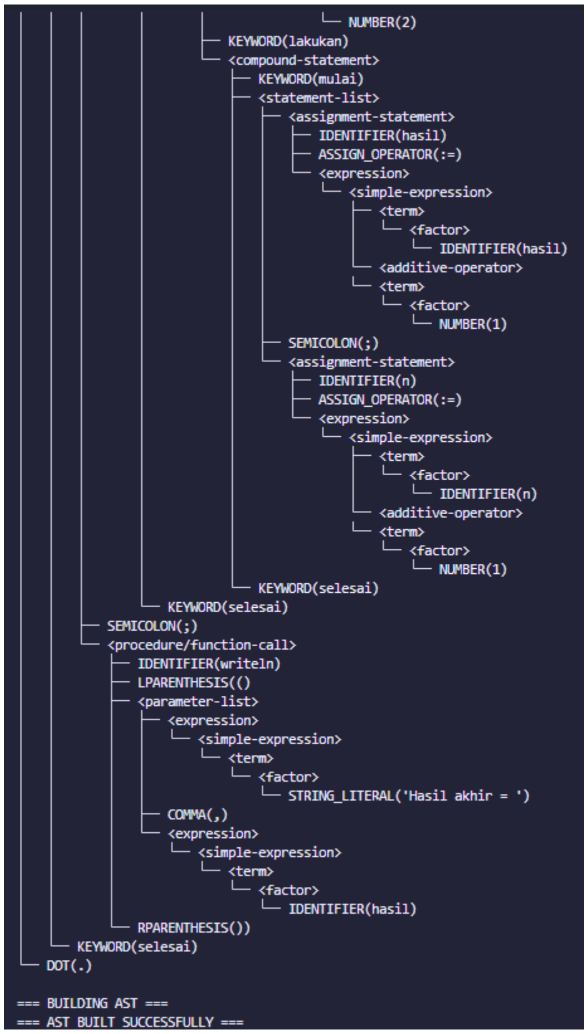
test_control_structures.pas

```
program TestControlStructures;

variabel
  i, n, hasil: integer;
  kondisi: boolean;

mulai
  n := 10;
  hasil := 0;
  jika n > 5 maka
    hasil := n * 2;
  jika n < 5 maka
    hasil := n + 10
  selain-itu
    hasil := n - 5;
  jika n > 0 maka
    jika n < 100 maka
      hasil := n
    selain-itu
      hasil := 100
  selain-itu
    hasil := 0;
  i := 1;
  selama i <= 5 lakukan
    mulai
      hasil := hasil + i;
      i := i + 1
    selesai;
  untuk i := 1 ke 10 lakukan
    hasil := hasil + i;
  untuk i := 10 turun-ke 1 lakukan
    hasil := hasil - 1;
  untuk i := 1 ke 3 lakukan
    mulai
      n := 1;
      selama n <= 2 lakukan
        mulai
          hasil := hasil + 1;
          n := n + 1
        selesai
    selesai;
  writeln('Hasil akhir = ', hasil)
selesai.
```

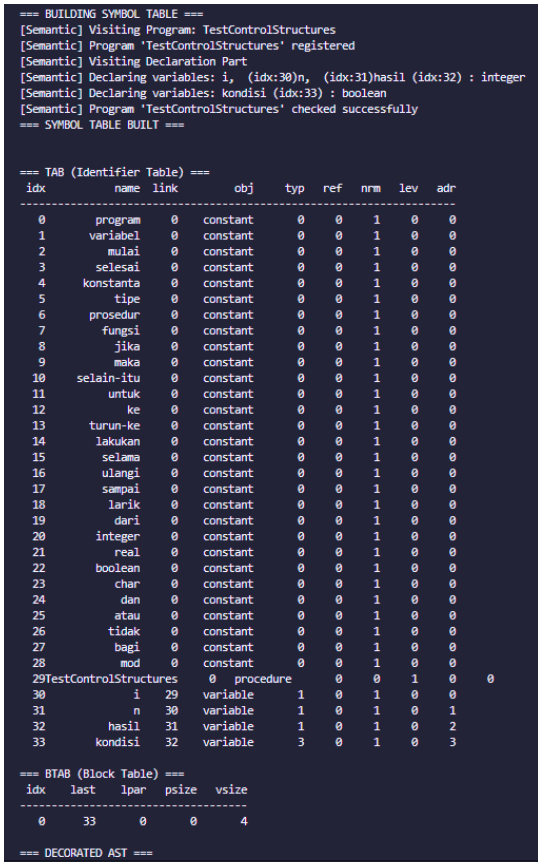
Output:



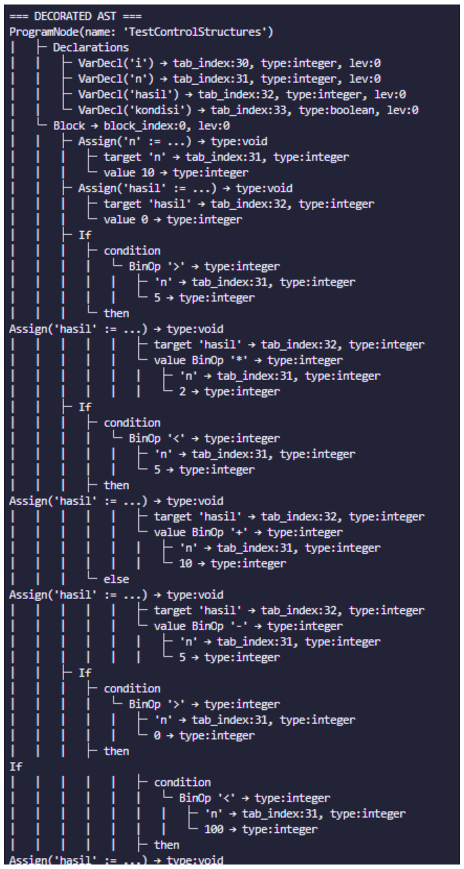
```
bev@bev:/mnt/c/Users/Bevinda/Documents/0atbfo
=== LEXICAL ANALYSIS SUCCESSFUL ===
Total tokens: 164

=== PARSING SUCCESSFUL ===
Program name: TestControlStructures

=== PARSE TREE ===
<program>
├── <program-header>
│   ├── KEYWORD(program)
│   ├── IDENTIFIER(TestControlStructures)
│   └── SEMICOLON(;)
├── <declaration-part>
│   ├── <var-declaration>
│   │   ├── KEYWORD(variabel)
│   │   ├── <identifier-list>
│   │   │   ├── IDENTIFIER(i)
│   │   │   ├── COMMA(,)
│   │   │   ├── IDENTIFIER(n)
│   │   │   ├── COMMA(,)
│   │   │   └── IDENTIFIER(hasil)
│   │   ├── COLON(:)
│   │   ├── <type>
│   │   │   └── KEYWORD(integer)
│   │   └── SEMICOLON(;)
│   └── <var-declaration>
│       ├── KEYWORD(variabel)
│       ├── <identifier-list>
│       │   └── IDENTIFIER(kondisi)
│       ├── COLON(:)
│       ├── <type>
│       │   └── KEYWORD(boolean)
│       └── SEMICOLON(;)
├── <compound-statement>
│   ├── KEYWORD(mulai)
│   ├── <statement-list>
│   │   ├── <assignment-statement>
│   │   │   ├── IDENTIFIER(n)
│   │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │   └── <expression>
│   │   │       └── <simple-expression>
│   │   │           └── <term>
│   │   │               └── <factor>
│   │   │                   └── NUMBER(10)
│   │   ├── SEMICOLON(;)
│   │   ├── <assignment-statement>
│   │   │   ├── IDENTIFIER(hasil)
│   │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │   └── <expression>
│   │   │       └── <simple-expression>
│   │   │           └── <term>
│   │   │               └── <factor>
│   │   │                   └── NUMBER(0)
│   │   ├── SEMICOLON(;)
```



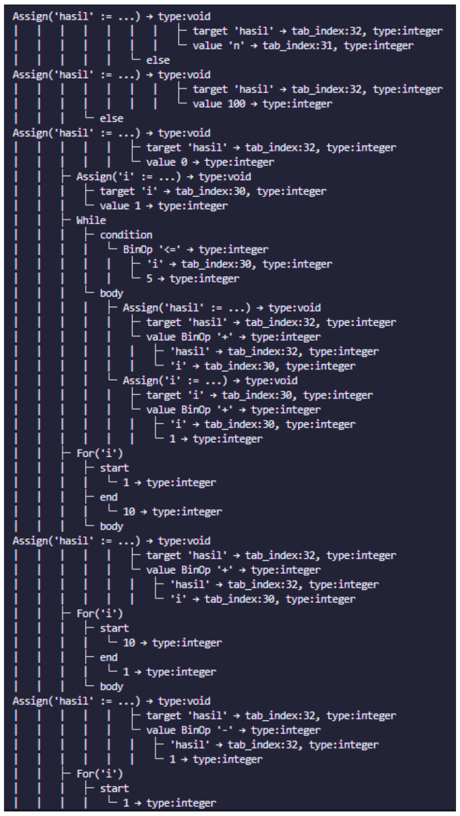
```
│   │   │       │       │               └── NUMBER(2)
│   │   │       │       ├── KEYWORD(lakukan)
│   │   │       │       └── <compound-statement>
│   │   │       │           ├── KEYWORD(mulai)
│   │   │       │           ├── <statement-list>
│   │   │       │           │   ├── <assignment-statement>
│   │   │       │           │   │   ├── IDENTIFIER(hasil)
│   │   │       │           │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │       │           │   │   └── <expression>
│   │   │       │           │   │       └── <simple-expression>
│   │   │       │           │   │           ├── <term>
│   │   │       │           │   │           │   └── <factor>
│   │   │       │           │   │           │       └── IDENTIFIER(hasil)
│   │   │       │           │   │           ├── <additive-operator>
│   │   │       │           │   │           └── <term>
│   │   │       │           │   │               └── <factor>
│   │   │       │           │   │                   └── NUMBER(1)
│   │   │       │           │   ├── SEMICOLON(;)
│   │   │       │           │   └── <assignment-statement>
│   │   │       │           │       ├── IDENTIFIER(n)
│   │   │       │           │       ├── ASSIGN_OPERATOR(:=)
│   │   │       │           │       └── <expression>
│   │   │       │           │           └── <simple-expression>
│   │   │       │           │               ├── <term>
│   │   │       │           │               │   └── <factor>
│   │   │       │           │               │       └── IDENTIFIER(n)
│   │   │       │           │               ├── <additive-operator>
│   │   │       │           │               └── <term>
│   │   │       │           │                   └── <factor>
│   │   │       │           │                       └── NUMBER(1)
│   │   │       │           └── KEYWORD(selesai)
│   │   │       └── KEYWORD(selesai)
│   │   ├── SEMICOLON(;)
│   │   └── <procedure/function-call>
│   │       ├── IDENTIFIER(writeln)
│   │       ├── LPARENTHESIS(()
│   │       ├── <parameter-list>
│   │       │   ├── <expression>
│   │       │   │   └── <simple-expression>
│   │       │   │       └── <term>
│   │       │   │           └── <factor>
│   │       │   │               └── STRING_LITERAL('Hasil akhir = ')
│   │       │   ├── COMMA(,)
│   │       │   └── <expression>
│   │       │       └── <simple-expression>
│   │       │           └── <term>
│   │       │               └── <factor>
│   │       │                   └── IDENTIFIER(hasil)
│   │       └── RPARENTHESIS())
│   └── KEYWORD(selesai)
└── DOT(.)

=== BUILDING AST ===
=== AST BUILT SUCCESSFULLY ===
```



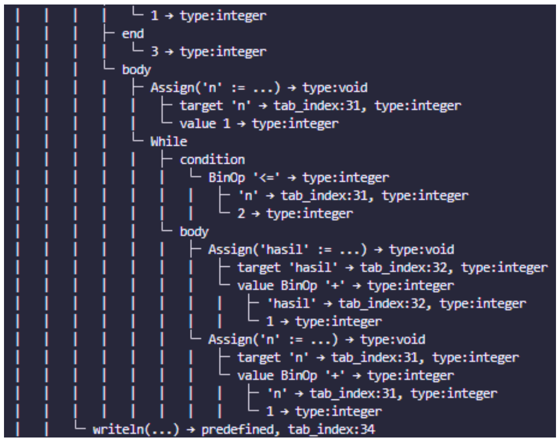
```
=== BUILDING SYMBOL TABLE ===
[Semantic] Visiting Program: TestControlStructures
[Semantic] Program 'TestControlStructures' registered
[Semantic] Visiting Declaration Part
[Semantic] Declaring variables: i,  (idx:30)n,  (idx:31)hasil (idx:32) : integer
[Semantic] Declaring variables: kondisi (idx:33) : boolean
[Semantic] Program 'TestControlStructures' checked successfully
=== SYMBOL TABLE BUILT ===


=== TAB (Identifier Table) ===
 idx         name  link         obj     typ   ref   nrm   lev   adr
---------------------------------------------------------------------
   0      program     0    constant       0     0     1     0     0
   1     variabel     0    constant       0     0     1     0     0
   2        mulai     0    constant       0     0     1     0     0
   3      selesai     0    constant       0     0     1     0     0
   4    konstanta     0    constant       0     0     1     0     0
   5         tipe     0    constant       0     0     1     0     0
   6     prosedur     0    constant       0     0     1     0     0
   7       fungsi     0    constant       0     0     1     0     0
   8         jika     0    constant       0     0     1     0     0
   9         maka     0    constant       0     0     1     0     0
  10   selain-itu     0    constant       0     0     1     0     0
  11        untuk     0    constant       0     0     1     0     0
  12           ke     0    constant       0     0     1     0     0
  13     turun-ke     0    constant       0     0     1     0     0
  14      lakukan     0    constant       0     0     1     0     0
  15       selama     0    constant       0     0     1     0     0
  16       ulangi     0    constant       0     0     1     0     0
  17       sampai     0    constant       0     0     1     0     0
  18        larik     0    constant       0     0     1     0     0
  19         dari     0    constant       0     0     1     0     0
  20      integer     0    constant       0     0     1     0     0
  21         real     0    constant       0     0     1     0     0
  22      boolean     0    constant       0     0     1     0     0
  23         char     0    constant       0     0     1     0     0
  24          dan     0    constant       0     0     1     0     0
  25         atau     0    constant       0     0     1     0     0
  26        tidak     0    constant       0     0     1     0     0
  27         bagi     0    constant       0     0     1     0     0
  28          mod     0    constant       0     0     1     0     0
  29TestControlStructures     0   procedure       0     0     1     0     0
  30            i    29    variable       1     0     1     0     0
  31            n    30    variable       1     0     1     0     1
  32        hasil    31    variable       1     0     1     0     2
  33      kondisi    32    variable       3     0     1     0     3

=== BTAB (Block Table) ===
 idx    last    lpar   psize   vsize
------------------------------------
   0      33       0       0       4

=== DECORATED AST ===
```



```
=== DECORATED AST ===
ProgramNode(name: 'TestControlStructures')
│   ├─ Declarations
│   │   ├─ VarDecl('i') → tab_index:30, type:integer, lev:0
│   │   ├─ VarDecl('n') → tab_index:31, type:integer, lev:0
│   │   ├─ VarDecl('hasil') → tab_index:32, type:integer, lev:0
│   │   └─ VarDecl('kondisi') → tab_index:33, type:boolean, lev:0
│   └─ Block → block_index:0, lev:0
│   │   ├─ Assign('n' := ...) → type:void
│   │   │   ├─ target 'n' → tab_index:31, type:integer
│   │   │   └─ value 10 → type:integer
│   │   ├─ Assign('hasil' := ...) → type:void
│   │   │   ├─ target 'hasil' → tab_index:32, type:integer
│   │   │   └─ value 0 → type:integer
│   │   ├─ If
│   │   │   ├─ condition
│   │   │   │   └─ BinOp '>' → type:integer
│   │   │   │   │   ├─ 'n' → tab_index:31, type:integer
│   │   │   │   │   └─ 5 → type:integer
│   │   │   └─ then
Assign('hasil' := ...) → type:void
│   │   │   │   │   ├─ target 'hasil' → tab_index:32, type:integer
│   │   │   │   │   └─ value BinOp '*' → type:integer
│   │   │   │   │   │   ├─ 'n' → tab_index:31, type:integer
│   │   │   │   │   │   └─ 2 → type:integer
│   │   ├─ If
│   │   │   ├─ condition
│   │   │   │   └─ BinOp '<' → type:integer
│   │   │   │   │   ├─ 'n' → tab_index:31, type:integer
│   │   │   │   │   └─ 5 → type:integer
│   │   │   ├─ then
Assign('hasil' := ...) → type:void
│   │   │   │   │   ├─ target 'hasil' → tab_index:32, type:integer
│   │   │   │   │   └─ value BinOp '+' → type:integer
│   │   │   │   │   │   ├─ 'n' → tab_index:31, type:integer
│   │   │   │   │   │   └─ 10 → type:integer
│   │   │   └─ else
Assign('hasil' := ...) → type:void
│   │   │   │   │   ├─ target 'hasil' → tab_index:32, type:integer
│   │   │   │   │   └─ value BinOp '-' → type:integer
│   │   │   │   │   │   ├─ 'n' → tab_index:31, type:integer
│   │   │   │   │   │   └─ 5 → type:integer
│   │   ├─ If
│   │   │   ├─ condition
│   │   │   │   └─ BinOp '>' → type:integer
│   │   │   │   │   ├─ 'n' → tab_index:31, type:integer
│   │   │   │   │   └─ 0 → type:integer
│   │   │   ├─ then
If
│   │   │   │   │   ├─ condition
│   │   │   │   │   │   └─ BinOp '<' → type:integer
│   │   │   │   │   │   │   ├─ 'n' → tab_index:31, type:integer
│   │   │   │   │   │   │   └─ 100 → type:integer
│   │   │   │   │   ├─ then
Assign('hasil' := ...) → type:void
```



```
Assign('hasil' := ...) → type:void
│   │   │   │   │   │   │   ├─ target 'hasil' → tab_index:32, type:integer
│   │   │   │   │   │   │   └─ value 'n' → tab_index:31, type:integer
│   │   │   │   │   └─ else
Assign('hasil' := ...) → type:void
│   │   │   │   │   │   │   ├─ target 'hasil' → tab_index:32, type:integer
│   │   │   │   │   │   │   └─ value 100 → type:integer
│   │   │   └─ else
Assign('hasil' := ...) → type:void
│   │   │   │   │   ├─ target 'hasil' → tab_index:32, type:integer
│   │   │   │   │   └─ value 0 → type:integer
│   │   ├─ Assign('i' := ...) → type:void
│   │   │   ├─ target 'i' → tab_index:30, type:integer
│   │   │   └─ value 1 → type:integer
│   │   ├─ While
│   │   │   ├─ condition
│   │   │   │   └─ BinOp '<=' → type:integer
│   │   │   │   │   ├─ 'i' → tab_index:30, type:integer
│   │   │   │   │   └─ 5 → type:integer
│   │   │   └─ body
│   │   │   │   ├─ Assign('hasil' := ...) → type:void
│   │   │   │   │   ├─ target 'hasil' → tab_index:32, type:integer
│   │   │   │   │   └─ value BinOp '+' → type:integer
│   │   │   │   │   │   ├─ 'hasil' → tab_index:32, type:integer
│   │   │   │   │   │   └─ 'i' → tab_index:30, type:integer
│   │   │   │   └─ Assign('i' := ...) → type:void
│   │   │   │   │   ├─ target 'i' → tab_index:30, type:integer
│   │   │   │   │   └─ value BinOp '+' → type:integer
│   │   │   │   │   │   ├─ 'i' → tab_index:30, type:integer
│   │   │   │   │   │   └─ 1 → type:integer
│   │   ├─ For('i')
│   │   │   ├─ start
│   │   │   │   └─ 1 → type:integer
│   │   │   ├─ end
│   │   │   │   └─ 10 → type:integer
│   │   │   └─ body
Assign('hasil' := ...) → type:void
│   │   │   │   │   ├─ target 'hasil' → tab_index:32, type:integer
│   │   │   │   │   └─ value BinOp '+' → type:integer
│   │   │   │   │   │   ├─ 'hasil' → tab_index:32, type:integer
│   │   │   │   │   │   └─ 'i' → tab_index:30, type:integer
│   │   ├─ For('i')
│   │   │   ├─ start
│   │   │   │   └─ 10 → type:integer
│   │   │   ├─ end
│   │   │   │   └─ 1 → type:integer
│   │   │   └─ body
Assign('hasil' := ...) → type:void
│   │   │   │   │   ├─ target 'hasil' → tab_index:32, type:integer
│   │   │   │   │   └─ value BinOp '-' → type:integer
│   │   │   │   │   │   ├─ 'hasil' → tab_index:32, type:integer
│   │   │   │   │   │   └─ 1 → type:integer
│   │   ├─ For('i')
│   │   │   ├─ start
│   │   │   │   └─ 1 → type:integer
```



```
│   │   │   │   └─ 1 → type:integer
│   │   │   ├─ end
│   │   │   │   └─ 3 → type:integer
│   │   │   └─ body
│   │   │   │   ├─ Assign('n' := ...) → type:void
│   │   │   │   │   ├─ target 'n' → tab_index:31, type:integer
│   │   │   │   │   └─ value 1 → type:integer
│   │   │   │   └─ While
│   │   │   │   │   ├─ condition
│   │   │   │   │   │   └─ BinOp '<=' → type:integer
│   │   │   │   │   │   │   ├─ 'n' → tab_index:31, type:integer
│   │   │   │   │   │   │   └─ 2 → type:integer
│   │   │   │   │   └─ body
│   │   │   │   │   │   ├─ Assign('hasil' := ...) → type:void
│   │   │   │   │   │   │   ├─ target 'hasil' → tab_index:32, type:integer
│   │   │   │   │   │   │   └─ value BinOp '+' → type:integer
│   │   │   │   │   │   │   │   ├─ 'hasil' → tab_index:32, type:integer
│   │   │   │   │   │   │   │   └─ 1 → type:integer
│   │   │   │   │   │   └─ Assign('n' := ...) → type:void
│   │   │   │   │   │   │   ├─ target 'n' → tab_index:31, type:integer
│   │   │   │   │   │   │   └─ value BinOp '+' → type:integer
│   │   │   │   │   │   │   │   ├─ 'n' → tab_index:31, type:integer
│   │   │   │   │   │   │   │   └─ 1 → type:integer
│   │   └─ writeln(...) → predefined, tab_index:34
```

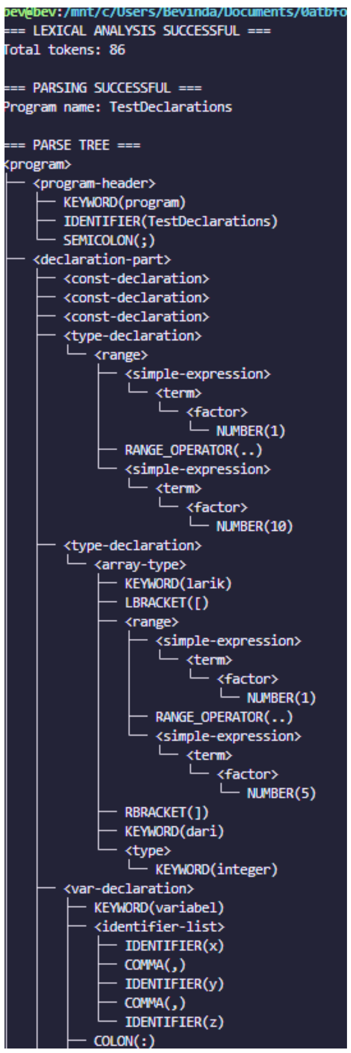
3.7. Test Case 7 - TestDeclarations

Test case ini bertujuan untuk memvalidasi kemampuan parser menangani const declaration, type declaration (subrange dan array), dan variable declaration dengan semua tipe data dasar Pascal-S: integer, real, char, boolean, custom range, dan array.

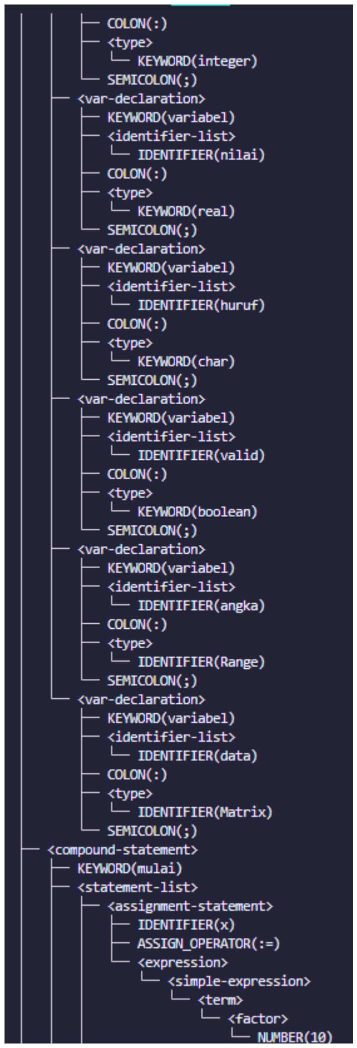
test_declarations.pas

```
program TestDeclarations;

konstanta
  MAX = 100;
  MIN = 0;
  PI = 3.14;

tipe
  Range = 1..10;
  Matrix = larik[1..5] dari integer;

variabel
  x, y, z: integer;
  nilai: real;
  huruf: char;
  valid: boolean;
  angka: Range;
  data: Matrix;

mulai
  x := 10;
  y := 20;
  z := x + y;
  writeln('Hasil = ', z)
selesai.
```

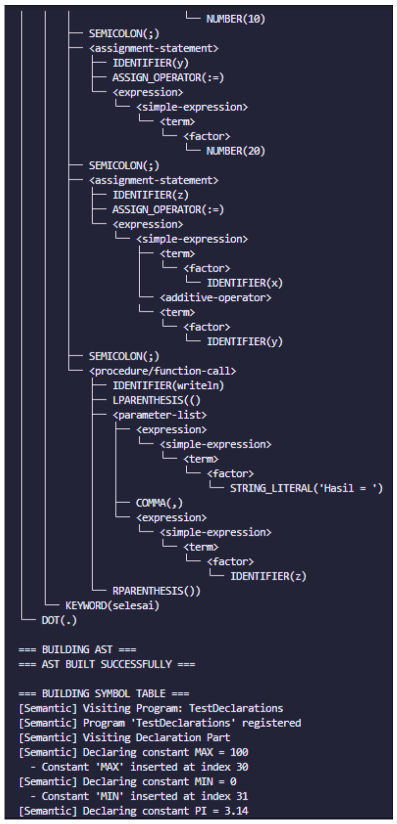
Output:



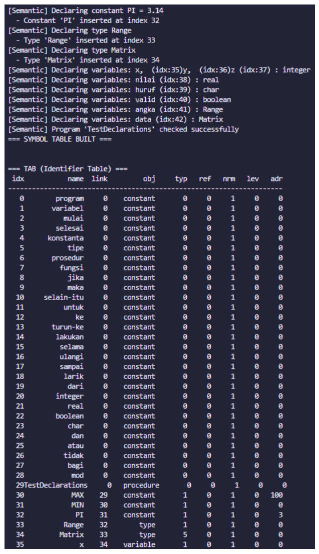
```
bev@bev:/mnt/c/Users/Bevinda/Documents/0atbfo
=== LEXICAL ANALYSIS SUCCESSFUL ===
Total tokens: 86

=== PARSING SUCCESSFUL ===
Program name: TestDeclarations

=== PARSE TREE ===
<program>
├── <program-header>
│   ├── KEYWORD(program)
│   ├── IDENTIFIER(TestDeclarations)
│   └── SEMICOLON(;)
├── <declaration-part>
│   ├── <const-declaration>
│   ├── <const-declaration>
│   ├── <const-declaration>
│   ├── <type-declaration>
│   │   └── <range>
│   │       ├── <simple-expression>
│   │       │   └── <term>
│   │       │       └── <factor>
│   │       │           └── NUMBER(1)
│   │       ├── RANGE_OPERATOR(..)
│   │       └── <simple-expression>
│   │           └── <term>
│   │               └── <factor>
│   │                   └── NUMBER(10)
│   ├── <type-declaration>
│   │   └── <array-type>
│   │       ├── KEYWORD(larik)
│   │       ├── LBRACKET([)
│   │       ├── <range>
│   │       │   ├── <simple-expression>
│   │       │   │   └── <term>
│   │       │   │       └── <factor>
│   │       │   │           └── NUMBER(1)
│   │       │   ├── RANGE_OPERATOR(..)
│   │       │   └── <simple-expression>
│   │       │       └── <term>
│   │       │           └── <factor>
│   │       │               └── NUMBER(5)
│   │       ├── RBRACKET(])
│   │       ├── KEYWORD(dari)
│   │       └── <type>
│   │           └── KEYWORD(integer)
│   ├── <var-declaration>
│   │   ├── KEYWORD(variabel)
│   │   ├── <identifier-list>
│   │   │   ├── IDENTIFIER(x)
│   │   │   ├── COMMA(,)
│   │   │   ├── IDENTIFIER(y)
│   │   │   ├── COMMA(,)
│   │   │   └── IDENTIFIER(z)
│   │   ├── COLON(:)
```



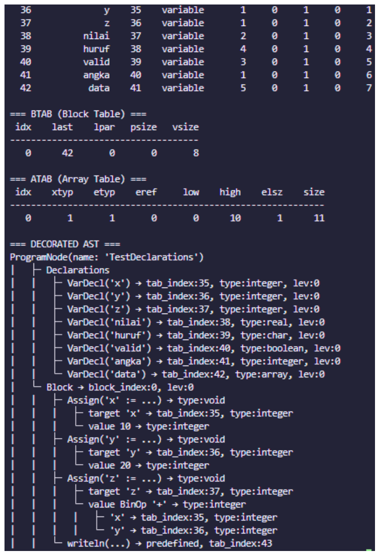
```
│   │   ├── COLON(:)
│   │   ├── <type>
│   │   │   └── KEYWORD(integer)
│   │   └── SEMICOLON(;)
│   ├── <var-declaration>
│   │   ├── KEYWORD(variabel)
│   │   ├── <identifier-list>
│   │   │   └── IDENTIFIER(nilai)
│   │   ├── COLON(:)
│   │   ├── <type>
│   │   │   └── KEYWORD(real)
│   │   └── SEMICOLON(;)
│   ├── <var-declaration>
│   │   ├── KEYWORD(variabel)
│   │   ├── <identifier-list>
│   │   │   └── IDENTIFIER(huruf)
│   │   ├── COLON(:)
│   │   ├── <type>
│   │   │   └── KEYWORD(char)
│   │   └── SEMICOLON(;)
│   ├── <var-declaration>
│   │   ├── KEYWORD(variabel)
│   │   ├── <identifier-list>
│   │   │   └── IDENTIFIER(valid)
│   │   ├── COLON(:)
│   │   ├── <type>
│   │   │   └── KEYWORD(boolean)
│   │   └── SEMICOLON(;)
│   ├── <var-declaration>
│   │   ├── KEYWORD(variabel)
│   │   ├── <identifier-list>
│   │   │   └── IDENTIFIER(angka)
│   │   ├── COLON(:)
│   │   ├── <type>
│   │   │   └── IDENTIFIER(Range)
│   │   └── SEMICOLON(;)
│   └── <var-declaration>
│       ├── KEYWORD(variabel)
│       ├── <identifier-list>
│       │   └── IDENTIFIER(data)
│       ├── COLON(:)
│       ├── <type>
│       │   └── IDENTIFIER(Matrix)
│       └── SEMICOLON(;)
├── <compound-statement>
│   ├── KEYWORD(mulai)
│   ├── <statement-list>
│   │   ├── <assignment-statement>
│   │   │   ├── IDENTIFIER(x)
│   │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │   └── <expression>
│   │   │       └── <simple-expression>
│   │   │           └── <term>
│   │   │               └── <factor>
│   │   │                   └── NUMBER(10)
```



```
│   │   │                   └── NUMBER(10)
│   │   ├── SEMICOLON(;)
│   │   ├── <assignment-statement>
│   │   │   ├── IDENTIFIER(y)
│   │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │   └── <expression>
│   │   │       └── <simple-expression>
│   │   │           └── <term>
│   │   │               └── <factor>
│   │   │                   └── NUMBER(20)
│   │   ├── SEMICOLON(;)
│   │   ├── <assignment-statement>
│   │   │   ├── IDENTIFIER(z)
│   │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │   └── <expression>
│   │   │       └── <simple-expression>
│   │   │           ├── <term>
│   │   │           │   └── <factor>
│   │   │           │       └── IDENTIFIER(x)
│   │   │           └── <additive-operator>
│   │   │           └── <term>
│   │   │               └── <factor>
│   │   │                   └── IDENTIFIER(y)
│   │   ├── SEMICOLON(;)
│   │   └── <procedure/function-call>
│   │       ├── IDENTIFIER(writeln)
│   │       ├── LPARENTHESIS(()
│   │       ├── <parameter-list>
│   │       │   ├── <expression>
│   │       │   │   └── <simple-expression>
│   │       │   │       └── <term>
│   │       │   │           └── <factor>
│   │       │   │               └── STRING_LITERAL('Hasil = ')
│   │       │   ├── COMMA(,)
│   │       │   └── <expression>
│   │       │       └── <simple-expression>
│   │       │           └── <term>
│   │       │               └── <factor>
│   │       │                   └── IDENTIFIER(z)
│   │       └── RPARENTHESIS())
│   └── KEYWORD(selesai)
└── DOT(.)

=== BUILDING AST ===
=== AST BUILT SUCCESSFULLY ===

=== BUILDING SYMBOL TABLE ===
[Semantic] Visiting Program: TestDeclarations
[Semantic] Program 'TestDeclarations' registered
[Semantic] Visiting Declaration Part
[Semantic] Declaring constant MAX = 100
  - Constant 'MAX' inserted at index 30
[Semantic] Declaring constant MIN = 0
  - Constant 'MIN' inserted at index 31
[Semantic] Declaring constant PI = 3.14
```



```
[Semantic] Declaring constant PI = 3.14
  - Constant 'PI' inserted at index 32
[Semantic] Declaring type Range
  - Type 'Range' inserted at index 33
[Semantic] Declaring type Matrix
  - Type 'Matrix' inserted at index 34
[Semantic] Declaring variables: x,  (idx:35)y,  (idx:36)z (idx:37) : integer
[Semantic] Declaring variables: nilai (idx:38) : real
[Semantic] Declaring variables: huruf (idx:39) : char
[Semantic] Declaring variables: valid (idx:40) : boolean
[Semantic] Declaring variables: angka (idx:41) : Range
[Semantic] Declaring variables: data (idx:42) : Matrix
[Semantic] Program 'TestDeclarations' checked successfully
=== SYMBOL TABLE BUILT ===


=== TAB (Identifier Table) ===
 idx          name  link        obj     typ   ref   nrm   lev   adr
-------------------------------------------------------------------
   0       program     0   constant       0     0     1     0     0
   1      variabel     0   constant       0     0     1     0     0
   2         mulai     0   constant       0     0     1     0     0
   3       selesai     0   constant       0     0     1     0     0
   4     konstanta     0   constant       0     0     1     0     0
   5          tipe     0   constant       0     0     1     0     0
   6      prosedur     0   constant       0     0     1     0     0
   7        fungsi     0   constant       0     0     1     0     0
   8          jika     0   constant       0     0     1     0     0
   9          maka     0   constant       0     0     1     0     0
  10    selain-itu     0   constant       0     0     1     0     0
  11         untuk     0   constant       0     0     1     0     0
  12            ke     0   constant       0     0     1     0     0
  13      turun-ke     0   constant       0     0     1     0     0
  14       lakukan     0   constant       0     0     1     0     0
  15        selama     0   constant       0     0     1     0     0
  16        ulangi     0   constant       0     0     1     0     0
  17        sampai     0   constant       0     0     1     0     0
  18         larik     0   constant       0     0     1     0     0
  19          dari     0   constant       0     0     1     0     0
  20       integer     0   constant       0     0     1     0     0
  21          real     0   constant       0     0     1     0     0
  22       boolean     0   constant       0     0     1     0     0
  23          char     0   constant       0     0     1     0     0
  24           dan     0   constant       0     0     1     0     0
  25          atau     0   constant       0     0     1     0     0
  26         tidak     0   constant       0     0     1     0     0
  27          bagi     0   constant       0     0     1     0     0
  28           mod     0   constant       0     0     1     0     0
  29TestDeclarations     0   procedure      0     0     1     0     0
  30           MAX    29   constant       1     0     1     0   100
  31           MIN    30   constant       1     0     1     0     0
  32            PI    31   constant       1     0     1     0     3
  33         Range    32       type       1     0     1     0     0
  34        Matrix    33       type       5     0     1     0     0
  35             x    34   variable       1     0     1     0     0
```



```
  36             y    35   variable       1     0     1     0     1
  37             z    36   variable       1     0     1     0     2
  38         nilai    37   variable       2     0     1     0     3
  39         huruf    38   variable       4     0     1     0     4
  40         valid    39   variable       3     0     1     0     5
  41         angka    40   variable       1     0     1     0     6
  42          data    41   variable       5     0     1     0     7

=== BTAB (Block Table) ===
 idx    last    lpar   psize   vsize
------------------------------------
   0      42       0       0       8

=== ATAB (Array Table) ===
 idx    xtyp    etyp    eref     low    high    elsz    size
------------------------------------------------------------
   0       1       1       0       0      10       1      11

=== DECORATED AST ===
ProgramNode(name: 'TestDeclarations')
|   ├─ Declarations
|   |   ├─ VarDecl('x') → tab_index:35, type:integer, lev:0
|   |   ├─ VarDecl('y') → tab_index:36, type:integer, lev:0
|   |   ├─ VarDecl('z') → tab_index:37, type:integer, lev:0
|   |   ├─ VarDecl('nilai') → tab_index:38, type:real, lev:0
|   |   ├─ VarDecl('huruf') → tab_index:39, type:char, lev:0
|   |   ├─ VarDecl('valid') → tab_index:40, type:boolean, lev:0
|   |   ├─ VarDecl('angka') → tab_index:41, type:integer, lev:0
|   |   └─ VarDecl('data') → tab_index:42, type:array, lev:0
|   └─ Block → block_index:0, lev:0
|   |   ├─ Assign('x' := ...) → type:void
|   |   |   ├─ target 'x' → tab_index:35, type:integer
|   |   |   └─ value 10 → type:integer
|   |   ├─ Assign('y' := ...) → type:void
|   |   |   ├─ target 'y' → tab_index:36, type:integer
|   |   |   └─ value 20 → type:integer
|   |   ├─ Assign('z' := ...) → type:void
|   |   |   ├─ target 'z' → tab_index:37, type:integer
|   |   |   └─ value BinOp '+' → type:integer
|   |   |   |   ├─ 'x' → tab_index:35, type:integer
|   |   |   |   └─ 'y' → tab_index:36, type:integer
|   |   └─ writeln(...) → predefined, tab_index:43
```

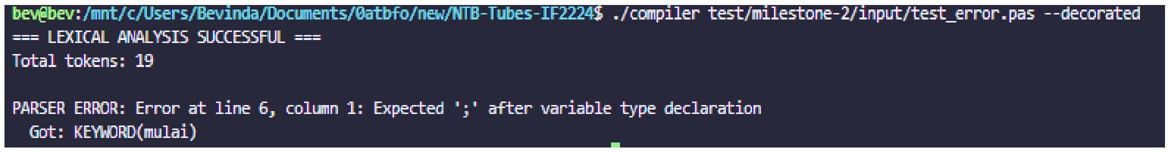
3.8. Test Case 8 - TestError

Test case ini sengaja mengandung error sintaks (kurang semicolons) untuk menguji validasi bahwa parser dapat mendeteksi kesalahan dan memberikan error message.

test_declarations.pas

`program TestError;`
 	
`variabel`
`  x, y: integer`
 	
`mulai`
`  x := 5;`
`  y := 10`
`selesai.`

Output:



```
bev@bev:/mnt/c/Users/Bevinda/Documents/0atbfo/new/NTB-Tubes-IF2224$ ./compiler test/milestone-2/input/test_error.pas --decorated
=== LEXICAL ANALYSIS SUCCESSFUL ===
Total tokens: 19

PARSER ERROR: Error at line 6, column 1: Expected ';' after variable type declaration
  Got: KEYWORD(mulai)
```

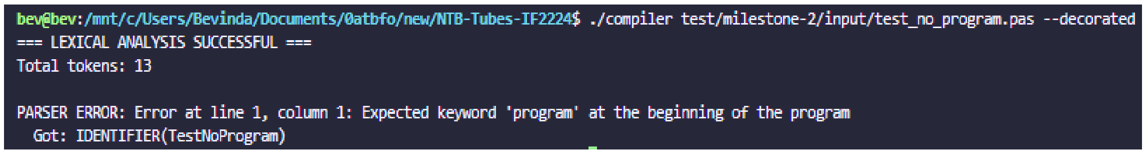
3.9. Test Case 9 - TestNoProgram

Test case ini sengaja tidak menggunakan keyword program di awal untuk memvalidasi bahwa parser dapat mendeteksi error pada program header dan memberikan pesan error yang sesuai.

test_no_program.pas

`TestNoProgram;`
 	
`variabel`
`  x: integer;`
 	
`mulai`
`  x := 5`
`selesai.`

Output:


```
bev@bev:/mnt/c/Users/Bevinda/Documents/0atbfo/new/NTB-Tubes-IF2224$ ./compiler test/milestone-2/input/test_no_program.pas --decorated
=== LEXICAL ANALYSIS SUCCESSFUL ===
Total tokens: 13

PARSER ERROR: Error at line 1, column 1: Expected keyword 'program' at the beginning of the program
  Got: IDENTIFIER(TestNoProgram)
```

3.10. Test Case 10 - TestProceduresAndFunctions

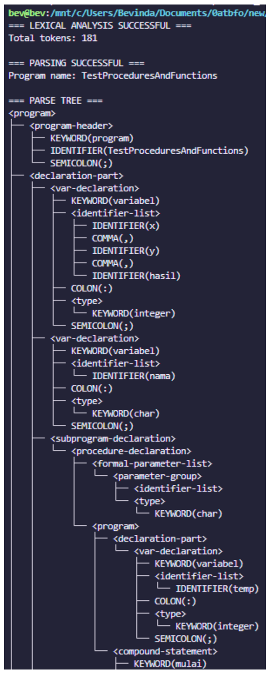
Test case ini fokus pada menguji subprogram declaration yaitu procedure dengan/tanpa parameter, function dengan return type, formal parameter lists, dan pemanggilan subprogram dengan actual parameters. Juga menguji nested function call sebagai parameter dan function assignment.

test_subprograms.pas

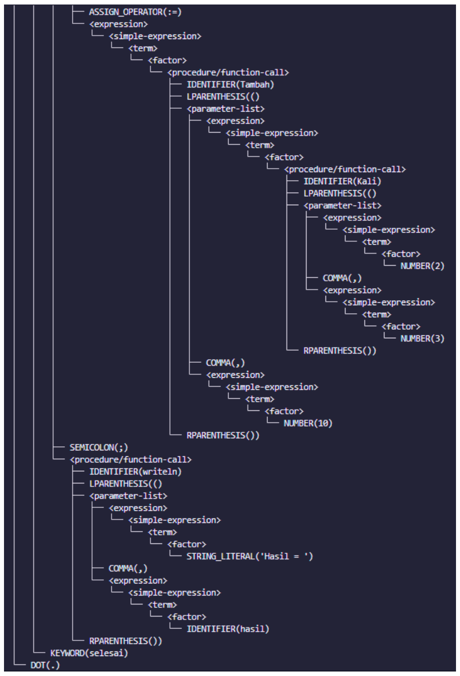
```
program TestProceduresAndFunctions;

variabel
  x, y, hasil: integer;
  nama: char;

prosedur PrintHello(msg: char);
variabel
  temp: integer;
mulai
  temp := 0;
  writeln(msg)
selesai;

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

fungsi Tambah(a, b: integer): integer;
variabel
  result: integer;
mulai
```

```
  result := a + b;
  Tambah := result
selesai;

fungsi Kali(x: integer; y: integer): integer;
mulai
  Kali := x * y
selesai;

mulai
  x := 10;
  y := 20;

  { Call procedures }
  PrintHello('A');
  Swap(x, y);

  { Call functions }
  hasil := Tambah(x, y);
  hasil := Kali(5, 6);
  hasil := Tambah(Kali(2, 3), 10);

  writeln('Hasil = ', hasil)
selesai.
```

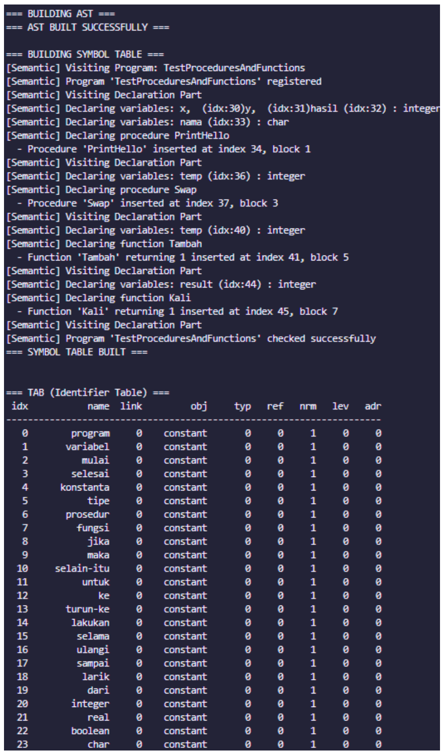
Output:



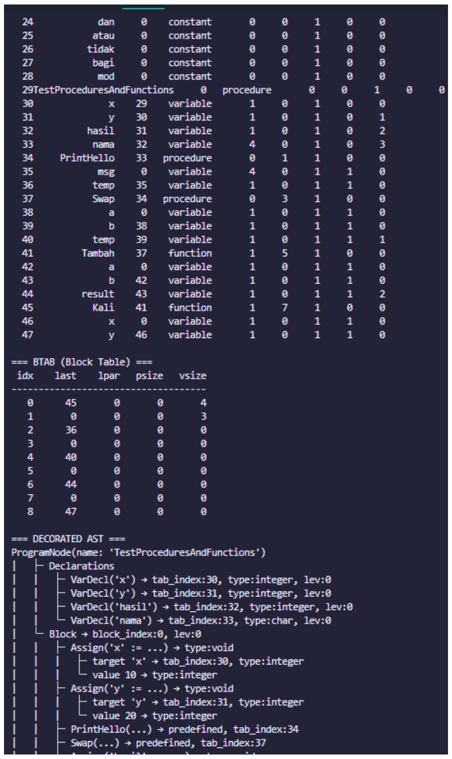
```
bev@bev:/mnt/c/Users/Bevinda/Documents/0atbfo/new
=== LEXICAL ANALYSIS SUCCESSFUL ===
Total tokens: 181

=== PARSING SUCCESSFUL ===
Program name: TestProceduresAndFunctions

=== PARSE TREE ===
<program>
├── <program-header>
│   ├── KEYWORD(program)
│   ├── IDENTIFIER(TestProceduresAndFunctions)
│   └── SEMICOLON(;)
├── <declaration-part>
│   ├── <var-declaration>
│   │   ├── KEYWORD(variabel)
│   │   ├── <identifier-list>
│   │   │   ├── IDENTIFIER(x)
│   │   │   ├── COMMA(,)
│   │   │   ├── IDENTIFIER(y)
│   │   │   ├── COMMA(,)
│   │   │   └── IDENTIFIER(hasil)
│   │   ├── COLON(:)
│   │   ├── <type>
│   │   │   └── KEYWORD(integer)
│   │   └── SEMICOLON(;)
│   ├── <var-declaration>
│   │   ├── KEYWORD(variabel)
│   │   ├── <identifier-list>
│   │   │   └── IDENTIFIER(nama)
│   │   ├── COLON(:)
│   │   ├── <type>
│   │   │   └── KEYWORD(char)
│   │   └── SEMICOLON(;)
│   ├── <subprogram-declaration>
│   │   └── <procedure-declaration>
│   │       ├── <formal-parameter-list>
│   │       │   └── <parameter-group>
│   │       │       ├── <identifier-list>
│   │       │       └── <type>
│   │       │           └── KEYWORD(char)
│   │       └── <program>
│   │           ├── <declaration-part>
│   │           │   └── <var-declaration>
│   │           │       ├── KEYWORD(variabel)
│   │           │       ├── <identifier-list>
│   │           │       │   └── IDENTIFIER(temp)
│   │           │       ├── COLON(:)
│   │           │       ├── <type>
│   │           │       │   └── KEYWORD(integer)
│   │           │       └── SEMICOLON(;)
│   │           └── <compound-statement>
│   │               ├── KEYWORD(mulai)
```



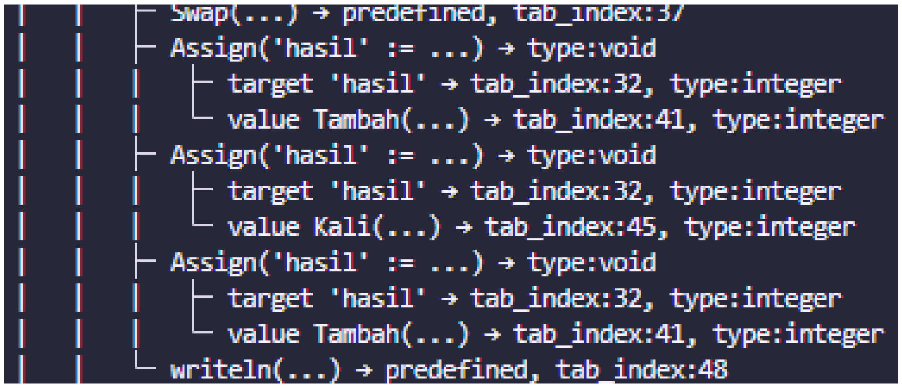
```
│   │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │   └── <expression>
│   │   │       └── <simple-expression>
│   │   │           └── <term>
│   │   │               └── <factor>
│   │   │                   └── <procedure/function-call>
│   │   │                       ├── IDENTIFIER(Tambah)
│   │   │                       ├── LPARENTHESIS(()
│   │   │                       ├── <parameter-list>
│   │   │                       │   ├── <expression>
│   │   │                       │   │   └── <simple-expression>
│   │   │                       │   │       └── <term>
│   │   │                       │   │           └── <factor>
│   │   │                       │   │               └── <procedure/function-call>
│   │   │                       │   │                   ├── IDENTIFIER(Kali)
│   │   │                       │   │                   ├── LPARENTHESIS(()
│   │   │                       │   │                   ├── <parameter-list>
│   │   │                       │   │                   │   ├── <expression>
│   │   │                       │   │                   │   │   └── <simple-expression>
│   │   │                       │   │                   │   │       └── <term>
│   │   │                       │   │                   │   │           └── <factor>
│   │   │                       │   │                   │   │               └── NUMBER(2)
│   │   │                       │   │                   │   ├── COMMA(,)
│   │   │                       │   │                   │   └── <expression>
│   │   │                       │   │                   │       └── <simple-expression>
│   │   │                       │   │                   │           └── <term>
│   │   │                       │   │                   │               └── <factor>
│   │   │                       │   │                   │                   └── NUMBER(3)
│   │   │                       │   │                   └── RPARENTHESIS())
│   │   │                       │   ├── COMMA(,)
│   │   │                       │   └── <expression>
│   │   │                       │       └── <simple-expression>
│   │   │                       │           └── <term>
│   │   │                       │               └── <factor>
│   │   │                       │                   └── NUMBER(10)
│   │   │                       └── RPARENTHESIS())
│   │   ├── SEMICOLON(;)
│   │   └── <procedure/function-call>
│   │       ├── IDENTIFIER(writeln)
│   │       ├── LPARENTHESIS(()
│   │       ├── <parameter-list>
│   │       │   ├── <expression>
│   │       │   │   └── <simple-expression>
│   │       │   │       └── <term>
│   │       │   │           └── <factor>
│   │       │   │               └── STRING_LITERAL('Hasil = ')
│   │       │   ├── COMMA(,)
│   │       │   └── <expression>
│   │       │       └── <simple-expression>
│   │       │           └── <term>
│   │       │               └── <factor>
│   │       │                   └── IDENTIFIER(hasil)
│   │       └── RPARENTHESIS())
│   └── KEYWORD(selesai)
└── DOT(.)
```



```
=== BUILDING AST ===
=== AST BUILT SUCCESSFULLY ===

=== BUILDING SYMBOL TABLE ===
[Semantic] Visiting Program: TestProceduresAndFunctions
[Semantic] Program 'TestProceduresAndFunctions' registered
[Semantic] Visiting Declaration Part
[Semantic] Declaring variables: x,  (idx:30)y,  (idx:31)hasil (idx:32) : integer
[Semantic] Declaring variables: nama (idx:33) : char
[Semantic] Declaring procedure PrintHello
  - Procedure 'PrintHello' inserted at index 34, block 1
[Semantic] Visiting Declaration Part
[Semantic] Declaring variables: temp (idx:36) : integer
[Semantic] Declaring procedure Swap
  - Procedure 'Swap' inserted at index 37, block 3
[Semantic] Visiting Declaration Part
[Semantic] Declaring variables: temp (idx:40) : integer
[Semantic] Declaring function Tambah
  - Function 'Tambah' returning 1 inserted at index 41, block 5
[Semantic] Visiting Declaration Part
[Semantic] Declaring variables: result (idx:44) : integer
[Semantic] Declaring function Kali
  - Function 'Kali' returning 1 inserted at index 45, block 7
[Semantic] Visiting Declaration Part
[Semantic] Program 'TestProceduresAndFunctions' checked successfully
=== SYMBOL TABLE BUILT ===


=== TAB (Identifier Table) ===
 idx          name  link        obj    typ   ref   nrm   lev   adr
--------------------------------------------------------------------
   0       program     0   constant      0     0     1     0     0
   1      variabel     0   constant      0     0     1     0     0
   2         mulai     0   constant      0     0     1     0     0
   3       selesai     0   constant      0     0     1     0     0
   4     konstanta     0   constant      0     0     1     0     0
   5          tipe     0   constant      0     0     1     0     0
   6      prosedur     0   constant      0     0     1     0     0
   7        fungsi     0   constant      0     0     1     0     0
   8          jika     0   constant      0     0     1     0     0
   9          maka     0   constant      0     0     1     0     0
  10    selain-itu     0   constant      0     0     1     0     0
  11         untuk     0   constant      0     0     1     0     0
  12            ke     0   constant      0     0     1     0     0
  13      turun-ke     0   constant      0     0     1     0     0
  14       lakukan     0   constant      0     0     1     0     0
  15        selama     0   constant      0     0     1     0     0
  16        ulangi     0   constant      0     0     1     0     0
  17        sampai     0   constant      0     0     1     0     0
  18         larik     0   constant      0     0     1     0     0
  19          dari     0   constant      0     0     1     0     0
  20       integer     0   constant      0     0     1     0     0
  21          real     0   constant      0     0     1     0     0
  22       boolean     0   constant      0     0     1     0     0
  23          char     0   constant      0     0     1     0     0
```



```
  24           dan     0   constant      0     0     1     0     0
  25          atau     0   constant      0     0     1     0     0
  26         tidak     0   constant      0     0     1     0     0
  27          bagi     0   constant      0     0     1     0     0
  28           mod     0   constant      0     0     1     0     0
  29TestProceduresAndFunctions     0   procedure      0     0     1     0     0
  30             x    29   variable      1     0     1     0     0
  31             y    30   variable      1     0     1     0     1
  32         hasil    31   variable      1     0     1     0     2
  33          nama    32   variable      4     0     1     0     3
  34    PrintHello    33  procedure      0     1     1     0     0
  35           msg     0   variable      4     0     1     1     0
  36          temp    35   variable      1     0     1     1     0
  37          Swap    34  procedure      0     3     1     0     0
  38             a     0   variable      1     0     1     1     0
  39             b    38   variable      1     0     1     1     0
  40          temp    39   variable      1     0     1     1     1
  41        Tambah    37   function      1     5     1     0     0
  42             a     0   variable      1     0     1     1     0
  43             b    42   variable      1     0     1     1     0
  44        result    43   variable      1     0     1     1     2
  45          Kali    41   function      1     7     1     0     0
  46             x     0   variable      1     0     1     1     0
  47             y    46   variable      1     0     1     1     0

=== BTAB (Block Table) ===
 idx   last   lpar  psize  vsize
---------------------------------
   0     45      0      0      4
   1      0      0      0      3
   2     36      0      0      0
   3      0      0      0      0
   4     40      0      0      0
   5      0      0      0      0
   6     44      0      0      0
   7      0      0      0      0
   8     47      0      0      0

=== DECORATED AST ===
ProgramNode(name: 'TestProceduresAndFunctions')
|    ├─ Declarations
|    |    ├─ VarDecl('x') → tab_index:30, type:integer, lev:0
|    |    ├─ VarDecl('y') → tab_index:31, type:integer, lev:0
|    |    ├─ VarDecl('hasil') → tab_index:32, type:integer, lev:0
|    |    └─ VarDecl('nama') → tab_index:33, type:char, lev:0
|    └─ Block → block_index:0, lev:0
|    |    ├─ Assign('x' := ...) → type:void
|    |    |    ├─ target 'x' → tab_index:30, type:integer
|    |    |    └─ value 10 → type:integer
|    |    ├─ Assign('y' := ...) → type:void
|    |    |    ├─ target 'y' → tab_index:31, type:integer
|    |    |    └─ value 20 → type:integer
|    |    ├─ PrintHello(...) → predefined, tab_index:34
|    |    ├─ Swap(...) → predefined, tab_index:37
```



```
|    |    ├─ Swap(...) → predefined, tab_index:37
|    |    ├─ Assign('hasil' := ...) → type:void
|    |    |    ├─ target 'hasil' → tab_index:32, type:integer
|    |    |    └─ value Tambah(...) → tab_index:41, type:integer
|    |    ├─ Assign('hasil' := ...) → type:void
|    |    |    ├─ target 'hasil' → tab_index:32, type:integer
|    |    |    └─ value Kali(...) → tab_index:45, type:integer
|    |    ├─ Assign('hasil' := ...) → type:void
|    |    |    ├─ target 'hasil' → tab_index:32, type:integer
|    |    |    └─ value Tambah(...) → tab_index:41, type:integer
|    |    └─ writeln(...) → predefined, tab_index:48
```

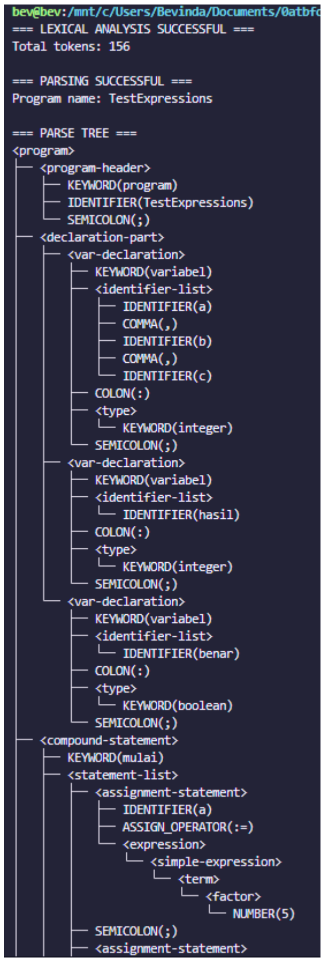
3.11. Test Case 11 - TestExpressions

Test case ini bertujuan untuk menguji validasi expression parsing dengan semua operator: aritmatika (+, -, *, bagi, mod), relasional (=, <>, <, <=, >, >=), dan logika (dan, atau, tidak). Menguji operator precedence dan associativity, serta ekspresi parenthesized.

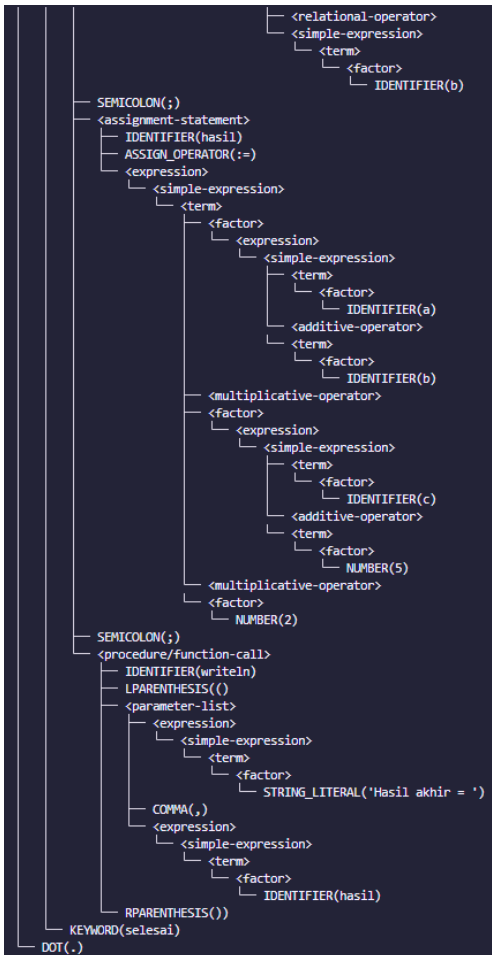
test_expressions.pas

```
program TestExpressions;

variabel
  a, b, c: integer;
  hasil: integer;
  benar: boolean;

mulai
  a := 5;
  b := 10;
  c := a + b;
  c := a - b;
  c := a * b;
  c := a bagi b;
  c := a mod b;
  benar := a = b;
  benar := a <> b;
  benar := a < b;
  benar := a <= b;
  benar := a > b;
  benar := a >= b;
  benar := (a < b) dan (b < 20);
  benar := (a > b) atau (b > 5);
  benar := tidak (a = b);
  hasil := (a + b) * (c - 5) bagi 2;
  writeln('Hasil akhir = ', hasil)
selesai.
```

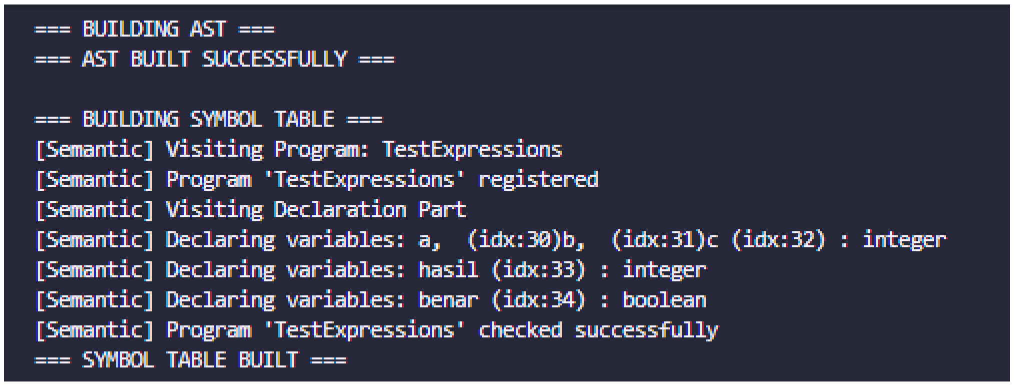
Output:



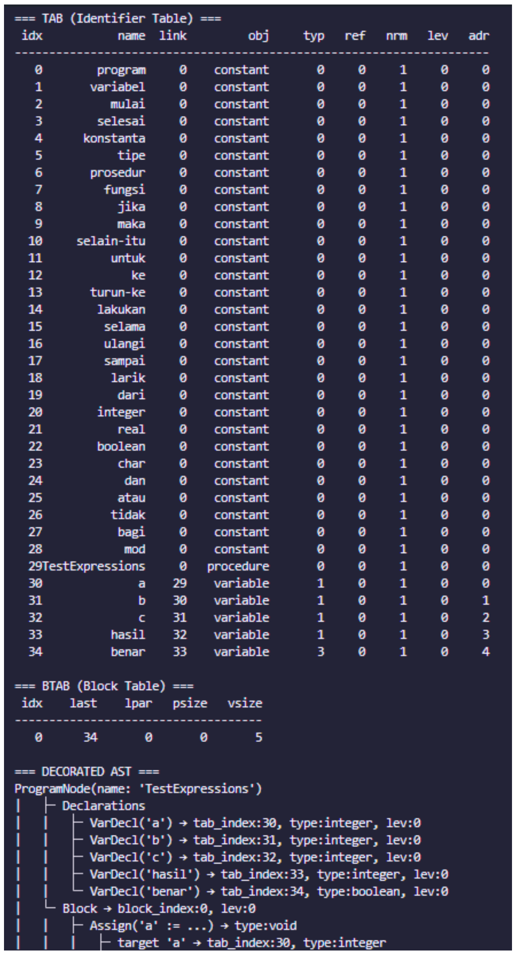
```
bev@bev:/mnt/c/Users/Bevinda/Documents/0atbfc
=== LEXICAL ANALYSIS SUCCESSFUL ===
Total tokens: 156

=== PARSING SUCCESSFUL ===
Program name: TestExpressions

=== PARSE TREE ===
<program>
├── <program-header>
│   ├── KEYWORD(program)
│   ├── IDENTIFIER(TestExpressions)
│   └── SEMICOLON(;)
├── <declaration-part>
│   ├── <var-declaration>
│   │   ├── KEYWORD(variabel)
│   │   ├── <identifier-list>
│   │   │   ├── IDENTIFIER(a)
│   │   │   ├── COMMA(,)
│   │   │   ├── IDENTIFIER(b)
│   │   │   ├── COMMA(,)
│   │   │   └── IDENTIFIER(c)
│   │   ├── COLON(:)
│   │   ├── <type>
│   │   │   └── KEYWORD(integer)
│   │   └── SEMICOLON(;)
│   ├── <var-declaration>
│   │   ├── KEYWORD(variabel)
│   │   ├── <identifier-list>
│   │   │   └── IDENTIFIER(hasil)
│   │   ├── COLON(:)
│   │   ├── <type>
│   │   │   └── KEYWORD(integer)
│   │   └── SEMICOLON(;)
│   └── <var-declaration>
│       ├── KEYWORD(variabel)
│       ├── <identifier-list>
│       │   └── IDENTIFIER(benar)
│       ├── COLON(:)
│       ├── <type>
│       │   └── KEYWORD(boolean)
│       └── SEMICOLON(;)
├── <compound-statement>
│   ├── KEYWORD(mulai)
│   ├── <statement-list>
│   │   ├── <assignment-statement>
│   │   │   ├── IDENTIFIER(a)
│   │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │   └── <expression>
│   │   │       └── <simple-expression>
│   │   │           └── <term>
│   │   │               └── <factor>
│   │   │                   └── NUMBER(5)
│   │   ├── SEMICOLON(;)
│   │   ├── <assignment-statement>
```



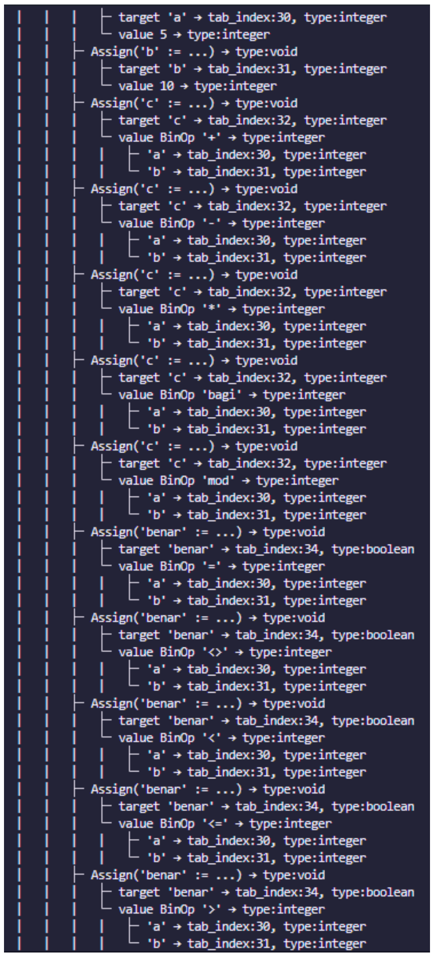
```
│   │   │                           ├── <relational-operator>
│   │   │                           └── <simple-expression>
│   │   │                               └── <term>
│   │   │                                   └── <factor>
│   │   │                                       └── IDENTIFIER(b)
│   │   ├── SEMICOLON(;)
│   │   ├── <assignment-statement>
│   │   │   ├── IDENTIFIER(hasil)
│   │   │   ├── ASSIGN_OPERATOR(:=)
│   │   │   └── <expression>
│   │   │       └── <simple-expression>
│   │   │           └── <term>
│   │   │               ├── <factor>
│   │   │               │   └── <expression>
│   │   │               │       └── <simple-expression>
│   │   │               │           ├── <term>
│   │   │               │           │   └── <factor>
│   │   │               │           │       └── IDENTIFIER(a)
│   │   │               │           ├── <additive-operator>
│   │   │               │           └── <term>
│   │   │               │               └── <factor>
│   │   │               │                   └── IDENTIFIER(b)
│   │   │               ├── <multiplicative-operator>
│   │   │               ├── <factor>
│   │   │               │   └── <expression>
│   │   │               │       └── <simple-expression>
│   │   │               │           ├── <term>
│   │   │               │           │   └── <factor>
│   │   │               │           │       └── IDENTIFIER(c)
│   │   │               │           ├── <additive-operator>
│   │   │               │           └── <term>
│   │   │               │               └── <factor>
│   │   │               │                   └── NUMBER(5)
│   │   │               ├── <multiplicative-operator>
│   │   │               └── <factor>
│   │   │                   └── NUMBER(2)
│   │   ├── SEMICOLON(;)
│   │   └── <procedure/function-call>
│   │       ├── IDENTIFIER(writeln)
│   │       ├── LPARENTHESIS(()
│   │       ├── <parameter-list>
│   │       │   ├── <expression>
│   │       │   │   └── <simple-expression>
│   │       │   │       └── <term>
│   │       │   │           └── <factor>
│   │       │   │               └── STRING_LITERAL('Hasil akhir = ')
│   │       │   ├── COMMA(,)
│   │       │   └── <expression>
│   │       │       └── <simple-expression>
│   │       │           └── <term>
│   │       │               └── <factor>
│   │       │                   └── IDENTIFIER(hasil)
│   │       └── RPARENTHESIS())
│   └── KEYWORD(selesai)
└── DOT(.)
```



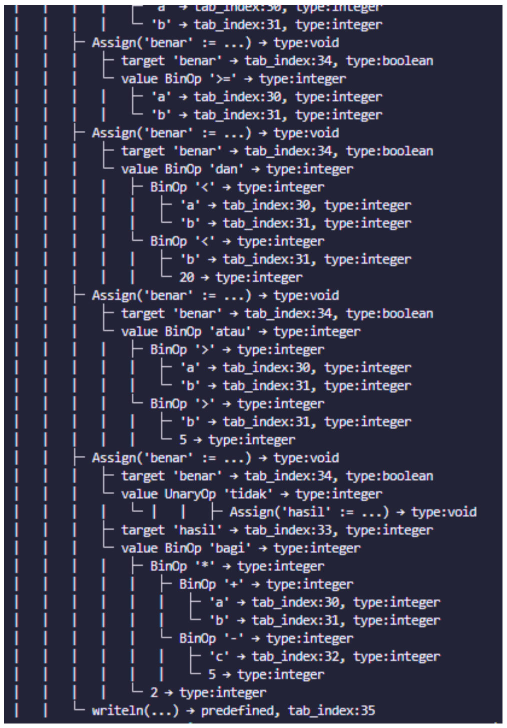
```
=== BUILDING AST ===
=== AST BUILT SUCCESSFULLY ===

=== BUILDING SYMBOL TABLE ===
[Semantic] Visiting Program: TestExpressions
[Semantic] Program 'TestExpressions' registered
[Semantic] Visiting Declaration Part
[Semantic] Declaring variables: a,  (idx:30)b,  (idx:31)c (idx:32) : integer
[Semantic] Declaring variables: hasil (idx:33) : integer
[Semantic] Declaring variables: benar (idx:34) : boolean
[Semantic] Program 'TestExpressions' checked successfully
=== SYMBOL TABLE BUILT ===
```



```
=== TAB (Identifier Table) ===
 idx          name  link         obj     typ   ref   nrm   lev   adr
------------------------------------------------------------------------
   0       program     0    constant       0     0     1     0     0
   1      variabel     0    constant       0     0     1     0     0
   2         mulai     0    constant       0     0     1     0     0
   3       selesai     0    constant       0     0     1     0     0
   4     konstanta     0    constant       0     0     1     0     0
   5          tipe     0    constant       0     0     1     0     0
   6      prosedur     0    constant       0     0     1     0     0
   7        fungsi     0    constant       0     0     1     0     0
   8          jika     0    constant       0     0     1     0     0
   9          maka     0    constant       0     0     1     0     0
  10    selain-itu     0    constant       0     0     1     0     0
  11         untuk     0    constant       0     0     1     0     0
  12            ke     0    constant       0     0     1     0     0
  13      turun-ke     0    constant       0     0     1     0     0
  14       lakukan     0    constant       0     0     1     0     0
  15        selama     0    constant       0     0     1     0     0
  16        ulangi     0    constant       0     0     1     0     0
  17        sampai     0    constant       0     0     1     0     0
  18         larik     0    constant       0     0     1     0     0
  19          dari     0    constant       0     0     1     0     0
  20       integer     0    constant       0     0     1     0     0
  21          real     0    constant       0     0     1     0     0
  22       boolean     0    constant       0     0     1     0     0
  23          char     0    constant       0     0     1     0     0
  24           dan     0    constant       0     0     1     0     0
  25          atau     0    constant       0     0     1     0     0
  26         tidak     0    constant       0     0     1     0     0
  27          bagi     0    constant       0     0     1     0     0
  28           mod     0    constant       0     0     1     0     0
  29TestExpressions    0   procedure       0     0     1     0     0
  30             a    29    variable       1     0     1     0     0
  31             b    30    variable       1     0     1     0     1
  32             c    31    variable       1     0     1     0     2
  33         hasil    32    variable       1     0     1     0     3
  34         benar    33    variable       3     0     1     0     4

=== BTAB (Block Table) ===
 idx    last    lpar   psize   vsize
------------------------------------
   0      34       0       0       5

=== DECORATED AST ===
ProgramNode(name: 'TestExpressions')
|   ├─ Declarations
|   |   ├─ VarDecl('a') → tab_index:30, type:integer, lev:0
|   |   ├─ VarDecl('b') → tab_index:31, type:integer, lev:0
|   |   ├─ VarDecl('c') → tab_index:32, type:integer, lev:0
|   |   ├─ VarDecl('hasil') → tab_index:33, type:integer, lev:0
|   |   └─ VarDecl('benar') → tab_index:34, type:boolean, lev:0
|   └─ Block → block_index:0, lev:0
|   |   ├─ Assign('a' := ...) → type:void
|   |   |   ├─ target 'a' → tab_index:30, type:integer
```



```
|   |   |   ├─ target 'a' → tab_index:30, type:integer
|   |   |   └─ value 5 → type:integer
|   |   ├─ Assign('b' := ...) → type:void
|   |   |   ├─ target 'b' → tab_index:31, type:integer
|   |   |   └─ value 10 → type:integer
|   |   ├─ Assign('c' := ...) → type:void
|   |   |   ├─ target 'c' → tab_index:32, type:integer
|   |   |   └─ value BinOp '+' → type:integer
|   |   |   |   ├─ 'a' → tab_index:30, type:integer
|   |   |   |   └─ 'b' → tab_index:31, type:integer
|   |   ├─ Assign('c' := ...) → type:void
|   |   |   ├─ target 'c' → tab_index:32, type:integer
|   |   |   └─ value BinOp '-' → type:integer
|   |   |   |   ├─ 'a' → tab_index:30, type:integer
|   |   |   |   └─ 'b' → tab_index:31, type:integer
|   |   ├─ Assign('c' := ...) → type:void
|   |   |   ├─ target 'c' → tab_index:32, type:integer
|   |   |   └─ value BinOp '*' → type:integer
|   |   |   |   ├─ 'a' → tab_index:30, type:integer
|   |   |   |   └─ 'b' → tab_index:31, type:integer
|   |   ├─ Assign('c' := ...) → type:void
|   |   |   ├─ target 'c' → tab_index:32, type:integer
|   |   |   └─ value BinOp 'bagi' → type:integer
|   |   |   |   ├─ 'a' → tab_index:30, type:integer
|   |   |   |   └─ 'b' → tab_index:31, type:integer
|   |   ├─ Assign('c' := ...) → type:void
|   |   |   ├─ target 'c' → tab_index:32, type:integer
|   |   |   └─ value BinOp 'mod' → type:integer
|   |   |   |   ├─ 'a' → tab_index:30, type:integer
|   |   |   |   └─ 'b' → tab_index:31, type:integer
|   |   ├─ Assign('benar' := ...) → type:void
|   |   |   ├─ target 'benar' → tab_index:34, type:boolean
|   |   |   └─ value BinOp '=' → type:integer
|   |   |   |   ├─ 'a' → tab_index:30, type:integer
|   |   |   |   └─ 'b' → tab_index:31, type:integer
|   |   ├─ Assign('benar' := ...) → type:void
|   |   |   ├─ target 'benar' → tab_index:34, type:boolean
|   |   |   └─ value BinOp '<>' → type:integer
|   |   |   |   ├─ 'a' → tab_index:30, type:integer
|   |   |   |   └─ 'b' → tab_index:31, type:integer
|   |   ├─ Assign('benar' := ...) → type:void
|   |   |   ├─ target 'benar' → tab_index:34, type:boolean
|   |   |   └─ value BinOp '<' → type:integer
|   |   |   |   ├─ 'a' → tab_index:30, type:integer
|   |   |   |   └─ 'b' → tab_index:31, type:integer
|   |   ├─ Assign('benar' := ...) → type:void
|   |   |   ├─ target 'benar' → tab_index:34, type:boolean
|   |   |   └─ value BinOp '<=' → type:integer
|   |   |   |   ├─ 'a' → tab_index:30, type:integer
|   |   |   |   └─ 'b' → tab_index:31, type:integer
|   |   ├─ Assign('benar' := ...) → type:void
|   |   |   ├─ target 'benar' → tab_index:34, type:boolean
|   |   |   └─ value BinOp '>' → type:integer
|   |   |   |   ├─ 'a' → tab_index:30, type:integer
|   |   |   |   └─ 'b' → tab_index:31, type:integer
```



```
|   |   |   |   ├─ 'a' → tab_index:30, type:integer
|   |   |   |   └─ 'b' → tab_index:31, type:integer
|   |   ├─ Assign('benar' := ...) → type:void
|   |   |   ├─ target 'benar' → tab_index:34, type:boolean
|   |   |   └─ value BinOp '>=' → type:integer
|   |   |   |   ├─ 'a' → tab_index:30, type:integer
|   |   |   |   └─ 'b' → tab_index:31, type:integer
|   |   ├─ Assign('benar' := ...) → type:void
|   |   |   ├─ target 'benar' → tab_index:34, type:boolean
|   |   |   └─ value BinOp 'dan' → type:integer
|   |   |   |   ├─ BinOp '<' → type:integer
|   |   |   |   |   ├─ 'a' → tab_index:30, type:integer
|   |   |   |   |   └─ 'b' → tab_index:31, type:integer
|   |   |   |   └─ BinOp '<' → type:integer
|   |   |   |   |   ├─ 'b' → tab_index:31, type:integer
|   |   |   |   |   └─ 20 → type:integer
|   |   ├─ Assign('benar' := ...) → type:void
|   |   |   ├─ target 'benar' → tab_index:34, type:boolean
|   |   |   └─ value BinOp 'atau' → type:integer
|   |   |   |   ├─ BinOp '>' → type:integer
|   |   |   |   |   ├─ 'a' → tab_index:30, type:integer
|   |   |   |   |   └─ 'b' → tab_index:31, type:integer
|   |   |   |   └─ BinOp '>' → type:integer
|   |   |   |   |   ├─ 'b' → tab_index:31, type:integer
|   |   |   |   |   └─ 5 → type:integer
|   |   ├─ Assign('benar' := ...) → type:void
|   |   |   ├─ target 'benar' → tab_index:34, type:boolean
|   |   |   └─ value UnaryOp 'tidak' → type:integer
|   |   |   |   └─ |   |   ├─ Assign('hasil' := ...) → type:void
|   |   |   ├─ target 'hasil' → tab_index:33, type:integer
|   |   |   └─ value BinOp 'bagi' → type:integer
|   |   |   |   ├─ BinOp '*' → type:integer
|   |   |   |   |   ├─ BinOp '+' → type:integer
|   |   |   |   |   |   ├─ 'a' → tab_index:30, type:integer
|   |   |   |   |   |   └─ 'b' → tab_index:31, type:integer
|   |   |   |   |   └─ BinOp '-' → type:integer
|   |   |   |   |   |   ├─ 'c' → tab_index:32, type:integer
|   |   |   |   |   |   └─ 5 → type:integer
|   |   |   |   └─ 2 → type:integer
|   |   └─ writeln(...) → predefined, tab_index:35
```

3.12. Test Case 12 - TestComprehensive

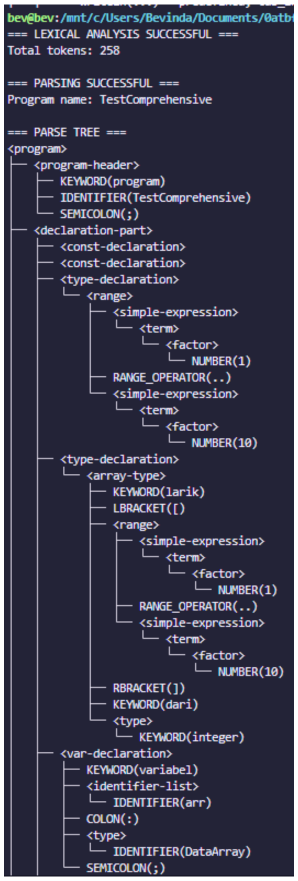
Test case bertujuan untuk mengintegrasikan hampir semua fitur Pascal-S seperti konstanta, tipe custom, variabel, fungsi, prosedur, semua jenis control flow, array indexing, nested loops, operator logika (dan, atau, tidak), dan boolean literals (benar, salah).

test_comprehensive.pas

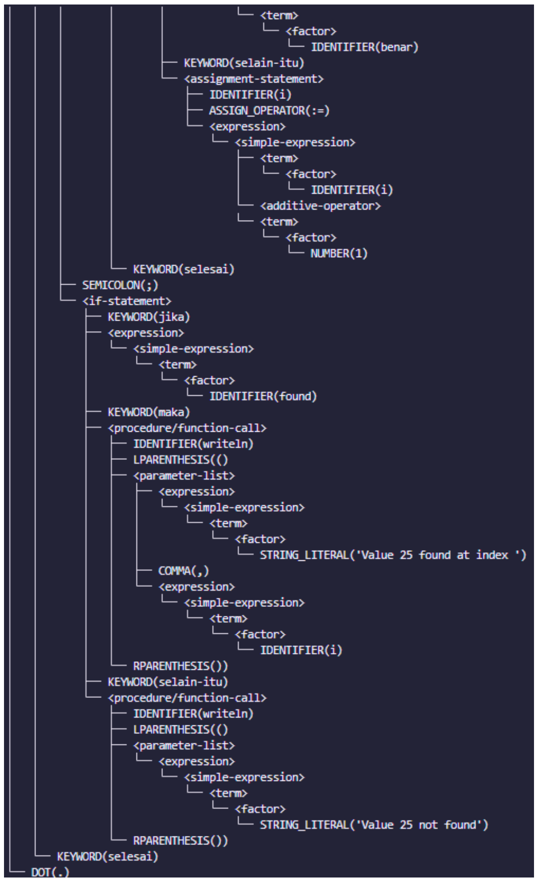
```
program TestComprehensive;

konstanta
  MAX_SIZE = 100;
  VERSION = 2;

tipe
  IndexRange = 1..10;
  DataArray = larik[1..10] dari integer;

variabel
  arr: DataArray;
  i, sum, avg: integer;
  found: boolean;

fungsi FindMax(data: DataArray; size: integer): integer;
variabel
  max, idx: integer;
mulai
  max := data;
  untuk idx := 2 ke size lakukan
    jika data > max maka
      max := data;
  FindMax := max
selesai;

prosedur PrintArray(data: DataArray; count: integer);
variabel
  j: integer;
mulai
  untuk j := 1 ke count lakukan
    write(data, ' ')
selesai;

mulai
  { Initialize array }
  sum := 0;
```

```
    untuk i := 1 ke 10 lakukan
      mulai
        arr := i * 5;
        sum := sum + arr
      selesai;

    { Calculate average }
    avg := sum bagi 10;

    { Find maximum }
    i := FindMax(arr, 10);

    { Print results }
    writeln('Array values:');
    PrintArray(arr, 10);
    writeln();
    writeln('Sum = ', sum);
    writeln('Average = ', avg);
    writeln('Maximum = ', i);

    { Search for value }
    found := salah;
    i := 1;
    selama (i <= 10) dan (tidak found) lakukan
      mulai
        jika arr = 25 maka
          found := benar
        selain-itu
          i := i + 1
      selesai;

    jika found maka
      writeln('Value 25 found at index ', i)
    selain-itu
      writeln('Value 25 not found')
  selesai.
```

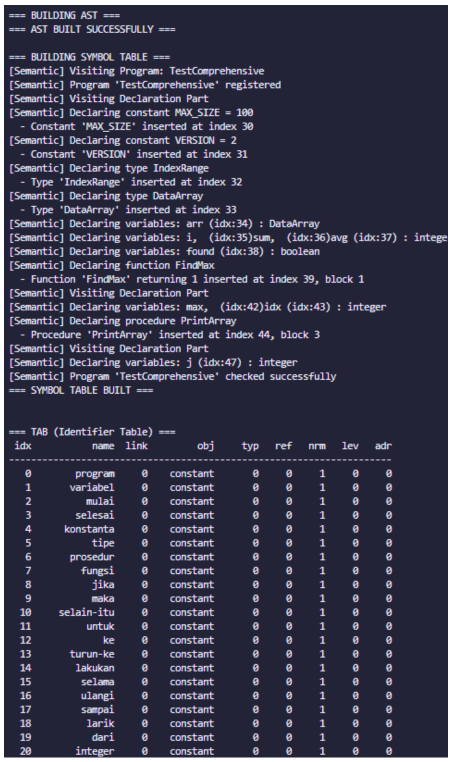
Output:



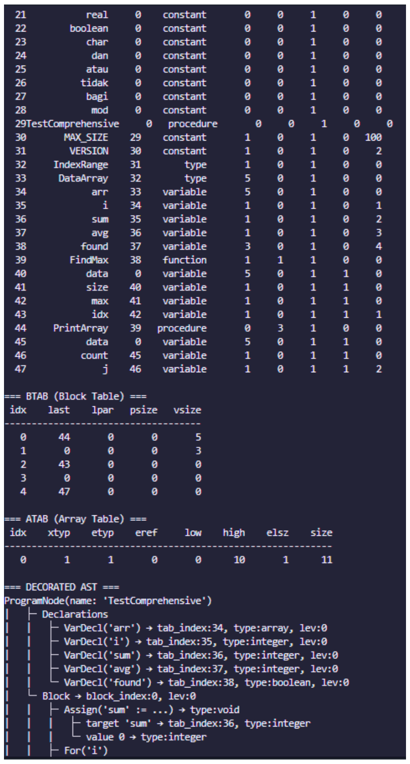
```
bev@bev:/mnt/c/Users/Bevinda/Documents/0atb
=== LEXICAL ANALYSIS SUCCESSFUL ===
Total tokens: 258

=== PARSING SUCCESSFUL ===
Program name: TestComprehensive

=== PARSE TREE ===
<program>
├── <program-header>
│   ├── KEYWORD(program)
│   ├── IDENTIFIER(TestComprehensive)
│   └── SEMICOLON(;)
├── <declaration-part>
│   ├── <const-declaration>
│   ├── <const-declaration>
│   ├── <type-declaration>
│   │   └── <range>
│   │       ├── <simple-expression>
│   │       │   └── <term>
│   │       │       └── <factor>
│   │       │           └── NUMBER(1)
│   │       ├── RANGE_OPERATOR(..)
│   │       └── <simple-expression>
│   │           └── <term>
│   │               └── <factor>
│   │                   └── NUMBER(10)
│   ├── <type-declaration>
│   │   └── <array-type>
│   │       ├── KEYWORD(larik)
│   │       ├── LBRACKET([)
│   │       ├── <range>
│   │       │   ├── <simple-expression>
│   │       │   │   └── <term>
│   │       │   │       └── <factor>
│   │       │   │           └── NUMBER(1)
│   │       │   ├── RANGE_OPERATOR(..)
│   │       │   └── <simple-expression>
│   │       │       └── <term>
│   │       │           └── <factor>
│   │       │               └── NUMBER(10)
│   │       ├── RBRACKET(])
│   │       ├── KEYWORD(dari)
│   │       └── <type>
│   │           └── KEYWORD(integer)
│   ├── <var-declaration>
│   │   ├── KEYWORD(variabel)
│   │   ├── <identifier-list>
│   │   │   └── IDENTIFIER(arr)
│   │   ├── COLON(:)
│   │   ├── <type>
│   │   │   └── IDENTIFIER(DataArray)
│   │   └── SEMICOLON(;)
```



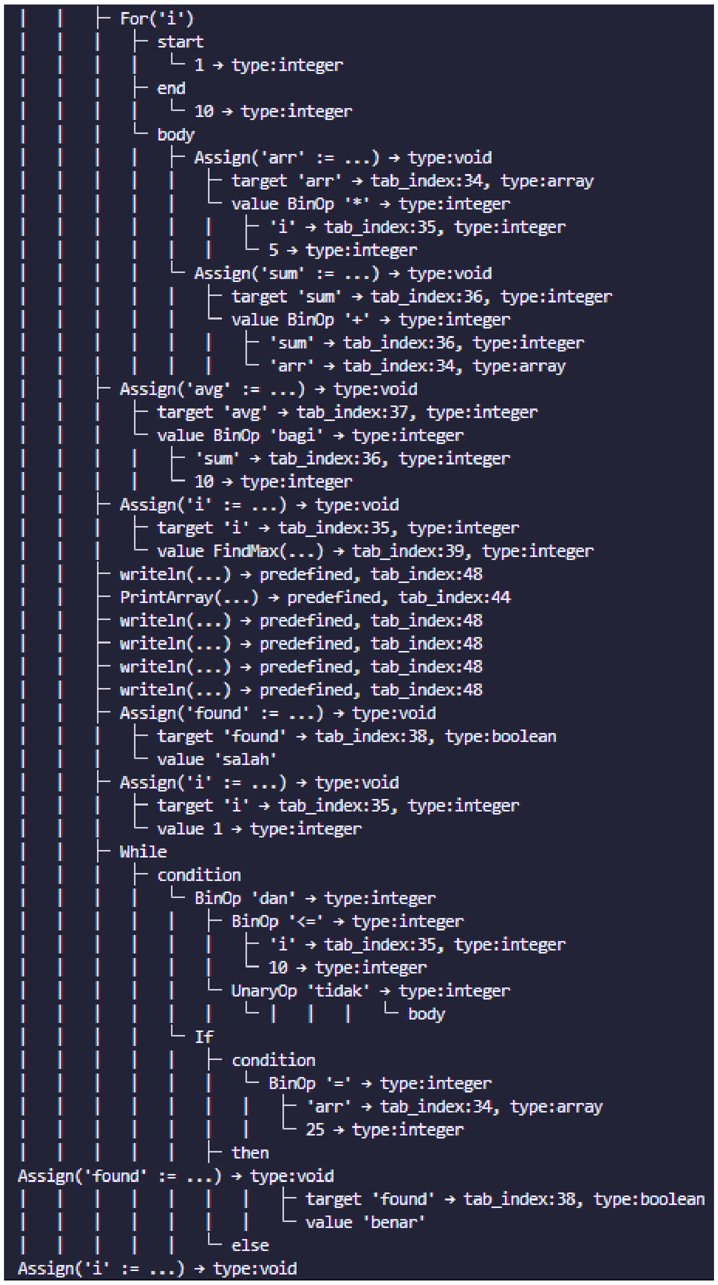
```
│   │   │       │       │           └── <term>
│   │   │       │       │               └── <factor>
│   │   │       │       │                   └── IDENTIFIER(benar)
│   │   │       │       ├── KEYWORD(selain-itu)
│   │   │       │       └── <assignment-statement>
│   │   │       │           ├── IDENTIFIER(i)
│   │   │       │           ├── ASSIGN_OPERATOR(:=)
│   │   │       │           └── <expression>
│   │   │       │               └── <simple-expression>
│   │   │       │                   ├── <term>
│   │   │       │                   │   └── <factor>
│   │   │       │                   │       └── IDENTIFIER(i)
│   │   │       │                   ├── <additive-operator>
│   │   │       │                   └── <term>
│   │   │       │                       └── <factor>
│   │   │       │                           └── NUMBER(1)
│   │   │       └── KEYWORD(selesai)
│   │   ├── SEMICOLON(;)
│   │   └── <if-statement>
│   │       ├── KEYWORD(jika)
│   │       ├── <expression>
│   │       │   └── <simple-expression>
│   │       │       └── <term>
│   │       │           └── <factor>
│   │       │               └── IDENTIFIER(found)
│   │       ├── KEYWORD(maka)
│   │       ├── <procedure/function-call>
│   │       │   ├── IDENTIFIER(writeln)
│   │       │   ├── LPARENTHESIS(()
│   │       │   ├── <parameter-list>
│   │       │   │   ├── <expression>
│   │       │   │   │   └── <simple-expression>
│   │       │   │   │       └── <term>
│   │       │   │   │           └── <factor>
│   │       │   │   │               └── STRING_LITERAL('Value 25 found at index ')
│   │       │   │   ├── COMMA(,)
│   │       │   │   └── <expression>
│   │       │   │       └── <simple-expression>
│   │       │   │           └── <term>
│   │       │   │               └── <factor>
│   │       │   │                   └── IDENTIFIER(i)
│   │       │   └── RPARENTHESIS())
│   │       ├── KEYWORD(selain-itu)
│   │       └── <procedure/function-call>
│   │           ├── IDENTIFIER(writeln)
│   │           ├── LPARENTHESIS(()
│   │           ├── <parameter-list>
│   │           │   └── <expression>
│   │           │       └── <simple-expression>
│   │           │           └── <term>
│   │           │               └── <factor>
│   │           │                   └── STRING_LITERAL('Value 25 not found')
│   │           └── RPARENTHESIS())
│   └── KEYWORD(selesai)
└── DOT(.)
```



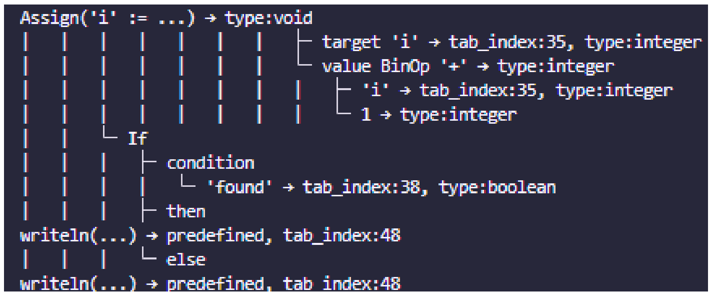
```
=== BUILDING AST ===
=== AST BUILT SUCCESSFULLY ===

=== BUILDING SYMBOL TABLE ===
[Semantic] Visiting Program: TestComprehensive
[Semantic] Program 'TestComprehensive' registered
[Semantic] Visiting Declaration Part
[Semantic] Declaring constant MAX_SIZE = 100
  - Constant 'MAX_SIZE' inserted at index 30
[Semantic] Declaring constant VERSION = 2
  - Constant 'VERSION' inserted at index 31
[Semantic] Declaring type IndexRange
  - Type 'IndexRange' inserted at index 32
[Semantic] Declaring type DataArray
  - Type 'DataArray' inserted at index 33
[Semantic] Declaring variables: arr (idx:34) : DataArray
[Semantic] Declaring variables: i,  (idx:35)sum,  (idx:36)avg (idx:37) : intege
[Semantic] Declaring variables: found (idx:38) : boolean
[Semantic] Declaring function FindMax
  - Function 'FindMax' returning 1 inserted at index 39, block 1
[Semantic] Visiting Declaration Part
[Semantic] Declaring variables: max,  (idx:42)idx (idx:43) : integer
[Semantic] Declaring procedure PrintArray
  - Procedure 'PrintArray' inserted at index 44, block 3
[Semantic] Visiting Declaration Part
[Semantic] Declaring variables: j (idx:47) : integer
[Semantic] Program 'TestComprehensive' checked successfully
=== SYMBOL TABLE BUILT ===


=== TAB (Identifier Table) ===
 idx          name  link       obj    typ   ref   nrm   lev   adr
-------------------------------------------------------------------
   0       program     0  constant      0     0     1     0     0
   1      variabel     0  constant      0     0     1     0     0
   2         mulai     0  constant      0     0     1     0     0
   3       selesai     0  constant      0     0     1     0     0
   4     konstanta     0  constant      0     0     1     0     0
   5          tipe     0  constant      0     0     1     0     0
   6      prosedur     0  constant      0     0     1     0     0
   7        fungsi     0  constant      0     0     1     0     0
   8          jika     0  constant      0     0     1     0     0
   9          maka     0  constant      0     0     1     0     0
  10    selain-itu     0  constant      0     0     1     0     0
  11         untuk     0  constant      0     0     1     0     0
  12            ke     0  constant      0     0     1     0     0
  13      turun-ke     0  constant      0     0     1     0     0
  14       lakukan     0  constant      0     0     1     0     0
  15        selama     0  constant      0     0     1     0     0
  16        ulangi     0  constant      0     0     1     0     0
  17        sampai     0  constant      0     0     1     0     0
  18         larik     0  constant      0     0     1     0     0
  19          dari     0  constant      0     0     1     0     0
  20       integer     0  constant      0     0     1     0     0
```



```
  21          real     0  constant      0     0     1     0     0
  22       boolean     0  constant      0     0     1     0     0
  23          char     0  constant      0     0     1     0     0
  24           dan     0  constant      0     0     1     0     0
  25          atau     0  constant      0     0     1     0     0
  26         tidak     0  constant      0     0     1     0     0
  27          bagi     0  constant      0     0     1     0     0
  28           mod     0  constant      0     0     1     0     0
  29TestComprehensive     0  procedure      0     0     1     0     0
  30      MAX_SIZE    29  constant      1     0     1     0   100
  31       VERSION    30  constant      1     0     1     0     2
  32    IndexRange    31      type      1     0     1     0     0
  33     DataArray    32      type      5     0     1     0     0
  34           arr    33  variable      5     0     1     0     0
  35             i    34  variable      1     0     1     0     1
  36           sum    35  variable      1     0     1     0     2
  37           avg    36  variable      1     0     1     0     3
  38         found    37  variable      3     0     1     0     4
  39       FindMax    38  function      1     1     1     0     0
  40          data     0  variable      5     0     1     1     0
  41          size    40  variable      1     0     1     1     0
  42           max    41  variable      1     0     1     1     0
  43           idx    42  variable      1     0     1     1     1
  44    PrintArray    39 procedure      0     3     1     0     0
  45          data     0  variable      5     0     1     1     0
  46         count    45  variable      1     0     1     1     0
  47             j    46  variable      1     0     1     1     2

=== BTAB (Block Table) ===
 idx    last    lpar   psize   vsize
-------------------------------------
   0      44       0       0       5
   1       0       0       0       3
   2      43       0       0       0
   3       0       0       0       0
   4      47       0       0       0

=== ATAB (Array Table) ===
 idx    xtyp    etyp    eref     low    high    elsz    size
-------------------------------------------------------------
   0       1       1       0       0      10       1      11

=== DECORATED AST ===
ProgramNode(name: 'TestComprehensive')
|   ├─ Declarations
|   |   ├─ VarDecl('arr') → tab_index:34, type:array, lev:0
|   |   ├─ VarDecl('i') → tab_index:35, type:integer, lev:0
|   |   ├─ VarDecl('sum') → tab_index:36, type:integer, lev:0
|   |   ├─ VarDecl('avg') → tab_index:37, type:integer, lev:0
|   |   └─ VarDecl('found') → tab_index:38, type:boolean, lev:0
|   └─ Block → block_index:0, lev:0
|   |   ├─ Assign('sum' := ...) → type:void
|   |   |   ├─ target 'sum' → tab_index:36, type:integer
|   |   |   └─ value 0 → type:integer
|   |   ├─ For('i')
```



```
│   │   ├ For('i')
│   │   │   ├ start
│   │   │   │   └ 1 → type:integer
│   │   │   ├ end
│   │   │   │   └ 10 → type:integer
│   │   │   └ body
│   │   │   │   ├ Assign('arr' := ...) → type:void
│   │   │   │   │   ├ target 'arr' → tab_index:34, type:array
│   │   │   │   │   └ value BinOp '*' → type:integer
│   │   │   │   │   │   ├ 'i' → tab_index:35, type:integer
│   │   │   │   │   │   └ 5 → type:integer
│   │   │   │   └ Assign('sum' := ...) → type:void
│   │   │   │   │   ├ target 'sum' → tab_index:36, type:integer
│   │   │   │   │   └ value BinOp '+' → type:integer
│   │   │   │   │   │   ├ 'sum' → tab_index:36, type:integer
│   │   │   │   │   │   └ 'arr' → tab_index:34, type:array
│   │   ├ Assign('avg' := ...) → type:void
│   │   │   ├ target 'avg' → tab_index:37, type:integer
│   │   │   └ value BinOp 'bagi' → type:integer
│   │   │   │   ├ 'sum' → tab_index:36, type:integer
│   │   │   │   └ 10 → type:integer
│   │   ├ Assign('i' := ...) → type:void
│   │   │   ├ target 'i' → tab_index:35, type:integer
│   │   │   └ value FindMax(...) → tab_index:39, type:integer
│   │   ├ writeln(...) → predefined, tab_index:48
│   │   ├ PrintArray(...) → predefined, tab_index:44
│   │   ├ writeln(...) → predefined, tab_index:48
│   │   ├ writeln(...) → predefined, tab_index:48
│   │   ├ writeln(...) → predefined, tab_index:48
│   │   ├ writeln(...) → predefined, tab_index:48
│   │   ├ Assign('found' := ...) → type:void
│   │   │   ├ target 'found' → tab_index:38, type:boolean
│   │   │   └ value 'salah'
│   │   ├ Assign('i' := ...) → type:void
│   │   │   ├ target 'i' → tab_index:35, type:integer
│   │   │   └ value 1 → type:integer
│   │   ├ While
│   │   │   ├ condition
│   │   │   │   └ BinOp 'dan' → type:integer
│   │   │   │   │   ├ BinOp '<=' → type:integer
│   │   │   │   │   │   ├ 'i' → tab_index:35, type:integer
│   │   │   │   │   │   └ 10 → type:integer
│   │   │   │   │   └ UnaryOp 'tidak' → type:integer
│   │   │   │   │   │   └ │   │   │   └ body
│   │   │   │   └ If
│   │   │   │   │   ├ condition
│   │   │   │   │   │   └ BinOp '=' → type:integer
│   │   │   │   │   │   │   ├ 'arr' → tab_index:34, type:array
│   │   │   │   │   │   │   └ 25 → type:integer
│   │   │   │   │   ├ then
Assign('found' := ...) → type:void
│   │   │   │   │   │   │   ├ target 'found' → tab_index:38, type:boolean
│   │   │   │   │   │   │   └ value 'benar'
│   │   │   │   │   └ else
Assign('i' := ...) → type:void
```



```
Assign('i' := ...) → type:void
│   │   │   │   │   │   │   ├ target 'i' → tab_index:35, type:integer
│   │   │   │   │   │   │   └ value BinOp '+' → type:integer
│   │   │   │   │   │   │   │   ├ 'i' → tab_index:35, type:integer
│   │   │   │   │   │   │   │   └ 1 → type:integer
│   │   └ If
│   │   │   ├ condition
│   │   │   │   └ 'found' → tab_index:38, type:boolean
│   │   │   ├ then
writeln(...) → predefined, tab_index:48
│   │   │   └ else
writeln(...) → predefined, tab_index:48
```

BAB IV

KESIMPULAN DAN SARAN

4.1. Kesimpulan

Berdasarkan seluruh rangkaian perancangan, implementasi, dan pengujian yang telah dilakukan pada *Milestone 3*, dapat disimpulkan bahwa *Semantic Analyzer* untuk *compiler* Pascal-S telah berhasil dibangun menggunakan pendekatan *L-Attributed Grammar*. Sistem ini terbukti mampu menjalankan fungsi utamanya yaitu memvalidasi konsistensi makna (*semantic*) program melalui mekanisme pengecekan deklarasi, validasi tipe data, dan pengecekan lingkup (*scope*). Output akhir berupa *Decorated Abstract Syntax Tree* (AST) berhasil dibentuk dengan ditambahkan atribut-atribut *semantic* seperti tipe data dan level *nesting*.

Implementasi tabel simbol dibagi menjadi tiga bagian (TAB, BTAB, ATAB) dan menggunakan *display stack* untuk mengatur *scope* yang bertingkat. *Visitor pattern* yang diterapkan pada kelas ScopeTypeChecker juga terbukti efektif dalam memisahkan logika traversal AST dari aturan semantik sehingga kode menjadi lebih terstruktur. Kualitas kode sudah berhasil diuji melalui 12 *test cases*. Hasilnya, compiler sukses memproses fitur-fitur yang cukup kompleks.

4.2. Saran

Meskipun sistem sudah berjalan sesuai spesifikasi, pengembangan selanjutnya sebaiknya difokuskan pada perbaikan kerapian struktur kode. Disarankan untuk melakukan refactoring atau pengorganisasian ulang kode di setiap akhir milestone agar lebih bersih dan modular. Hal ini penting untuk menjaga agar kode tetap mudah dibaca dan dipahami, serta memudahkan proses pelacakan error (debugging) saat fitur-fitur baru ditambahkan pada tahap-tahap selanjutnya.

LAMPIRAN

Link Release Repository Github

https://github.com/ahsuunn/NTB-Tubes-IF2224/releases/tag/v0.3.2

Pembagian Tugas (menunjukan persentase kontribusi)

NIM	Persentase Kerja	Tugas
13523018	25%	Implementasi Scope and Type Checking Deklarasi
13523062	25%	Implementasi *symbol table*
13523074	25%	Implementasi Scope and Type Checking Pernyataan
13523120	25%	Implementasi *Syntax-Directed Translation Scheme* dan AST builder beserta struktur dasarnya.