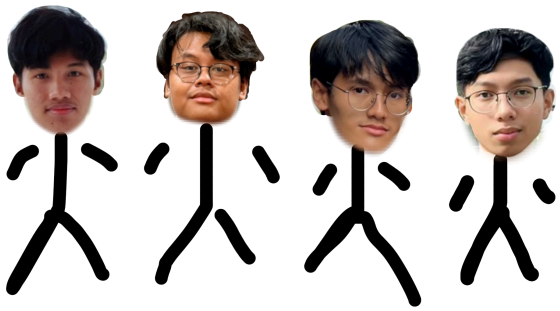
Tugas Besar IF2224 Teori Bahasa Formal & Otomata

Semantic Analysis



Oleh:

Kelompok JJK - JanganJanganKecompile

Muhammad Raihaan Perdana	13523124
Danendra Shafi Athallah	13523136
Nathanael Rachmat	13523142
M. Abizzar Gamadrian	13523155

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

2025

Daftar Isi

Bab I
Landasan Teori

1.1 Kompiler dan Tahapannya

Bahasa pemrograman merupakan jembatan komunikasi antara manusia dengan komputer. Agar instruksi yang ditulis dalam bahasa tingkat tinggi seperti Pascal-S dapat dimengerti dan dieksekusi oleh mesin, diperlukan sebuah program penerjemah yang disebut **kompiler** (*compiler*). Kompiler secara sistematis menerjemahkan kode sumber (*source code*) menjadi kode mesin (*machine code*) atau format tingkat rendah lainnya tanpa mengubah logika fungsionalitasnya. Proses ini tidak hanya memungkinkan eksekusi program, tetapi juga berperan penting dalam optimasi dan deteksi kesalahan dasar, sehingga menghasilkan aplikasi yang efisien dan andal.

Proses kompilasi secara umum terbagi menjadi beberapa tahapan utama yang berurutan, di mana keluaran dari satu tahap menjadi masukan bagi tahap berikutnya. Tahapan-tahapan tersebut adalah:

1. **Analisis Leksikal (*Lexical Analysis*):** Tahap awal yang memecah kode sumber menjadi unit-unit leksikal terkecil yang disebut *token*.
2. **Analisis Sintaksis (*Syntax Analysis*):** Memeriksa struktur gramatikal dari urutan *token* untuk memastikan sesuai dengan aturan sintaks bahasa pemrograman, biasanya dalam bentuk *parse tree*.
3. **Analisis Semantik (*Semantic Analysis*):** Memeriksa makna dan konsistensi dari program, seperti memeriksa kesesuaian tipe data dalam sebuah operasi.
4. **Generasi Kode Perantara (*Intermediate Code Generation*):** Menghasilkan representasi program dalam bentuk abstrak yang mudah dioptimasi dan diterjemahkan ke berbagai arsitektur mesin.
5. **Interpretasi (*Interpreter*):** Mengeksekusi kode, baik kode perantara maupun kode mesin, untuk menghasilkan *output* akhir.

Milestone ketiga dalam tugas besar ini berfokus pada implementasi tahap ketiga, yaitu Analisis Semantik.

1.2 Token

Token adalah unit leksikal terkecil dalam sebuah bahasa pemrograman yang memiliki makna. Setiap *token* yang dihasilkan oleh *lexer* umumnya terdiri dari dua komponen utama:

1. **Tipe Token (*Token Type*):** Kategori dari unit leksikal tersebut. Contohnya, KEYWORD, IDENTIFIER, NUMBER, atau ARITHMETIC_OPERATOR.
2. **Nilai Token (*Token Value*):** Rangkaian karakter aktual (disebut juga *lexeme*) yang cocok dengan pola tipe *token*. Contohnya, untuk *lexeme* "program", tipe *token*-nya adalah KEYWORD. Untuk *lexeme* "x", tipe *token*-nya adalah IDENTIFIER.

Sebagai ilustrasi, jika *lexer* memindai baris kode a := 5;, maka ia akan menghasilkan urutan *token* sebagai berikut: IDENTIFIER(a), ASSIGN_OPERATOR(:=), NUMBER(5), dan SEMICOLON(;)

1.3 Deterministic Finite Automaton (DFA)

Untuk mengenali pola-pola karakter dan mengelompokkannya menjadi *token*, *lexer* menggunakan model komputasi yang disebut **Deterministic Finite Automaton (DFA)**. Dalam teori bahasa formal, himpunan *token* dalam sebuah bahasa pemrograman dapat direpresentasikan sebagai **bahasa reguler**, yang secara matematis dapat dikenali oleh *finite automata*.

Secara formal, sebuah DFA didefinisikan sebagai 5-tuple $(Q, \Sigma, \delta, q_0, F)$, di mana:

- **Q** adalah himpunan hingga dari *state* (keadaan).
- **Σ** adalah himpunan hingga dari simbol input (alfabet).
- **δ** adalah fungsi transisi $(Q \times \Sigma \rightarrow Q)$.
- $\mathbf{q_0}$ adalah *state* awal.
- **F** adalah himpunan *state* akhir (penerima).

Dalam konteks *lexer*, DFA bekerja dengan membaca kode sumber karakter demi karakter. Dimulai dari *state* awal, *lexer* akan berpindah dari satu *state* ke *state* lain berdasarkan karakter yang dibaca dan fungsi transisi yang telah didefinisikan. Ketika *lexer* mencapai *state* akhir, itu menandakan bahwa serangkaian karakter yang telah dibaca berhasil dikenali sebagai sebuah *token* yang valid. Implementasi *lexer* berbasis DFA adalah contoh nyata bagaimana model teoretis dari automata digunakan untuk memecahkan masalah praktis dalam konstruksi kompiler.

1.4 Analisis Semantik

Analisis semantik merupakan tahap ketiga dalam proses kompilasi setelah analisis leksikal dan analisis sintaksis. Jika tahap sintaksis memastikan bahwa urutan token membentuk struktur yang sesuai dengan grammar bahasa, maka analisis semantik bertugas memastikan bahwa struktur tersebut memiliki

makna yang benar sesuai aturan bahasa pemrograman. Tahap ini sangat penting untuk mendeteksi kesalahan logis dalam program yang tidak bisa tertangkap oleh grammar saja.

Secara umum, analisis semantik mengambil masukan berupa syntax tree (atau abstract syntax tree / AST) dan menghasilkan decorated AST, yaitu pohon sintaks yang telah dilengkapi informasi semantik seperti tipe data, informasi scope, dan relasi lainnya.

Analisis semantik mengecek konsistensi semantik seperti:

- Variabel tidak boleh digunakan sebelum dideklarasikan.
- Variabel atau fungsi yang sama tidak boleh dideklarasikan dua kali dalam scope yang sama.
- Jumlah dan tipe argumen pada pemanggilan fungsi harus sesuai dengan definisinya.
- Operasi aritmatika hanya dilakukan pada tipe yang cocok (misalnya tidak boleh int * string).
- Ekspresi boolean harus digunakan pada kondisi if, while, dll.
- Tipe operand dalam ekspresi harus konsisten.

Analisis semantik adalah tahap terakhir yang dapat memberikan error ke user sebelum kode dijadikan representasi tingkat rendah, sehingga sangat penting untuk memastikan program valid. Untuk melakukan analisis semantik, compiler menggunakan konsep Syntax-Directed Definition (SDD), yaitu grammar yang diperluas dengan aturan untuk menghitung atribut.

Dua jenis atribut utama diperkenalkan:

a. **Synthesized Attributes**, yaitu atribut yang dihitung dari anak ke induk dalam parse tree. Contoh: E.val = E1.val + T.val

b. **Inherited Attributes,** yaitu atribut yang dihitung dari induk ke anak atau dari siblings di sebelah kiri. Digunakan misalnya untuk meneruskan tipe konteks ke child berikutnya.

   Melalui attributed grammar, compiler dapat menyimpan informasi seperti:

   - nilai ekspresi
   - tipe ekspresi
   - scope variabel
   - parameter fungsi
   - pointer ke entri di symbol table

AST adalah representasi sintaksis yang disederhanakan, berbeda dari parse tree yang mencerminkan grammar sepenuhnya. AST menghilangkan node-node yang tidak penting (seperti tanda kurung atau keyword tertentu), sehingga lebih mudah untuk dianalisis secara semantik.

Cara AST dibuat menggunakan SDD:

- Simbol terminal seperti id atau num menjadi Leaf Node.
- Operator seperti +, -, *, = menjadi internal nodes dengan anak-anak berupa operannya.

AST digunakan sebagai dasar untuk:

- pengecekan tipe,
- pengecekan scope,
- generasi kode perantara (intermediate code).

Bagian paling penting dalam analisis semantik adalah pengelolaan scope dan symbol table. Scope menentukan bagian program di mana suatu identifier (variabel, fungsi, parameter) dapat digunakan. Contoh bahasa yang memiliki nested scope:

```
begin
  int x;
  begin
    int x;     // variabel x yang berbeda
  end;
end;
```

Saat compiler memasuki scope baru, ia harus: membuat scope baru pada symbol table, menambahkan binding untuk deklarasi baru, menghapus binding saat scope selesai. Struktur data yang digunakan dapat berupa: list persisten, stack dengan marker, hash table per scope, hash table tunggal dengan penanda scope. Setiap lookup harus mencari dari scope paling dalam ke paling luar.

Type checking memastikan bahwa setiap operasi menggunakan tipe data yang benar. Compiler memberikan atribut type pada setiap node AST agar variabel, ekspresi, operator, dan pemanggilan fungsi dapat diverifikasi secara konsisten. Pada tahap ini, compiler memeriksa kecocokan tipe pada operasi aritmetika, boolean, assignment, kontrol alur, serta kesesuaian jumlah dan tipe argumen dalam pemanggilan fungsi. Dengan demikian, compiler dapat mendeteksi ketidaksesuaian tipe sebelum program dieksekusi.

Analisis semantik juga memvalidasi deklarasi variabel dan fungsi. Untuk variabel, compiler menentukan tipe inisialisasi (jika ada), mencatatnya dalam symbol table, dan membuat scope baru bila

diperlukan. Untuk fungsi, compiler mencatat tipe kembalian dan parameter, memasukkan parameter ke dalam scope fungsi, kemudian menentukan tipe body fungsi dan memastikan kesesuaiannya dengan tipe kembalian. Langkah ini menjamin bahwa variabel dan fungsi digunakan secara konsisten di seluruh program.

Bab II

Perancangan & Implementasi

2.1 Struktur Data Symbol Table

Untuk mengelola informasi mengenai identifier (variabel, konstanta, tipe, prosedur, fungsi), kami menggunakan struktur data Symbol Table yang terdiri dari tiga tabel utama yang saling terhubung. Pada tabel utama (tab) menyimpan atribut-atribut dari setiap identifier. Setiap entri (TabEntry) memiliki field:

tab	
Atribut	**Deskripsi**
name	Nama identifier (misalnya nama variabel, konstanta, tipe, prosedur, fungsi). Indeks dimulai dari 29 karena ada reserved words.
link	Pointer ke identifier sebelumnya dalam scope yang sama (untuk collision resolution/chaining).
obj	Jenis objek (Constant, Variable, Type, Procedure, Function, Program).
type	Tipe data (Integer, Real, Boolean, Char, String, atau referensi ke tipe bentukan).
ref	Referensi tambahan (misalnya ke ATAB untuk array, atau ukuran record).
nrm	Normal flag (untuk membedakan parameter by value/reference).
lev	Level scope dimana identifier dideklarasikan.
adr	Alamat memori relatif atau nilai (untuk konstanta).

Untuk menyimpan info spesifik untuk tipe data Array semua terpusat pada tabel array (atab). Field pada atab meliputi:

atab	
Atribut	**Deskripsi**
inxtyp	Tipe data indeks array.
eltyp	ipe data elemen array.
elref	Referensi tipe elemen.
low	Batas bawah indeks array.

high	Batas atas indeks array.
elsz	Ukuran satu elemen.
size	Total ukuran array.

Pada block tabel informasi mengenai blok atau scope program disimpan. Setiap kali compiler memasuki subprogram (procedure/function) atau program utama, entri baru dibuat di btab. Field meliputi:

btab	
Atribut	**Deskripsi**
blocks	Indeks entri block (setiap block mewakili prosedur, fungsi, atau record type definition).
last	Indeks identifier terakhir dalam TAB yang ada di scope ini.
lpar	Indeks parameter terakhir.
psze	Ukuran total parameter.
vsze	Ukuran total variabel lokal.

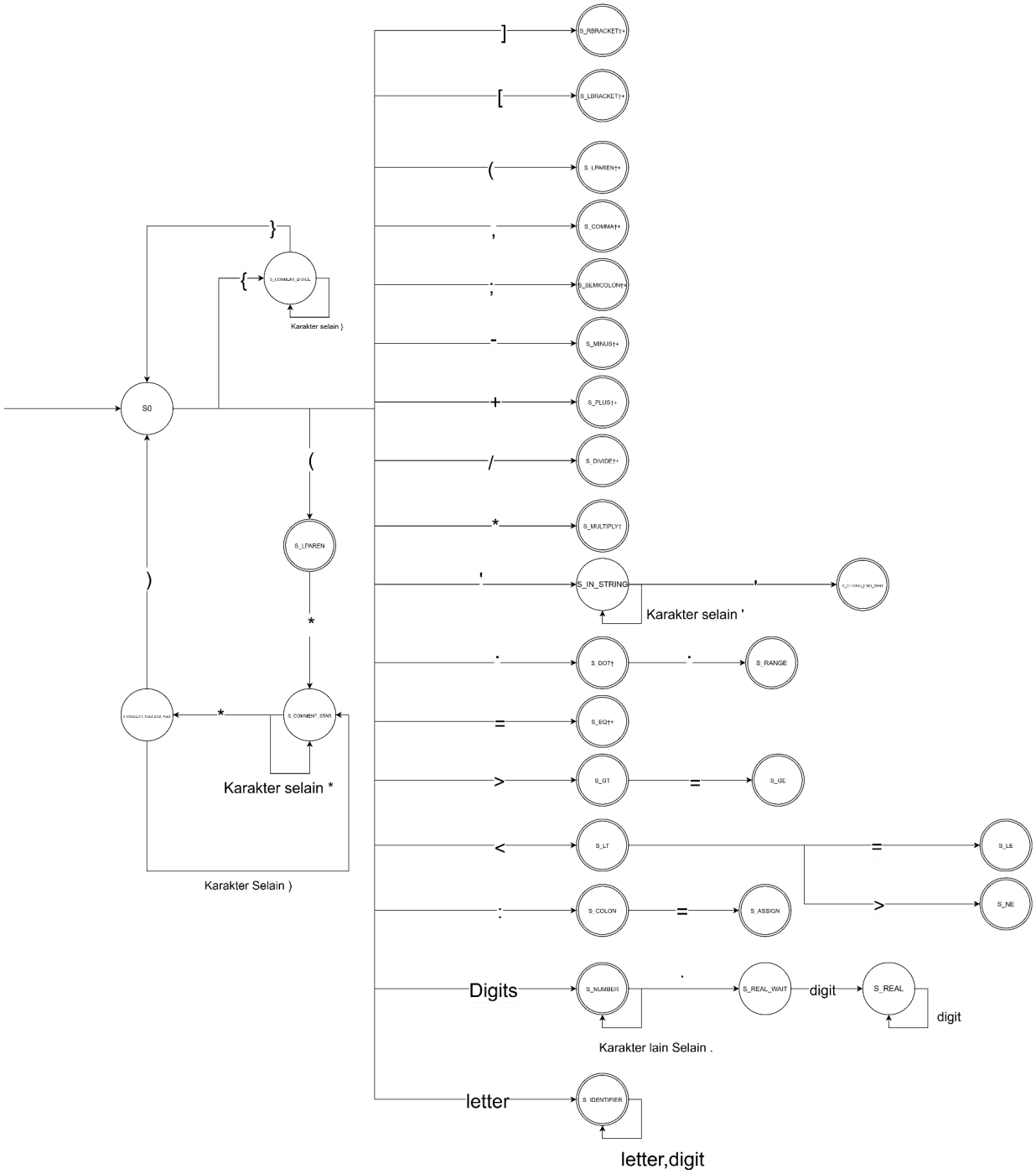


Diagram 1: Visualisasi Alur Transisi DFA

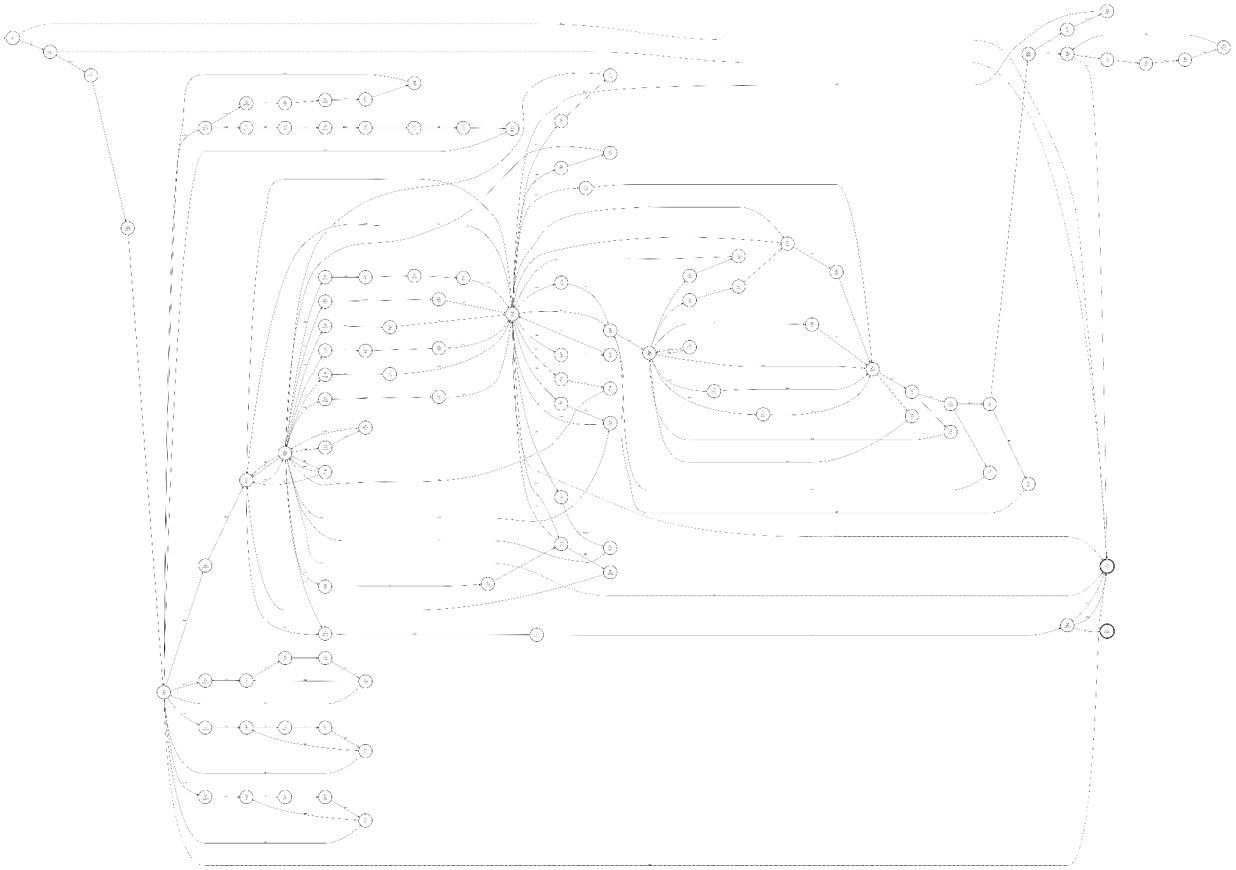


Diagram 2: Visualisasi Alur Syntax Analysis

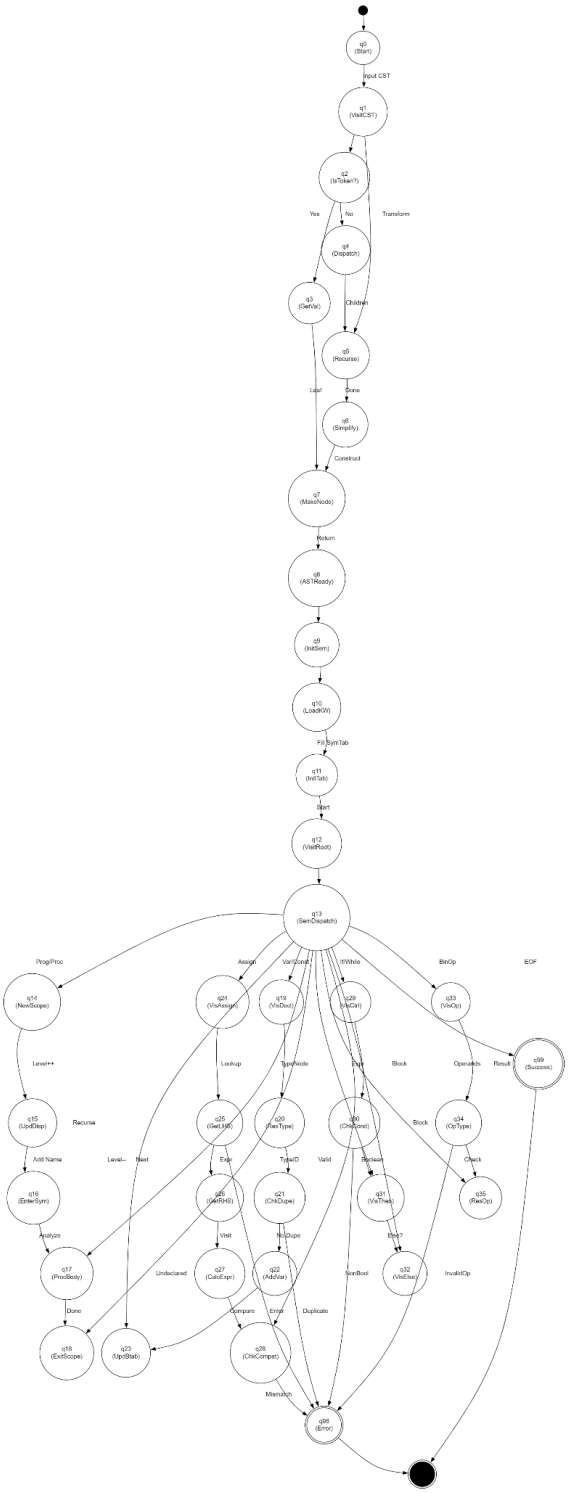


Diagram 3: Visualisasi Alur Semantic Analysis

2.2 Struktur Program

Implementasi Milestone 3 memodifikasi struktur program dari Milestone 1 dan 2 dengan menambahkan komponen Abstract Syntax Tree (AST) dan Semantic Analyzer. Struktur program terdiri dari 7 file utama dengan peran masing-masing sebagai berikut:

1. **compiler.py:** Bertindak sebagai *driver* atau titik masuk program. Skrip ini bertanggung jawab untuk memproses argumen *command-line*, membaca file sumber .pas, dan mengorkestrasi panggilan fungsi dari modul *lexer*.
2. **lexer.py:** Merupakan mesin utama dari *lexer*. Modul ini berisi semua logika inti, termasuk implementasi mesin DFA, fungsi untuk memuat aturan DFA dari file JSON, dan proses *tokenisasi* itu sendiri.
3. **dfa_rules.json:** File konfigurasi yang menyimpan definisi DFA untuk lexer. File ini berisi *start state, final states, transition table,* daftar *keywords, logical operators*, dan *arithmetic keywords* dalam bahasa Indonesia. Pemisahan konfigurasi DFA ke file JSON memudahkan modifikasi aturan lexical tanpa mengubah kode program.
4. **parser.py**: File ini merupakan inti dari Milestone 2. File berisi implementasi Recursive Descent Parser dengan beberapa kelas penting. Kelas Token merepresentasikan satu unit token dengan atribut tipe dan nilai. Kelas NodePohonParsing merepresentasikan node dalam parse tree dengan atribut nama node dan list anak-anaknya. Kelas Parser berisi logika parsing dengan method untuk setiap aturan grammar. Kelas SyntaxError digunakan untuk melaporkan kesalahan sintaksis dengan pesan error yang deskriptif.
5. **ast_nodes.py**: Berisi definisi kelas-kelas node untuk Abstract Syntax Tree (AST). Berbeda dengan CST yang merepresentasikan struktur sintaksis mentah, AST merepresentasikan struktur logika program yang lebih bersih dan siap untuk analisis semantik.
6. **ast_transformer.py**: Modul yang bertugas mengubah Concrete Syntax Tree (CST) hasil parsing menjadi Abstract Syntax Tree (AST). Proses ini menyederhanakan tree dengan membuang node yang tidak relevan secara semantik.
7. **ast_analyzer.py**: Inti dari Milestone 3. Modul ini mengimplementasikan Semantic Analyzer menggunakan pola Visitor. Bertugas melakukan pemeriksaan tipe (type checking), manajemen scope (symbol table), dan validasi aturan semantik lainnya pada AST.

   Alur kerja program secara keseluruhan berjalan sebagai berikut.

1. Pengguna menjalankan compiler.py dengan argumen berupa path file input dan optional flag.

2. Program memuat aturan DFA dari dfa_rules.json ke dalam memori.
3. Isi dari file kode .pas dibaca sebagai sebuah string tunggal.
4. Fungsi tokenize dari lexer.py dipanggil untuk memproses string kode tersebut karakter per karakter menggunakan mesin DFA yang telah dimuat.
5. Mesin DFA berpindah *state* sesuai dengan karakter input. Ketika sebuah token selesai dikenali, token tersebut ditambahkan ke dalam sebuah daftar.
6. Setelah seluruh kode diproses, daftar *token* yang dihasilkan akan ditulis ke dalam sebuah file output (misalnya, output-1.txt) di direktori yang sama dengan file input. Jika input berupa file .txt, program langsung membaca token dari file tersebut.
7. Setelah token tersedia, program membuat instance Parser dengan memasukkan list token.
8. Parser dijalankan dengan memanggil method parse yang mengembalikan root node dari parse tree.
9. Parse tree dicetak ke console menggunakan method cetak yang menghasilkan visualisasi tree dengan indentasi dan simbol cabang.
10. Jika terjadi syntax error, program menangkap exception dan menampilkan pesan error dengan informasi posisi dan token yang diharapkan.
11. Parse Tree (CST) dikonversi menjadi Abstract Syntax Tree (AST) menggunakan `ASTTransformer`. Struktur AST yang lebih ringkas kemudian dicetak.
12. Semantic Analyzer dijalankan pada AST. Analyzer menelusuri AST, membangun Symbol Table, dan memverifikasi aturan tipe serta scope.
13. Jika ditemukan pelanggaran semantik (seperti variabel belum dideklarasikan atau tipe tidak cocok), program akan menampilkan pesan error semantik dan berhenti.

Modifikasi dari Milestone 1 dan 2 meliputi penambahan tahap transformasi AST dan analisis semantik setelah proses parsing selesai, serta penggunaan struktur data Symbol Table yang lebih kompleks untuk mendukung pemeriksaan yang context-sensitive.

2.3 Implementasi Detail

Implementasi Semantic Analyzer terdiri dari beberapa komponen kunci yang bekerja sama untuk memverifikasi makna (semantics) program Pascal-S.

Potongan Kode 1:

```
class TabEntry:
    def __init__(self, name, obj, type_idx, ref, nrm, lev, adr, link):
```

```
        self.name = name    # Identifier name
        self.obj = obj      # Constant, Variable, etc.
          self.type = type_idx# Reference to type (Primitive ID or Tab
Index)
         self.ref = ref       # Reference to ATAB (if array) or Size (if
Record Type)
        self.nrm = nrm      # Normal (1) or Indirect/Param (0)
        self.lev = lev      # Scope Level
        self.adr = adr      # Memory Offset / Address / Value / Size
        self.link = link    # Pointer to previous symbol in same scope
```

Kelas `TabEntry` merupakan struktur data fundamental dalam Symbol Table yang berfungsi sebagai wadah penyimpanan informasi untuk setiap identifier yang dideklarasikan dalam program. Setiap atribut memiliki peran krusial: `obj` menentukan kategori identifier (apakah variabel, konstanta, atau prosedur), `type` menyimpan tipe data primitif atau referensi ke tipe bentukan, dan `lev` menandai kedalaman scope (level) di mana identifier tersebut berada. Atribut `link` sangat penting karena digunakan untuk menghubungkan identifier yang memiliki nilai hash sama atau berada dalam urutan deklarasi yang sama dalam satu scope, memungkinkan resolusi konflik dan traversal yang efisien. Selain itu, `adr` bersifat polimorfik, bisa menyimpan alamat memori relatif untuk variabel atau nilai aktual untuk konstanta.

Potongan Kode 2:

```
def analyze(self, node):
    if node is None: return T_NOTYPE
    method_name = f"visit_{node.__class__.__name__}"
    visitor = getattr(self, method_name, self.generic_visit)
    return visitor(node)
```

Method analyze adalah inti dari implementasi pola Visitor dalam Semantic Analyzer kami. Method ini memanfaatkan fitur introspeksi Python (`getattr`) untuk secara dinamis menentukan method kunjungan (`visit_*`) yang tepat berdasarkan nama kelas dari node AST yang sedang diproses. Pendekatan ini membuat kode sangat modular dan mudah diperluas; jika kita menambahkan jenis node baru di AST, kita cukup menambahkan method `visit_NewNode` yang sesuai tanpa perlu mengubah logika dispatch utama ini. Jika method spesifik tidak ditemukan, eksekusi akan dialihkan ke `generic_visit` untuk penanganan default.

Potongan Kode 3:

```
def enter(self, name, obj, type_idx, ref=0, nrm=1, adr=0):
    # ... (logika pengecekan duplikasi) ...
    new_entry = TabEntry(name, obj, type_idx, ref, nrm, self.level, adr,
link)
    self.tab.append(new_entry)
    return idx

def lookup(self, name):
    curr_lev = self.level
    while curr_lev >= 0:
        # ... (logika pencarian di display table) ...
            if entry.name == name:
                return curr_idx, entry
        curr_lev -= 1
    return 0, None
```

Dua method ini, `enter` dan `lookup`, merepresentasikan operasi inti dalam manajemen scope dan symbol table. Method `enter` bertugas mendaftarkan identifier baru ke dalam tabel; ia membuat objek `TabEntry` baru dengan informasi lengkap termasuk level scope saat ini, lalu menyisipkannya ke dalam list `tab`. Di sisi lain, `lookup` mengimplementasikan logika pencarian bertingkat yang dimulai dari scope terdalam (level saat ini) dan berjalan mundur hingga ke scope global (level 0). Algoritma ini memanfaatkan struktur `display` untuk melompat langsung ke blok yang relevan di setiap level, memastikan bahwa aturan shadowing variabel (variabel lokal menutupi variabel global dengan nama sama) ditegakkan dengan benar.

Untuk menangani nested scope (prosedur di dalam prosedur), kami menggunakan mekanisme 'Display':

- Variable `level` melacak kedalaman scope saat ini (0 = global/predefined, 1 = program utama, >1 = subprogram).
- Array `display` menyimpan indeks BTAB yang aktif untuk setiap level. Hal ini memungkinkan akses cepat ke identifier global maupun lokal yang valid pada scope saat ini.

Pemeriksaan tipe dilakukan secara statis saat traversing AST. Beberapa pemeriksaan kunci meliputi:

- Assignment Check: Memastikan tipe variabel tujuan (LHS) kompatibel dengan tipe ekspresi sumber (RHS). Misalnya, tidak boleh meng-assign nilai Real ke variabel Integer (kecuali dengan casting eksplisit, jika didukung), namun Integer ke Real diperbolehkan.
- Operation Check: Memastikan operand pada operasi aritmatika (+, -, *, /) bertipe numerik (Integer/Real). Operasi logika (and, or, not) hanya berlaku untuk Boolean.
- Parameter Check: Saat pemanggilan prosedur/fungsi, jumlah dan tipe argumen aktual dicocokkan dengan parameter formal yang dideklarasikan.

Bab III

Pengujian

3.1 Kasus Pengujian Unik

3.1.1 Uji Kompleksitas Program

test1_complex.pas

```
program test0;

konstanta
    ten = 10;
    plus = '+';

tipe
    row = larik [1..ten] dari real;
    complex = rekaman
        re, im: real
    selesai;

variabel
    i, j: integer;
    p: boolean;
    z: complex;
    matrix: larik [-3..+3] dari row;
    pattern: larik [1..5] dari larik [2..5] dari char;

prosedur dummy(variabel i: integer; variabel z: complex);
variabel
    u, v: row;
    h1, h2: rekaman
        c: complex;
        r: row
    selesai;

    fungsi null(x, y: real; z: complex): boolean;
    variabel
```

```
        a: larik ['a'..'z'] dari complex;
        u: char;
    mulai
        selama x < y lakukan
            x := x + 2;
        null := x = y
    selesai;

mulai
    p := null(h1.c.re, h2.c.im, z)
selesai;

mulai
    i := 85;
    j := 51;
    ulangi
        jika i > j maka
            i := i - j
        selain-itu
            j := j - i
    sampai i = j;
    writeln(i)
selesai.
```

Test case ini menguji kemampuan Semantic Analyzer dalam menangani struktur program yang kompleks namun valid secara semantik. Program ini mencakup deklarasi konstanta, tipe bentukan (array dan record), variabel global, serta subprogram bersarang (nested function dalam procedure). Tujuannya adalah memastikan bahwa compiler dapat membangun Symbol Table dengan benar untuk berbagai scope dan memvalidasi operasi yang melibatkan tipe data kompleks tanpa kesalahan.

3.1.2 Uji Deklarasi Ganda

test2_dupe_declaration.pas

```
program test0;

konstanta
```

```
    ten = 10;
    plus = '+';

tipe
    row = larik [1..ten] dari real;
    complex = rekaman
        re, im: real
    selesai;

variabel
    i, j: integer;
    p: boolean;
    z: complex;
    i: real;
    matrix: larik [-3..+3] dari row;
    pattern: larik [1..5] dari larik [2..5] dari char;

prosedur dummy(variabel i: integer; variabel z: complex);
variabel
    u, v: row;
    h1, h2: rekaman
        c: complex;
        r: row
    selesai;

    fungsi null(x, y: real; z: complex): boolean;
    variabel
        a: larik ['a'..'z'] dari complex;
        u: char;
    mulai
        selama x < y lakukan
            x := x + 2;
        null := x = y
    selesai;

mulai
    p := null(h1.c.re, h2.c.im, z)
```

```
selesai;

mulai
    i := 85;
    j := 51;
    ulangi
        jika i > j maka
            i := i - j
        selain-itu
            j := j - i
    sampai i = j;
    writeln(i)
selesai.
```

Dilakukan pengujian pada mekanisme deteksi deklarasi ganda. Pada kode di atas, variabel `i` dideklarasikan dua kali dalam scope yang sama (pertama sebagai integer, kemudian sebagai real). Semantic Analyzer diharapkan dapat mendeteksi konflik ini saat mencoba memasukkan entri kedua ke dalam Symbol Table dan membangkitkan pesan error yang sesuai, mencegah ambiguitas identifier.

3.1.3 Uji Identifier Tidak Dideklarasikan

test3_undeclared_iden.pas

```
program test0;

konstanta
    ten = 10;
    plus = '+';

tipe
    row = larik [1..ten] dari real;
    complex = rekaman
        re, im: real
    selesai;

variabel
    i, j: integer;
```

```
    p: boolean;
    z: complex;
    matrix: larik [-3..+3] dari row;
    pattern: larik [1..5] dari larik [2..5] dari char;

prosedur dummy(variabel i: integer; variabel z: complex);
variabel
    u, v: row;
    h1, h2: rekaman
        c: complex;
        r: row
    selesai;

    fungsi null(x, y: real; z: complex): boolean;
    variabel
        a: larik ['a'..'z'] dari complex;
        u: char;
    mulai
        selama x < y lakukan
            x := x + 2;
        null := x = y
    selesai;

mulai
    p := null(h1.c.re, h2.c.im, z)
selesai;

mulai
    i := 85;
    j := 51;
    ulangi
        jika i > j maka
            i := i - k
        selain-itu
            j := j - i
    sampai i = j;
    writeln(i)
```

```
selesai.
```

Pada test case diatas dilakukan uji pada validasi keberadaan identifier. Variabel `k` digunakan dalam ekspresi pengurangan `i - k` tanpa pernah dideklarasikan sebelumnya di bagian `variabel` atau parameter. Semantic Analyzer harus melakukan lookup ke semua scope yang valid, dan ketika `k` tidak ditemukan, ia wajib melaporkan error "Undeclared identifier".

3.1.4 Uji Ketidakcocokan Tipe Assignment

test4_type_mismatch.pas

```
program test0;

konstanta
    ten = 10;
    plus = '+';

tipe
    row = larik [1..ten] dari real;
    complex = rekaman
        re, im: real
    selesai;

variabel
    i, j: integer;
    p: boolean;
    z: complex;
    matrix: larik [-3..+3] dari row;
    pattern: larik [1..5] dari larik [2..5] dari char;

prosedur dummy(variabel i: integer; variabel z: complex);
variabel
    u, v: row;
    h1, h2: rekaman
        c: complex;
        r: row
    selesai;
```

```
    fungsi null(x, y: real; z: complex): boolean;
    variabel
        a: larik ['a'..'z'] dari complex;
        u: char;
    mulai
        selama x < y lakukan
            x := x + 2;
        null := x = y
    selesai;

mulai
    p := z
selesai;

mulai
    i := 85;
    j := 51;
    ulangi
        jika i > j maka
            i := i - j
        selain-itu
            j := j - i
    sampai i = j;
    writeln(i)
selesai.
```

Test case ini mengetes ketatnya aturan tipe pada operasi assignment. Variabel `p` bertipe `boolean`, sedangkan `z` bertipe `complex` (record). Karena kedua tipe ini tidak kompatibel dan tidak ada aturan konversi implisit antar keduanya, Semantic Analyzer harus menolak operasi ini dengan pesan error "Type mismatch", menjaga integritas tipe data dalam program.

3.1.5 Uji Operasi Tipe Invalid

test5_invalid_type.pas

```
program test0;
```

```
konstanta
    ten = 10;
    plus = '+';

tipe
    row = larik [1..ten] dari real;
    complex = rekaman
        re, im: real
    selesai;

variabel
    i, j: integer;
    p: boolean;
    z: complex;
    matrix: larik [-3..+3] dari row;
    pattern: larik [1..5] dari larik [2..5] dari char;

prosedur dummy(variabel i: integer; variabel z: complex);
variabel
    u, v: row;
    h1, h2: rekaman
        c: complex;
        r: row
    selesai;

    fungsi null(x, y: real; z: complex): boolean;
    variabel
        a: larik ['a'..'z'] dari complex;
        u: char;
    mulai
        selama x < y lakukan
            x := x + 2;
        null := x + p
    selesai;

mulai
```

```
    p := null(h1.c.re, h2.c.im, z)
selesai;

mulai
    i := 85;
    j := 51;
    ulangi
        jika i > j maka
            i := i - j
        selain-itu
            j := j - i
    sampai i = j;
    writeln(i)
selesai.
```

Test case ini menguji validasi tipe operand dalam ekspresi aritmatika. Ekspresi `x + p` mencoba menjumlahkan variabel `x` (real) dengan `p` (boolean). Operasi penjumlahan hanya valid untuk tipe numerik. Semantic Analyzer bertugas memeriksa tipe setiap operand sebelum operasi dilakukan dan mendeteksi bahwa penjumlahan antara real dan boolean adalah ilegal.

3.1.6 Uji Kondisi Non-Boolean

test6_no_boolean.pas

```
program test0;

konstanta
    ten = 10;
    plus = '+';

tipe
    row = larik [1..ten] dari real;
    complex = rekaman
        re, im: real
    selesai;

variabel
```

```
    i, j: integer;
    p: boolean;
    z: complex;
    matrix: larik [-3..+3] dari row;
    pattern: larik [1..5] dari larik [2..5] dari char;

prosedur dummy(variabel i: integer; variabel z: complex);
variabel
    u, v: row;
    h1, h2: rekaman
        c: complex;
        r: row
    selesai;

    fungsi null(x, y: real; z: complex): boolean;
    variabel
        a: larik ['a'..'z'] dari complex;
        u: char;
    mulai
        selama x + y lakukan
        { x + y menghasilkan real }
            x := x + 2;
        null := x = y
    selesai;

mulai
    p := null(h1.c.re, h2.c.im, z)
selesai;

mulai
    i := 85;
    j := 51;
    ulangi
        jika i > j maka
            i := i - j
        selain-itu
            j := j - i
```

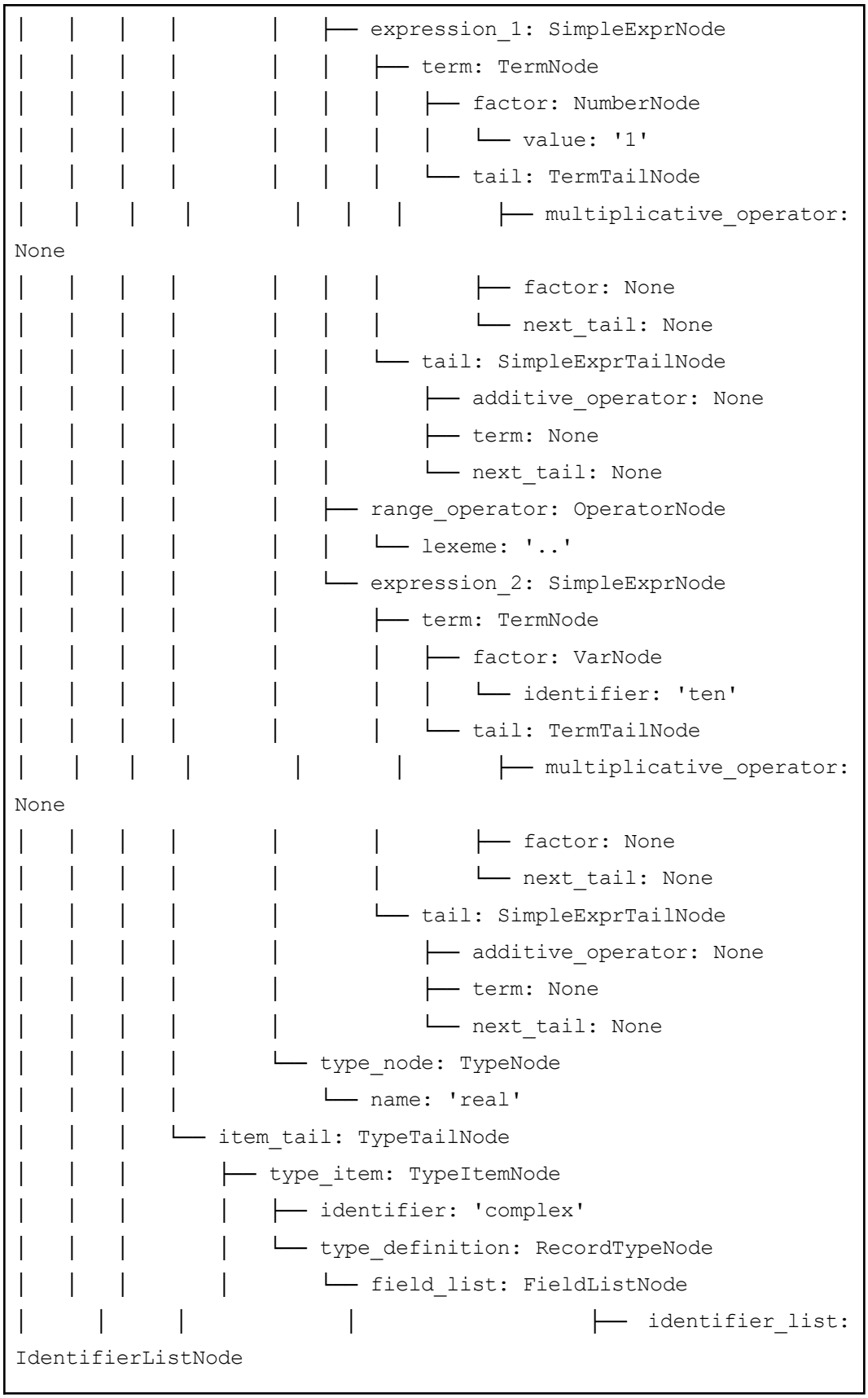
```
    sampai i = j;
    writeln(i)
selesai.
```

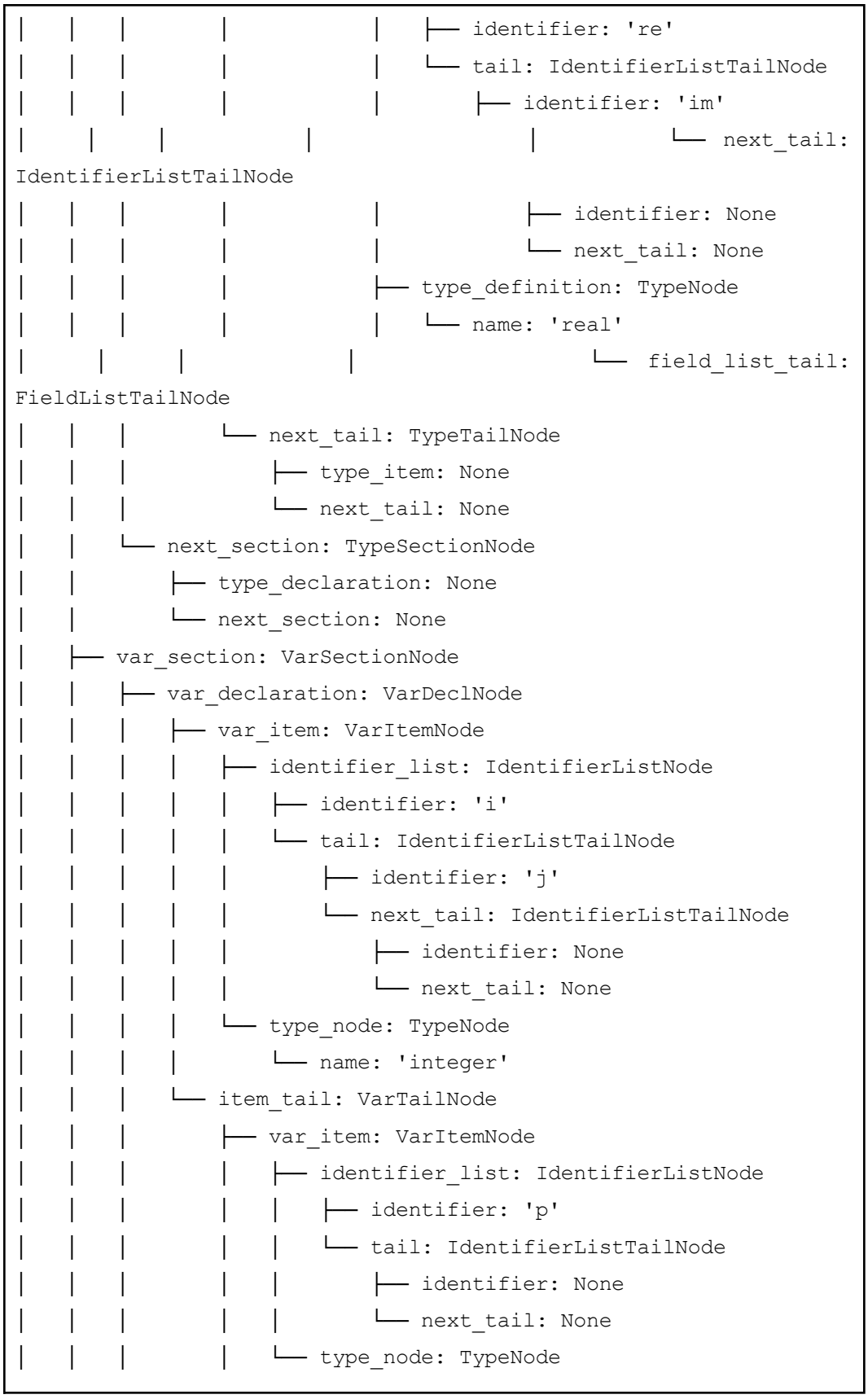
Test case ini memastikan bahwa ekspresi yang digunakan sebagai kondisi pada struktur kontrol (seperti `selama` atau `jika`) harus mengevaluasi ke tipe boolean. Pada contoh ini, `x + y` menghasilkan nilai numerik (real). Semantic Analyzer harus memverifikasi tipe hasil ekspresi kondisi dan menolak program karena `real` tidak dapat digunakan sebagai kondisi logika.

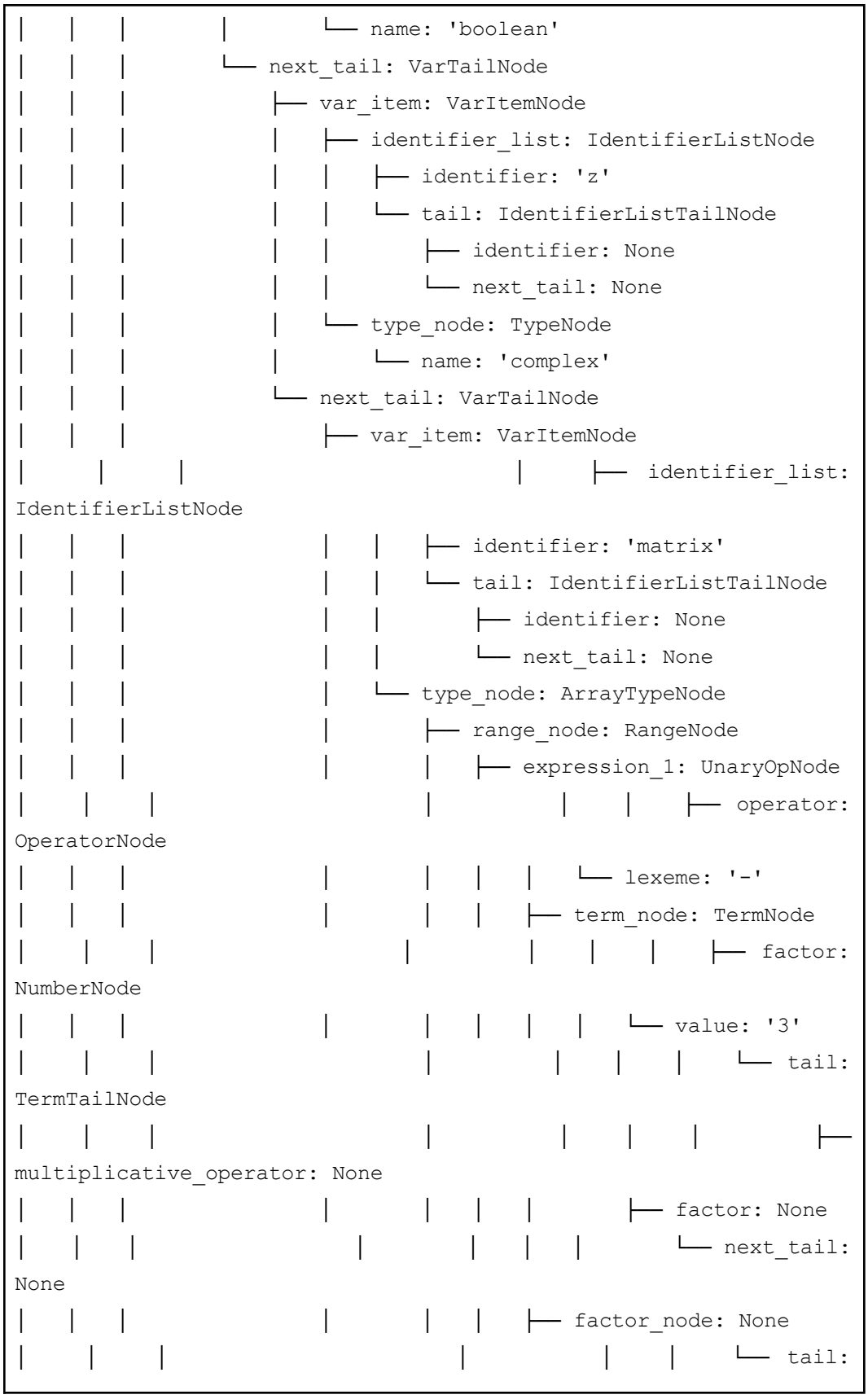
3.2 Hasil dan Pembahasan Pengujian

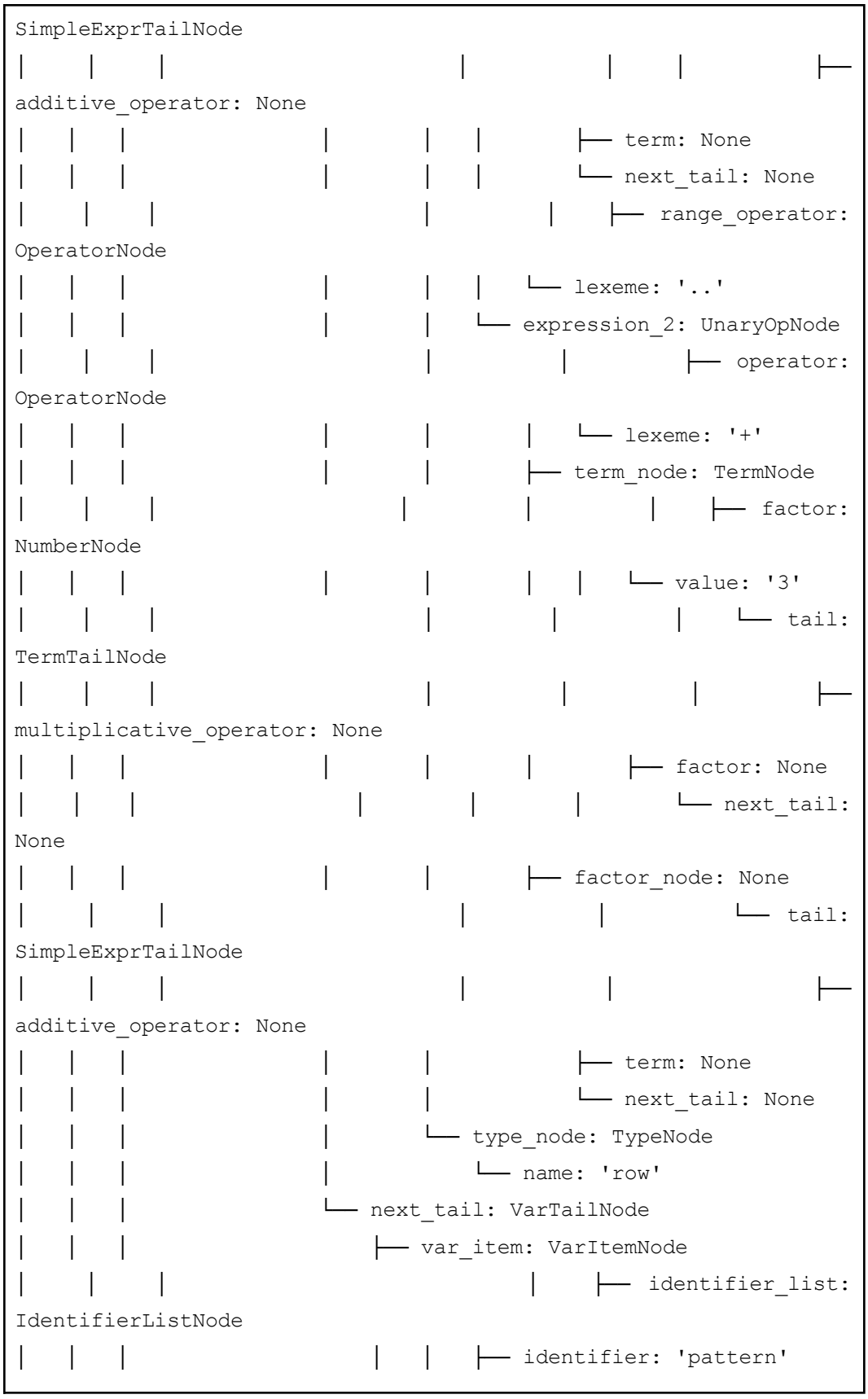
3.2.1 Hasil Uji 1

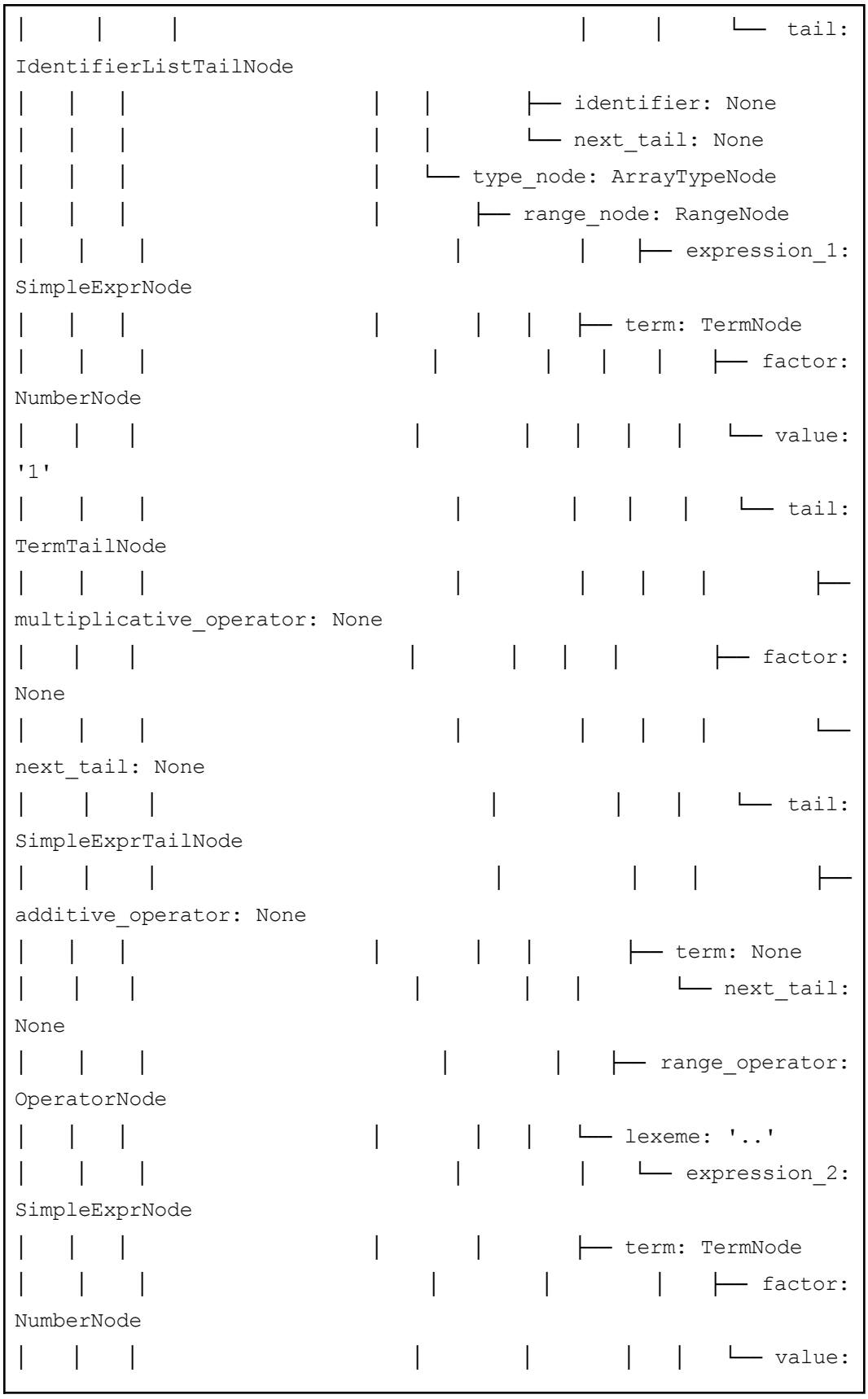
```
================================================
Tahap: AST Generation (Abstract Syntax Tree)
================================================
AST Structure:
------------------------------
ProgramNode
├── program_header: ProgramHeaderNode
│   └── identifier: 'test0'
├── declaration_part: DeclarationPartNode
│   ├── const_section: ConstSectionNode
│   │   ├── const_declaration: ConstDeclNode
│   │   │   ├── const_item: ConstItemNode
│   │   │   │   ├── identifier: 'ten'
│   │   │   │   ├── assign_operator: '='
│   │   │   │   └── value_node: NumberNode
│   │   │   │       └── value: '10'
│   │   │   └── item_tail: ConstTailNode
│   │   │       ├── const_item: ConstItemNode
│   │   │       │   ├── identifier: 'plus'
│   │   │       │   ├── assign_operator: '='
│   │   │       │   └── value_node: StringNode
│   │   │       │       └── value: ''+''
│   │   │       └── next_tail: ConstTailNode
│   │   │           ├── const_item: None
│   │   │           └── next_tail: None
│   │   └── next_section: ConstSectionNode
│   │       ├── const_declaration: None
│   │       └── next_section: None
│   ├── type_section: TypeSectionNode
│   │   ├── type_declaration: TypeDeclNode
│   │   │   ├── type_item: TypeItemNode
│   │   │   │   ├── identifier: 'row'
│   │   │   │   └── type_definition: ArrayTypeNode
│   │   │   │       ├── range_node: RangeNode
```

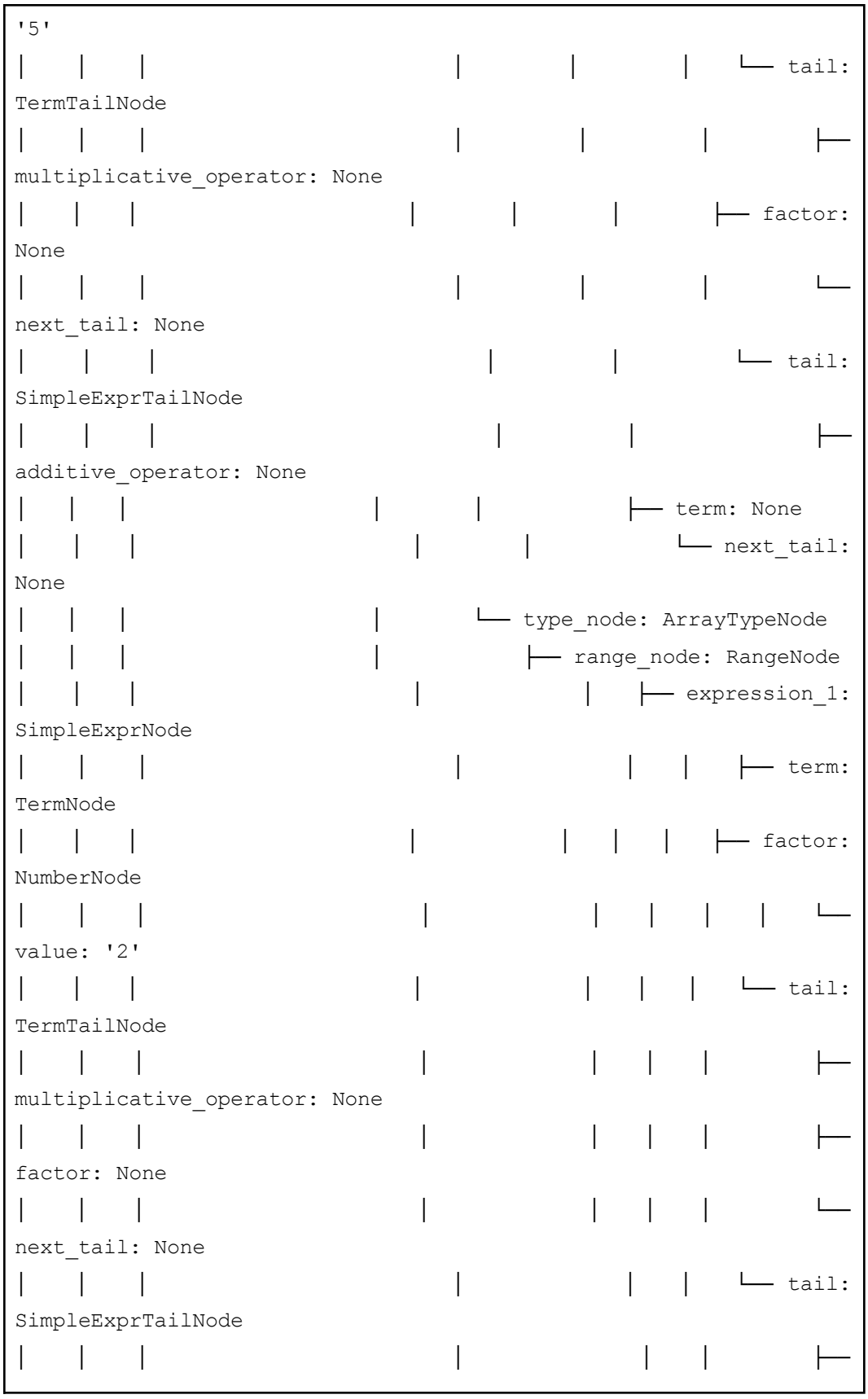

| | | | | ├── expression_1: SimpleExprNode
| | | | | | ├── term: TermNode
| | | | | | | ├── factor: NumberNode
| | | | | | | | └── value: '1'
| | | | | | | └── tail: TermTailNode
| | | | | | | ├── multiplicative_operator: None
| | | | | | | ├── factor: None
| | | | | | | └── next_tail: None
| | | | | | └── tail: SimpleExprTailNode
| | | | | | ├── additive_operator: None
| | | | | | ├── term: None
| | | | | | └── next_tail: None
| | | | | ├── range_operator: OperatorNode
| | | | | | └── lexeme: '..'
| | | | | └── expression_2: SimpleExprNode
| | | | | ├── term: TermNode
| | | | | | ├── factor: VarNode
| | | | | | | └── identifier: 'ten'
| | | | | | └── tail: TermTailNode
| | | | | | ├── multiplicative_operator: None
| | | | | | ├── factor: None
| | | | | | └── next_tail: None
| | | | | └── tail: SimpleExprTailNode
| | | | | ├── additive_operator: None
| | | | | ├── term: None
| | | | | └── next_tail: None
| | | | └── type_node: TypeNode
| | | | └── name: 'real'
| | | └── item_tail: TypeTailNode
| | | ├── type_item: TypeItemNode
| | | | ├── identifier: 'complex'
| | | | └── type_definition: RecordTypeNode
| | | | └── field_list: FieldListNode
| | | | ├── identifier_list: IdentifierListNode

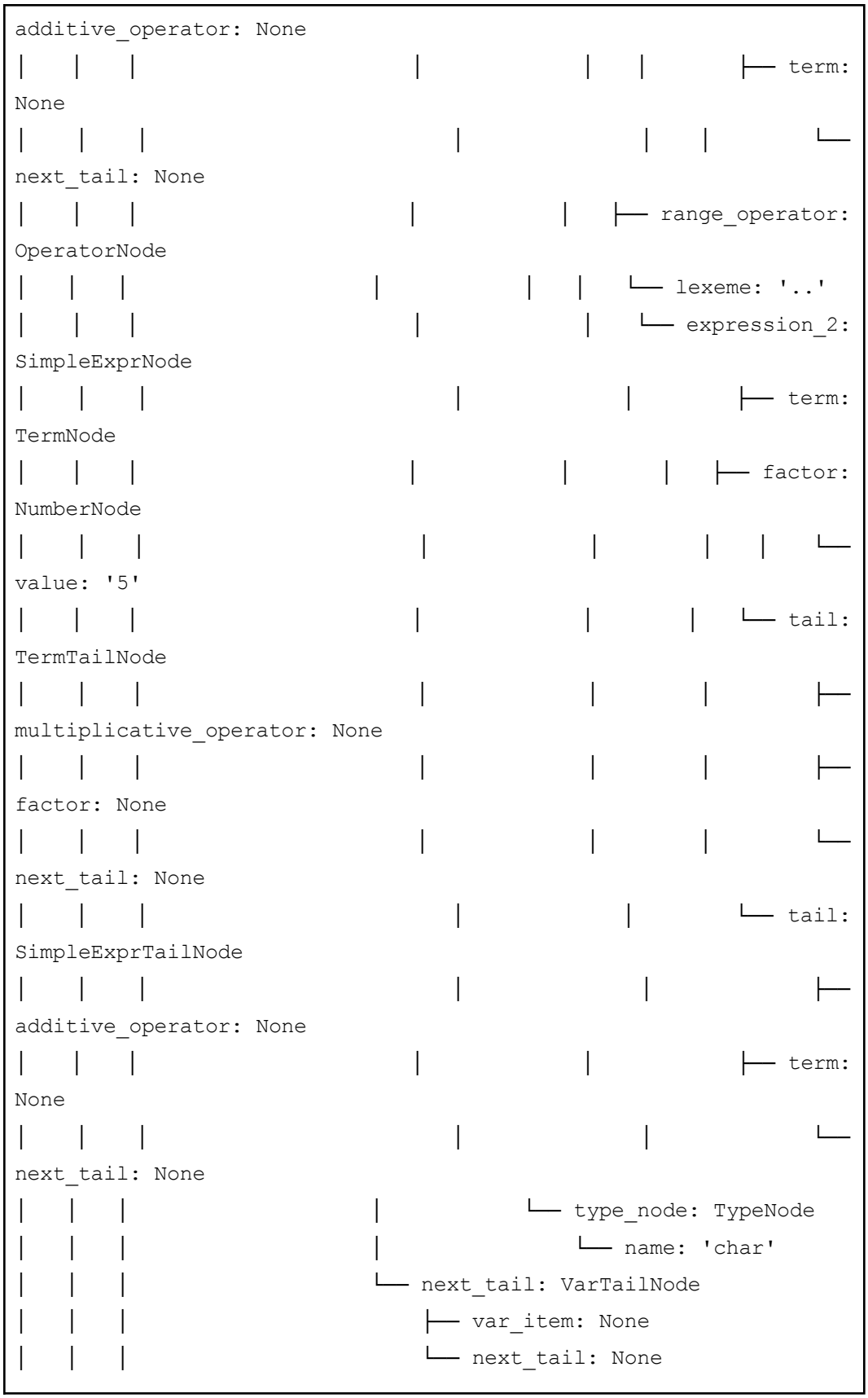

| | | | | | ├── identifier: 're'
| | | | | | └── tail: IdentifierListTailNode
| | | | | | ├── identifier: 'im'
| | | | | | └── next_tail:
IdentifierListTailNode
| | | | | | ├── identifier: None
| | | | | | └── next_tail: None
| | | | | ├── type_definition: TypeNode
| | | | | | └── name: 'real'
| | | | | └── field_list_tail:
FieldListTailNode
| | | | └── next_tail: TypeTailNode
| | | | ├── type_item: None
| | | | └── next_tail: None
| | | └── next_section: TypeSectionNode
| | | ├── type_declaration: None
| | | └── next_section: None
| | ├── var_section: VarSectionNode
| | | ├── var_declaration: VarDeclNode
| | | | ├── var_item: VarItemNode
| | | | | ├── identifier_list: IdentifierListNode
| | | | | | ├── identifier: 'i'
| | | | | | └── tail: IdentifierListTailNode
| | | | | | ├── identifier: 'j'
| | | | | | └── next_tail: IdentifierListTailNode
| | | | | | ├── identifier: None
| | | | | | └── next_tail: None
| | | | | └── type_node: TypeNode
| | | | | └── name: 'integer'
| | | | └── item_tail: VarTailNode
| | | | ├── var_item: VarItemNode
| | | | | ├── identifier_list: IdentifierListNode
| | | | | | ├── identifier: 'p'
| | | | | | └── tail: IdentifierListTailNode
| | | | | | ├── identifier: None
| | | | | | └── next_tail: None
| | | | | └── type_node: TypeNode


| | | | └── name: 'boolean'
| | | └── next_tail: VarTailNode
| | | ├── var_item: VarItemNode
| | | | ├── identifier_list: IdentifierListNode
| | | | | ├── identifier: 'z'
| | | | | └── tail: IdentifierListTailNode
| | | | | ├── identifier: None
| | | | | └── next_tail: None
| | | | └── type_node: TypeNode
| | | | └── name: 'complex'
| | | └── next_tail: VarTailNode
| | | ├── var_item: VarItemNode
| | | | ├── identifier_list:
IdentifierListNode
| | | | | ├── identifier: 'matrix'
| | | | | └── tail: IdentifierListTailNode
| | | | | ├── identifier: None
| | | | | └── next_tail: None
| | | | └── type_node: ArrayTypeNode
| | | | ├── range_node: RangeNode
| | | | | ├── expression_1: UnaryOpNode
| | | | | | ├── operator:
OperatorNode
| | | | | | | └── lexeme: '-'
| | | | | | ├── term_node: TermNode
| | | | | | | ├── factor:
NumberNode
| | | | | | | | └── value: '3'
| | | | | | | └── tail:
TermTailNode
| | | | | | | ├──
multiplicative_operator: None
| | | | | | | ├── factor: None
| | | | | | | └── next_tail:
None
| | | | | | ├── factor_node: None
| | | | | | └── tail:

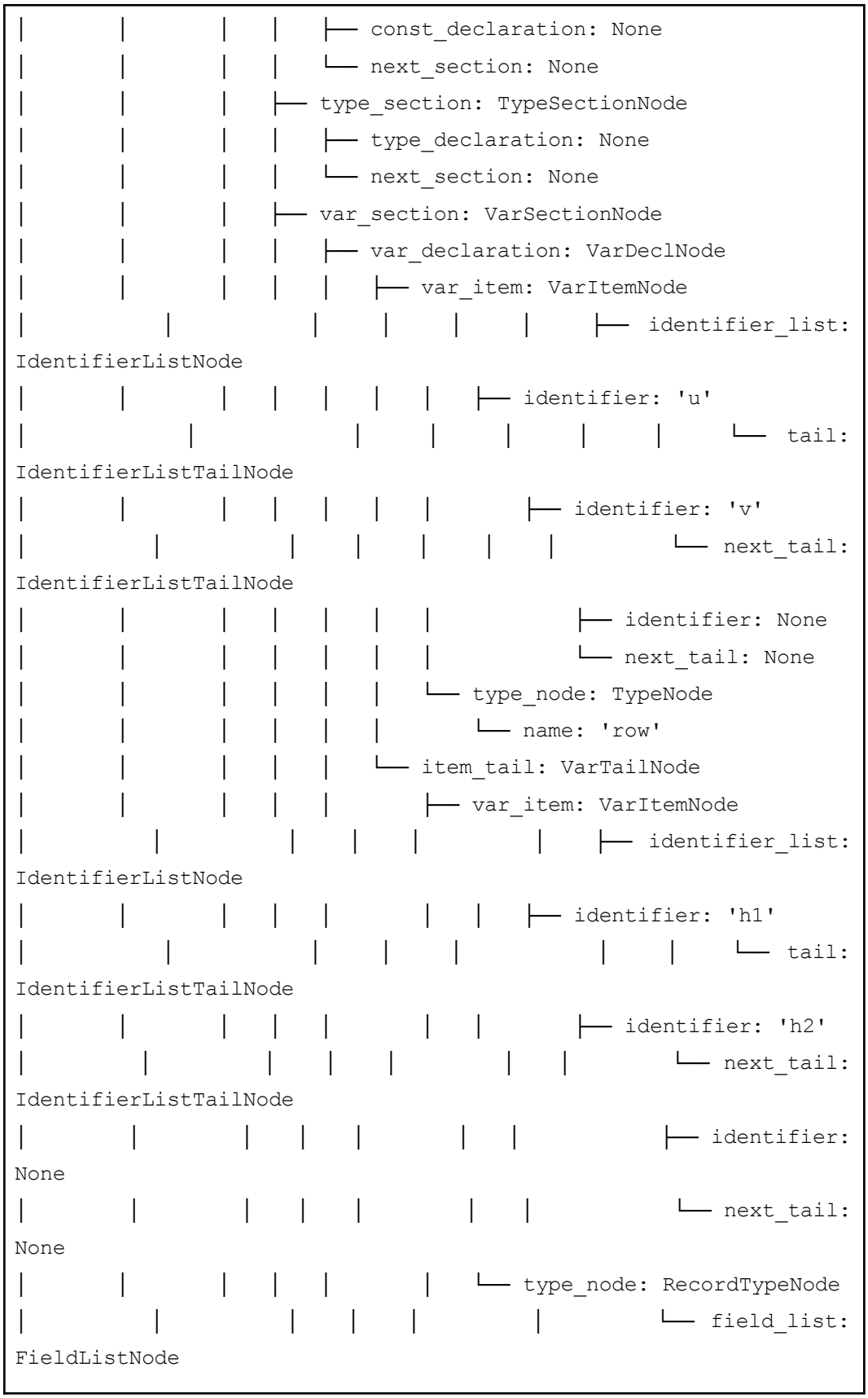

SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
range_operator: OperatorNode
lexeme: '..'
expression_2: UnaryOpNode
operator: OperatorNode
lexeme: '+'
term_node: TermNode
factor: NumberNode
value: '3'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
factor_node: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
type_node: TypeNode
name: 'row'
next_tail: VarTailNode
var_item: VarItemNode
identifier_list: IdentifierListNode
identifier: 'pattern'

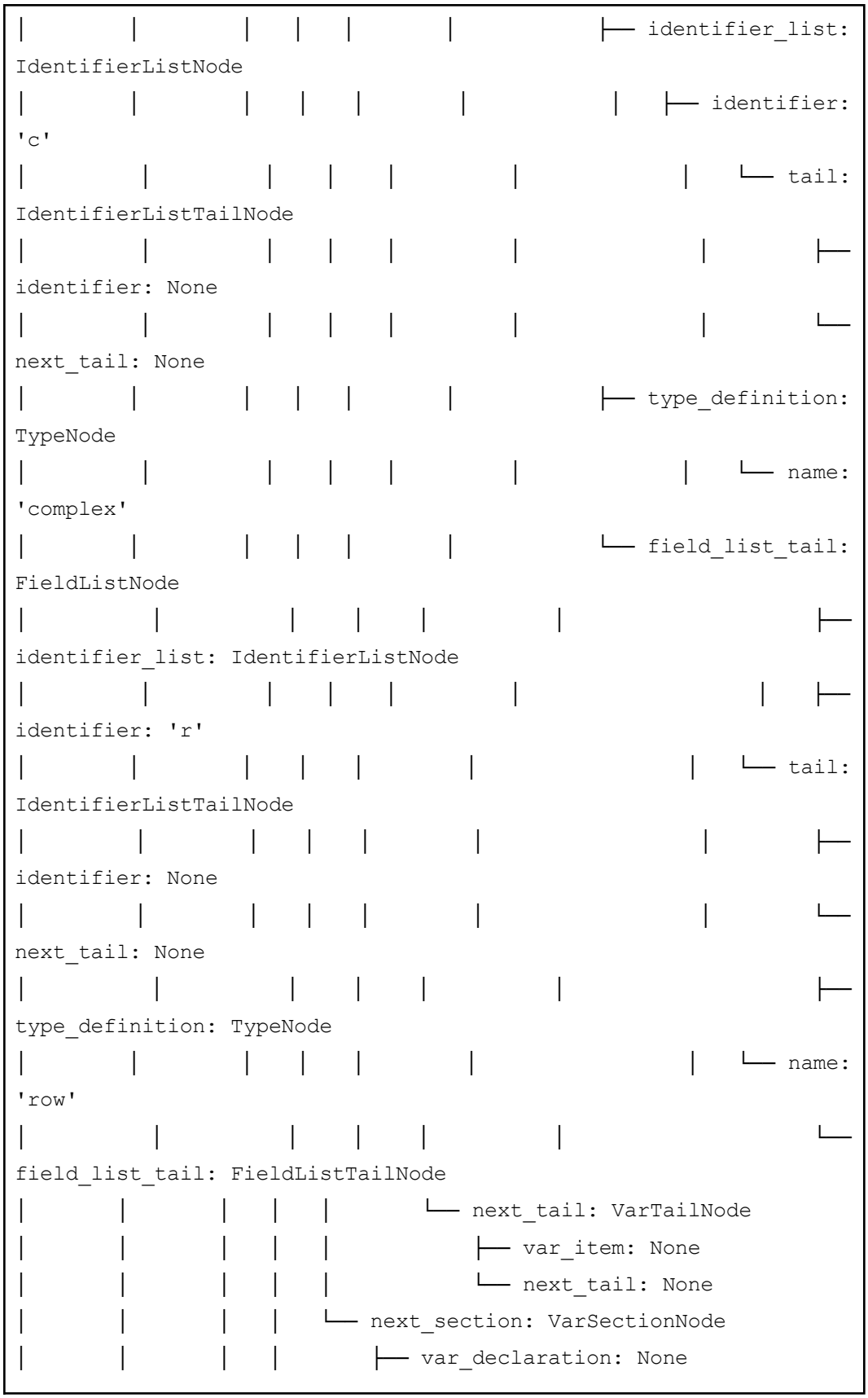

| | | | | | └── tail:
IdentifierListTailNode
| | | | | ├── identifier: None
| | | | | └── next_tail: None
| | | | └── type_node: ArrayTypeNode
| | | | ├── range_node: RangeNode
| | | | | | ├── expression_1:
SimpleExprNode
| | | | | | | ├── term: TermNode
| | | | | | | | ├── factor:
NumberNode
| | | | | | | | └── value:
'1'
| | | | | | | └── tail:
TermTailNode
| | | | | | | ├──
multiplicative_operator: None
| | | | | | ├── factor:
None
| | | | | | └──
next_tail: None
| | | | | └── tail:
SimpleExprTailNode
| | | | | ├──
additive_operator: None
| | | | ├── term: None
| | | | └── next_tail:
None
| | | | ├── range_operator:
OperatorNode
| | | | ├── lexeme: '..'
| | | | └── expression_2:
SimpleExprNode
| | | | ├── term: TermNode
| | | | | ├── factor:
NumberNode
| | | | └── value:


'5'
tail:
TermTailNode
multiplicative_operator: None
factor:
None
next_tail: None
tail:
SimpleExprTailNode
additive_operator: None
term: None
next_tail:
None
type_node: ArrayTypeNode
range_node: RangeNode
expression_1:
SimpleExprNode
term:
TermNode
factor:
NumberNode
value: '2'
tail:
TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail:
SimpleExprTailNode

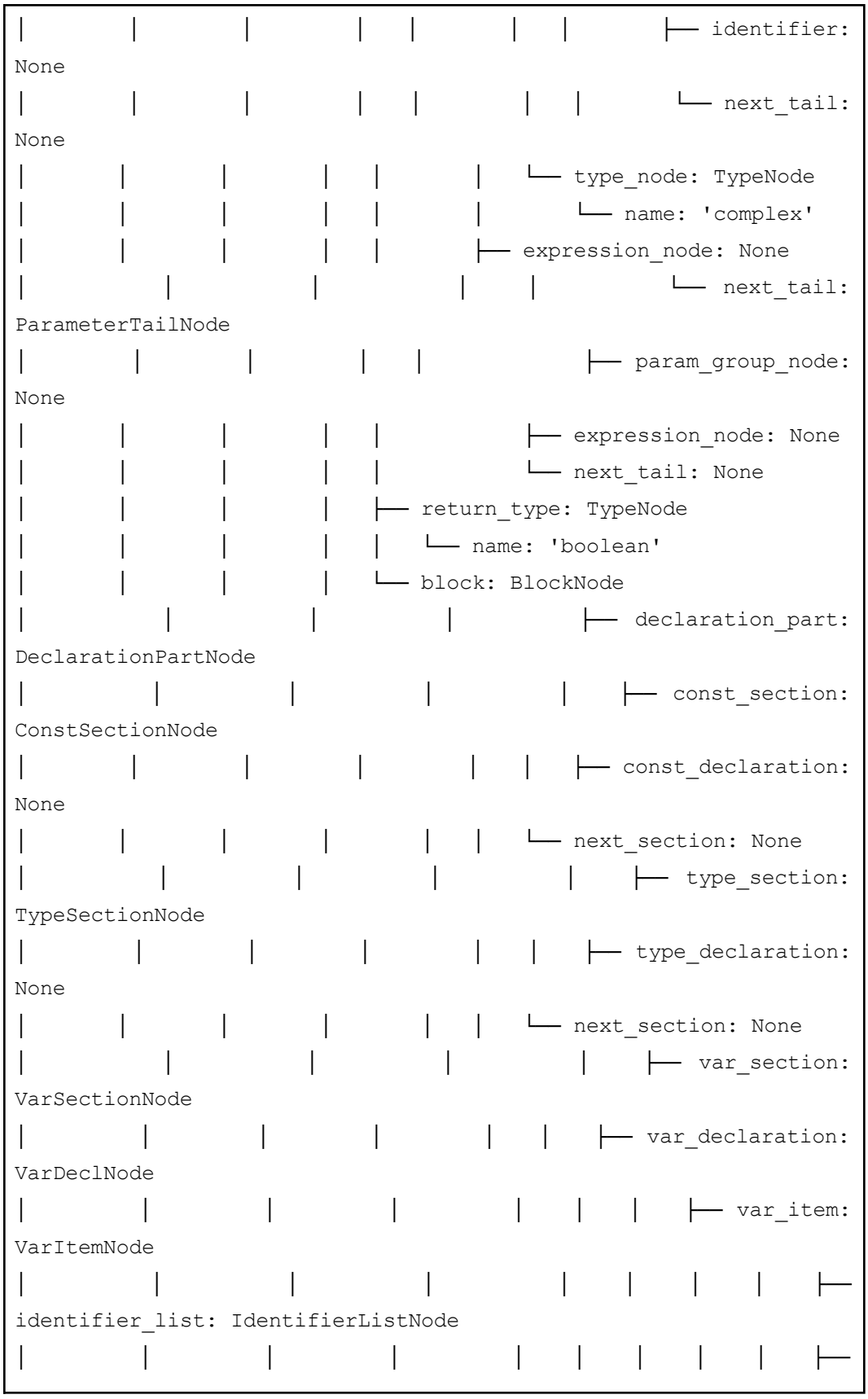

additive_operator: None
term: None
next_tail: None
range_operator: OperatorNode
lexeme: '..'
expression_2: SimpleExprNode
term: TermNode
factor: NumberNode
value: '5'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
type_node: TypeNode
name: 'char'
next_tail: VarTailNode
var_item: None
next_tail: None

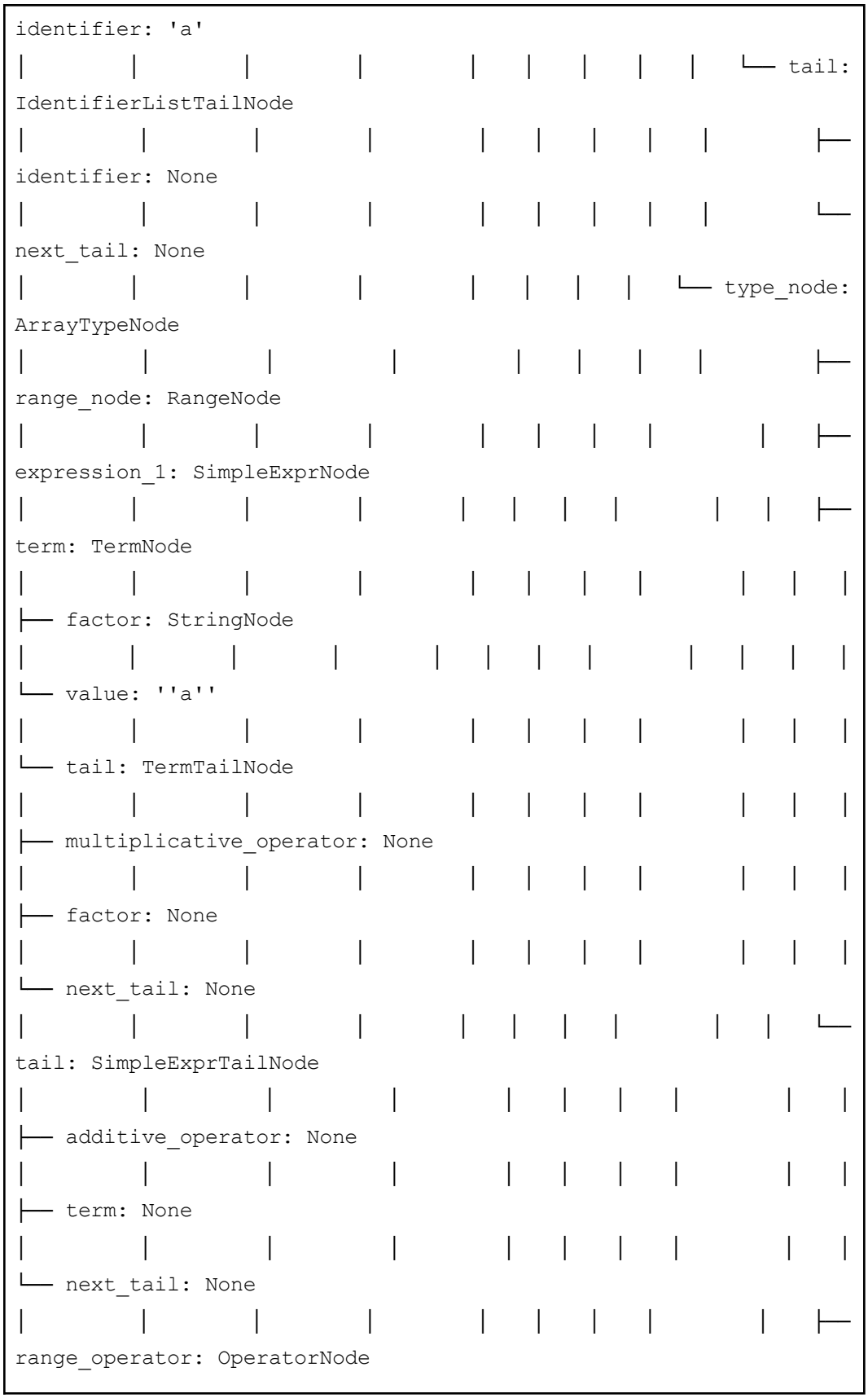
```
|   |   └── next_section: VarSectionNode
|   |       ├── var_declaration: None
|   |       └── next_section: None
|   └── subprogram_section: SubprogramSectionNode
|       ├── subprogram_declaration: ProcedureNode
|       |   ├── identifier: 'dummy'
|       |   ├── formal_parameter_list: ParameterListNode
|       |   |   ├── param_group_node: ParameterGroupNode
|       |   |   |   ├── modifier: ParameterModifierNode
|       |   |   |   |   └── keyword: 'variabel'
|       |   |   |   ├── identifier_list: IdentifierListNode
|       |   |   |   |   ├── identifier: 'i'
|       |   |   |   |   └── tail: IdentifierListTailNode
|       |   |   |   |       ├── identifier: None
|       |   |   |   |       └── next_tail: None
|       |   |   |   └── type_node: TypeNode
|       |   |   |       └── name: 'integer'
|       |   |   ├── expression_node: None
|       |   |   └── next_tail: ParameterTailNode
|       |   |       ├── param_group_node: ParameterGroupNode
|       |   |       |   ├── modifier: ParameterModifierNode
|       |   |       |   |   └── keyword: 'variabel'
|       |   |       |   ├── identifier_list: IdentifierListNode
|       |   |       |   |   ├── identifier: 'z'
|       |   |       |   |   └── tail: IdentifierListTailNode
|       |   |       |   |       ├── identifier: None
|       |   |       |   |       └── next_tail: None
|       |   |       |   └── type_node: TypeNode
|       |   |       |       └── name: 'complex'
|       |   |       ├── expression_node: None
|       |   |       └── next_tail: ParameterTailNode
|       |   |           ├── param_group_node: None
|       |   |           ├── expression_node: None
|       |   |           └── next_tail: None
|       |   └── block: BlockNode
|       |       ├── declaration_part: DeclarationPartNode
|       |       |   ├── const_section: ConstSectionNode
```

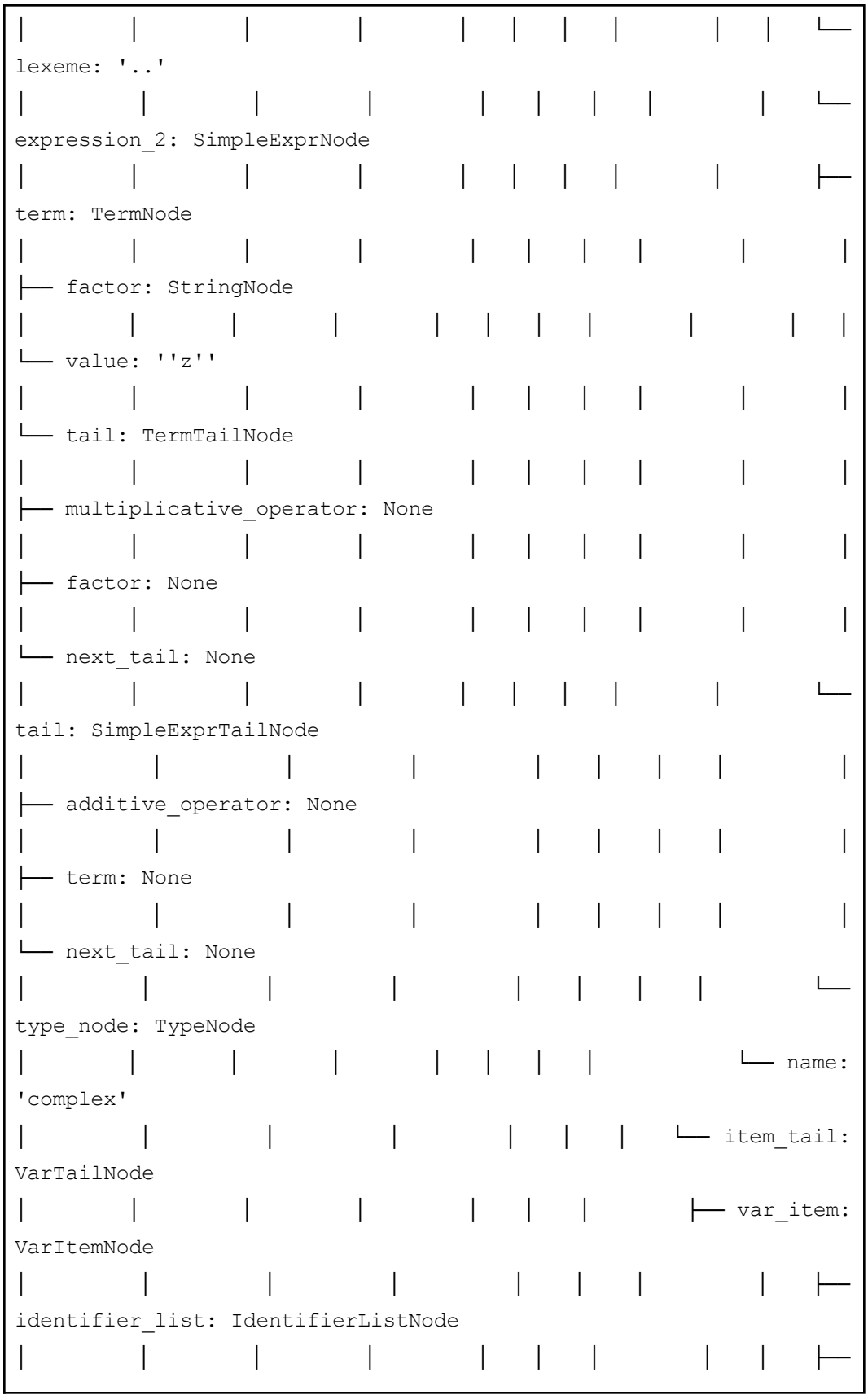

├── const_declaration: None
└── next_section: None
├── type_section: TypeSectionNode
├── type_declaration: None
└── next_section: None
├── var_section: VarSectionNode
├── var_declaration: VarDeclNode
├── var_item: VarItemNode
├── identifier_list: IdentifierListNode
├── identifier: 'u'
└── tail: IdentifierListTailNode
├── identifier: 'v'
└── next_tail: IdentifierListTailNode
├── identifier: None
└── next_tail: None
└── type_node: TypeNode
└── name: 'row'
└── item_tail: VarTailNode
├── var_item: VarItemNode
├── identifier_list: IdentifierListNode
├── identifier: 'h1'
└── tail: IdentifierListTailNode
├── identifier: 'h2'
└── next_tail: IdentifierListTailNode
├── identifier: None
└── next_tail: None
└── type_node: RecordTypeNode
└── field_list: FieldListNode

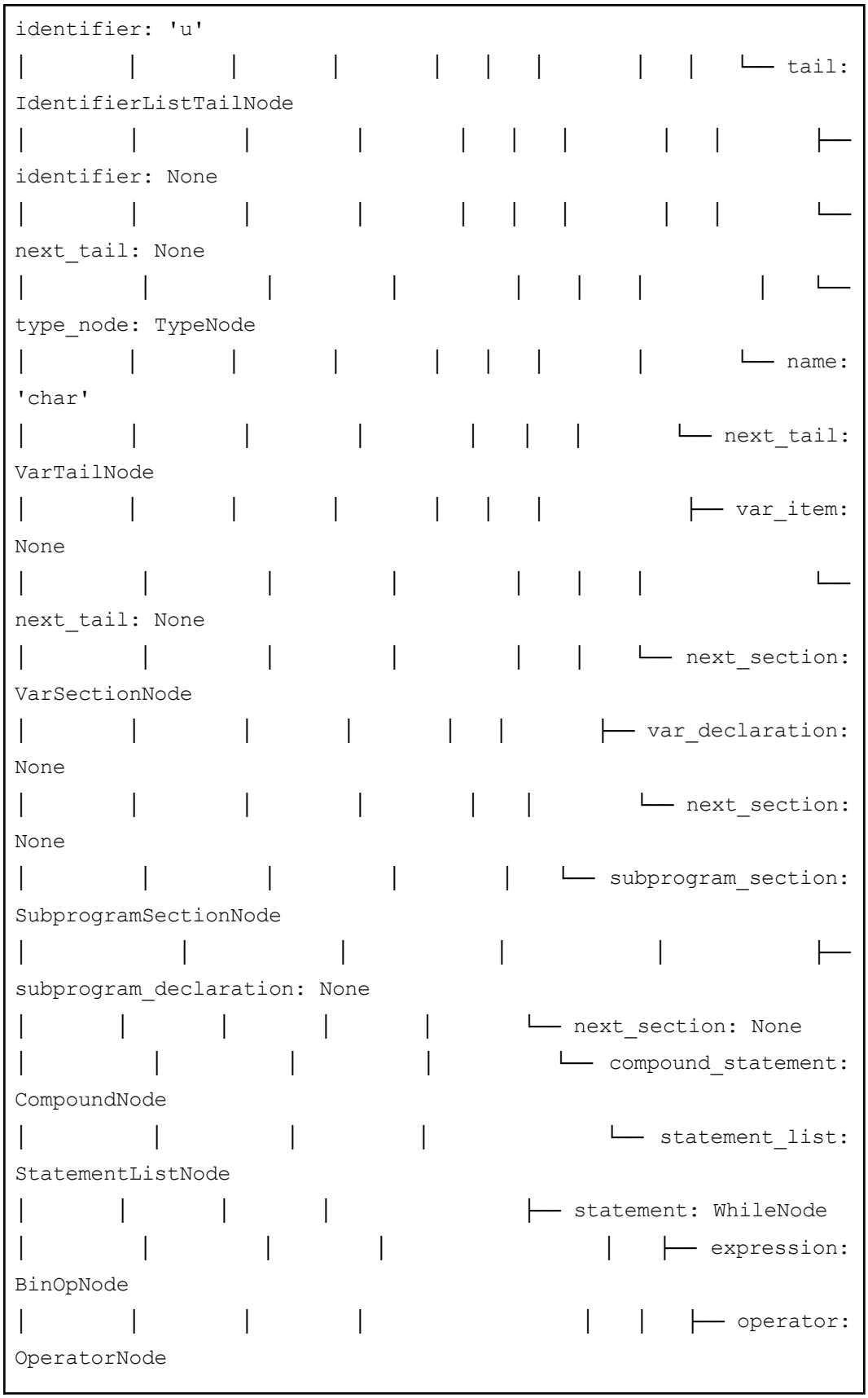

| | | | | | ├── identifier_list: IdentifierListNode
| | | | | | | ├── identifier: 'c'
| | | | | | | └── tail: IdentifierListTailNode
| | | | | | | | ├── identifier: None
| | | | | | | | └── next_tail: None
| | | | | | ├── type_definition: TypeNode
| | | | | | | └── name: 'complex'
| | | | | | └── field_list_tail: FieldListNode
| | | | | | ├── identifier_list: IdentifierListNode
| | | | | | | ├── identifier: 'r'
| | | | | | | └── tail: IdentifierListTailNode
| | | | | | | ├── identifier: None
| | | | | | | └── next_tail: None
| | | | | | ├── type_definition: TypeNode
| | | | | | | └── name: 'row'
| | | | | | └── field_list_tail: FieldListTailNode
| | | | | └── next_tail: VarTailNode
| | | | | ├── var_item: None
| | | | | └── next_tail: None
| | | | └── next_section: VarSectionNode
| | | | ├── var_declaration: None

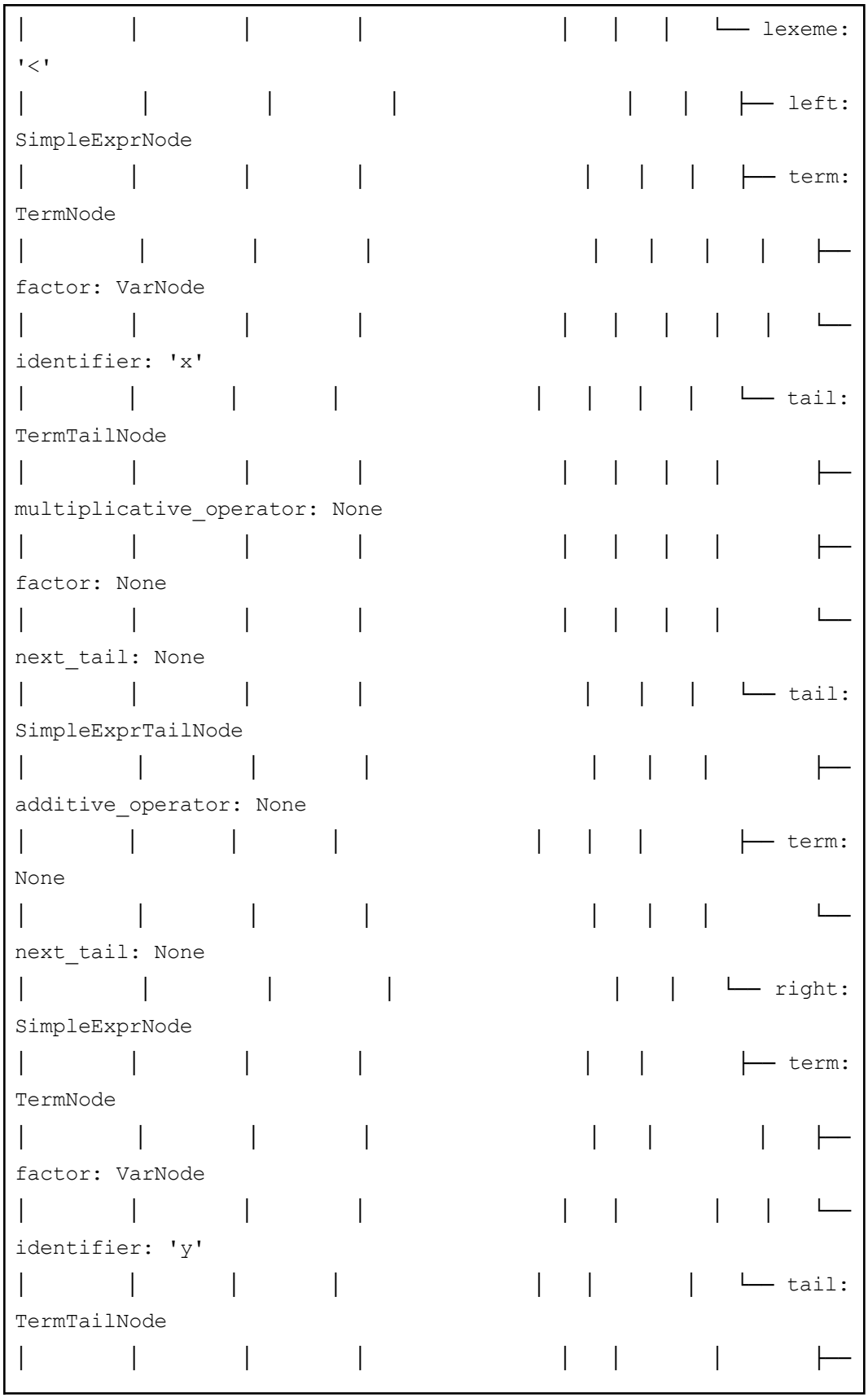
```
|       |       |   |       └── next_section: None
|       |       |   └── subprogram_section: SubprogramSectionNode
|       |       |       ├── subprogram_declaration: FunctionNode
|       |       |       |   ├── identifier: 'null'
|         |         |         |   ├── formal_parameter_list:
ParameterListNode
|         |         |         |   |   ├── param_group_node:
ParameterGroupNode
|           |         |         |   |   |   ├── modifier:
ParameterModifierNode
|       |       |       |   |   |   |   └── keyword: None
|         |         |         |   |   |   ├── identifier_list:
IdentifierListNode
|       |       |       |   |   |   |   ├── identifier: 'x'
|         |         |         |   |   |   |   └── tail:
IdentifierListTailNode
|       |       |       |   |   |   |       ├── identifier: 'y'
|         |         |         |   |   |   |       └── next_tail:
IdentifierListTailNode
|       |       |       |   |   |   |           ├── identifier:
None
|       |       |       |   |   |   |           └── next_tail:
None
|       |       |       |   |   |   └── type_node: TypeNode
|       |       |       |   |   |       └── name: 'real'
|       |       |       |   |   ├── expression_node: None
|       |       |       |   |   └── next_tail: ParameterTailNode
|         |         |         |   |       ├── param_group_node:
ParameterGroupNode
|         |         |         |   |       |   ├── modifier:
ParameterModifierNode
|       |       |       |   |       |   |   └── keyword: None
|         |         |         |   |       |   ├── identifier_list:
IdentifierListNode
|       |       |       |   |       |   |   ├── identifier: 'z'
|         |         |         |   |       |   |   └── tail:
IdentifierListTailNode
```

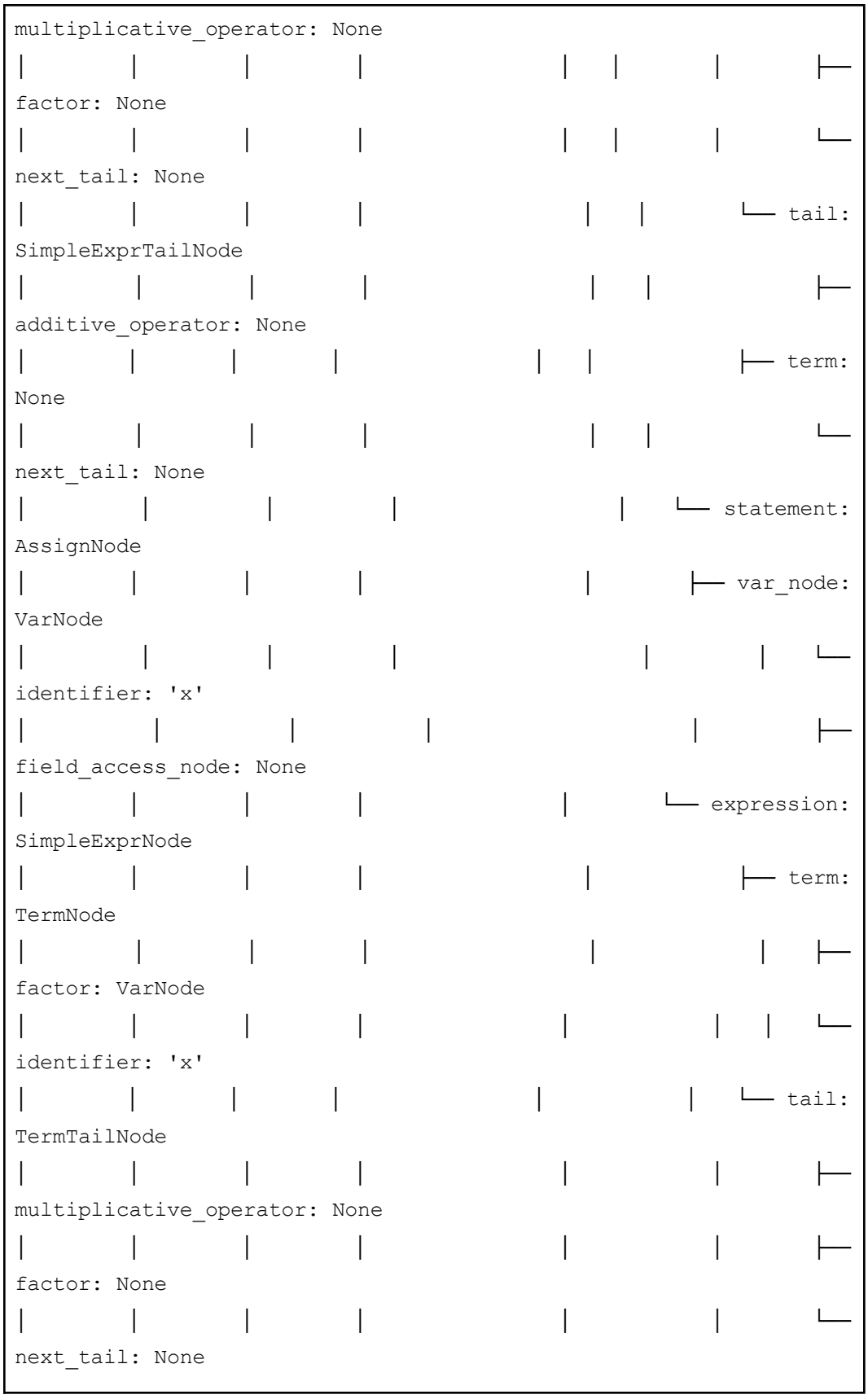

| | | | | | | ├── identifier: None
| | | | | | | └── next_tail: None
| | | | | | └── type_node: TypeNode
| | | | | | └── name: 'complex'
| | | | | ├── expression_node: None
| | | | | | └── next_tail: ParameterTailNode
| | | | | ├── param_group_node: None
| | | | | ├── expression_node: None
| | | | | └── next_tail: None
| | | | ├── return_type: TypeNode
| | | | | └── name: 'boolean'
| | | | └── block: BlockNode
| | | | ├── declaration_part: DeclarationPartNode
| | | | | ├── const_section: ConstSectionNode
| | | | | | ├── const_declaration: None
| | | | | | └── next_section: None
| | | | | ├── type_section: TypeSectionNode
| | | | | | ├── type_declaration: None
| | | | | | └── next_section: None
| | | | | ├── var_section: VarSectionNode
| | | | | | ├── var_declaration: VarDeclNode
| | | | | | | ├── var_item: VarItemNode
| | | | | | | | | ├── identifier_list: IdentifierListNode
| | | | | | | | | ├──

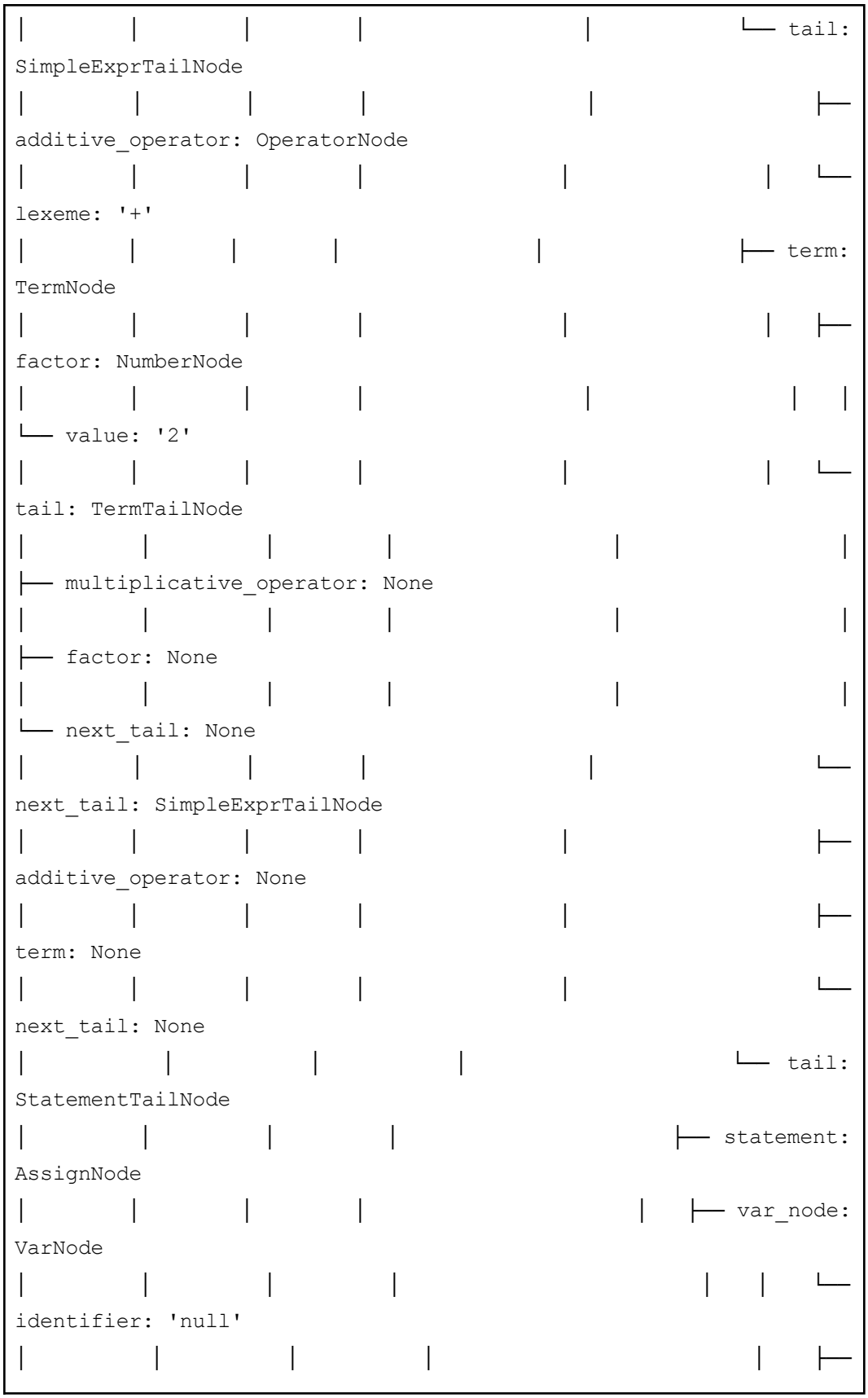

identifier: 'a'
└── tail: IdentifierListTailNode
├── identifier: None
└── next_tail: None
└── type_node: ArrayTypeNode
├── range_node: RangeNode
├── expression_1: SimpleExprNode
├── term: TermNode
├── factor: StringNode
└── value: ''a''
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
├── range_operator: OperatorNode

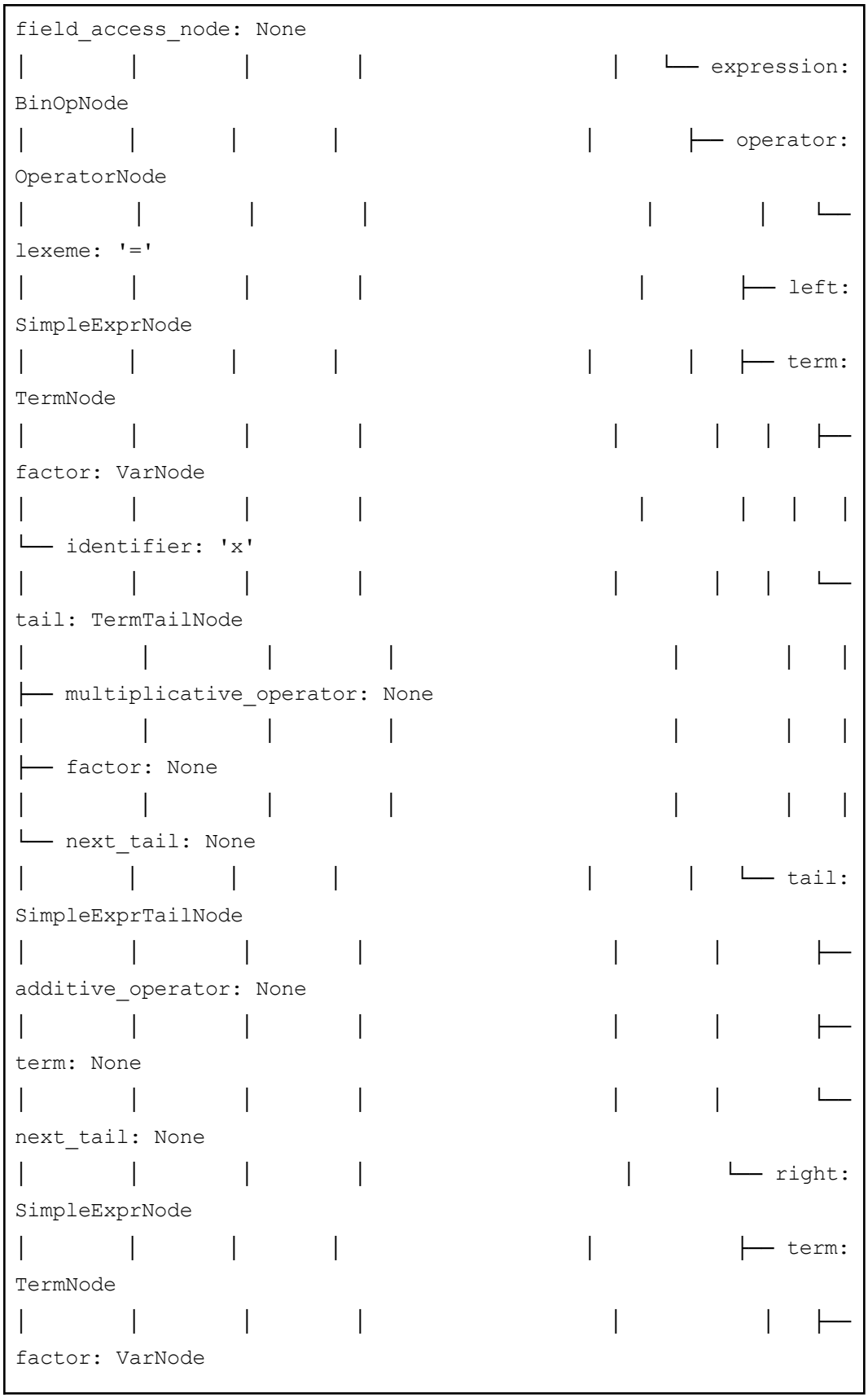

lexeme: '..'
expression_2: SimpleExprNode
term: TermNode
factor: StringNode
value: ''z''
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
type_node: TypeNode
name: 'complex'
item_tail: VarTailNode
var_item: VarItemNode
identifier_list: IdentifierListNode

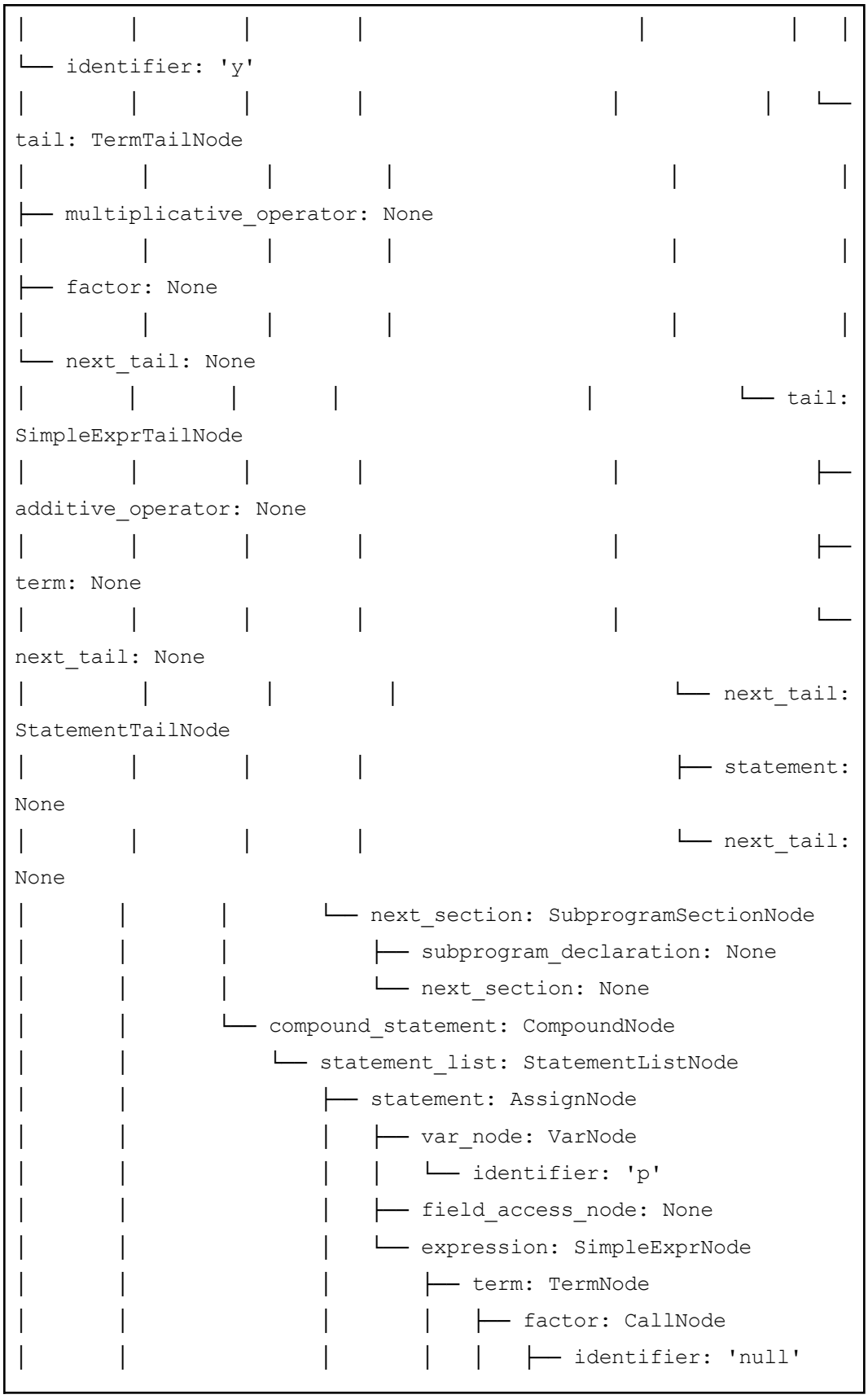

identifier: 'u'
└── tail: IdentifierListTailNode
├── identifier: None
└── next_tail: None
└── type_node: TypeNode
└── name: 'char'
└── next_tail: VarTailNode
├── var_item: None
└── next_tail: None
└── next_section: VarSectionNode
├── var_declaration: None
└── next_section: None
└── subprogram_section: SubprogramSectionNode
├── subprogram_declaration: None
└── next_section: None
└── compound_statement: CompoundNode
└── statement_list: StatementListNode
├── statement: WhileNode
├── expression: BinOpNode
└── operator: OperatorNode


└── lexeme: '<'
├── left: SimpleExprNode
├── term: TermNode
├── factor: VarNode
└── identifier: 'x'
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
└── right: SimpleExprNode
├── term: TermNode
├── factor: VarNode
└── identifier: 'y'
└── tail: TermTailNode
├──

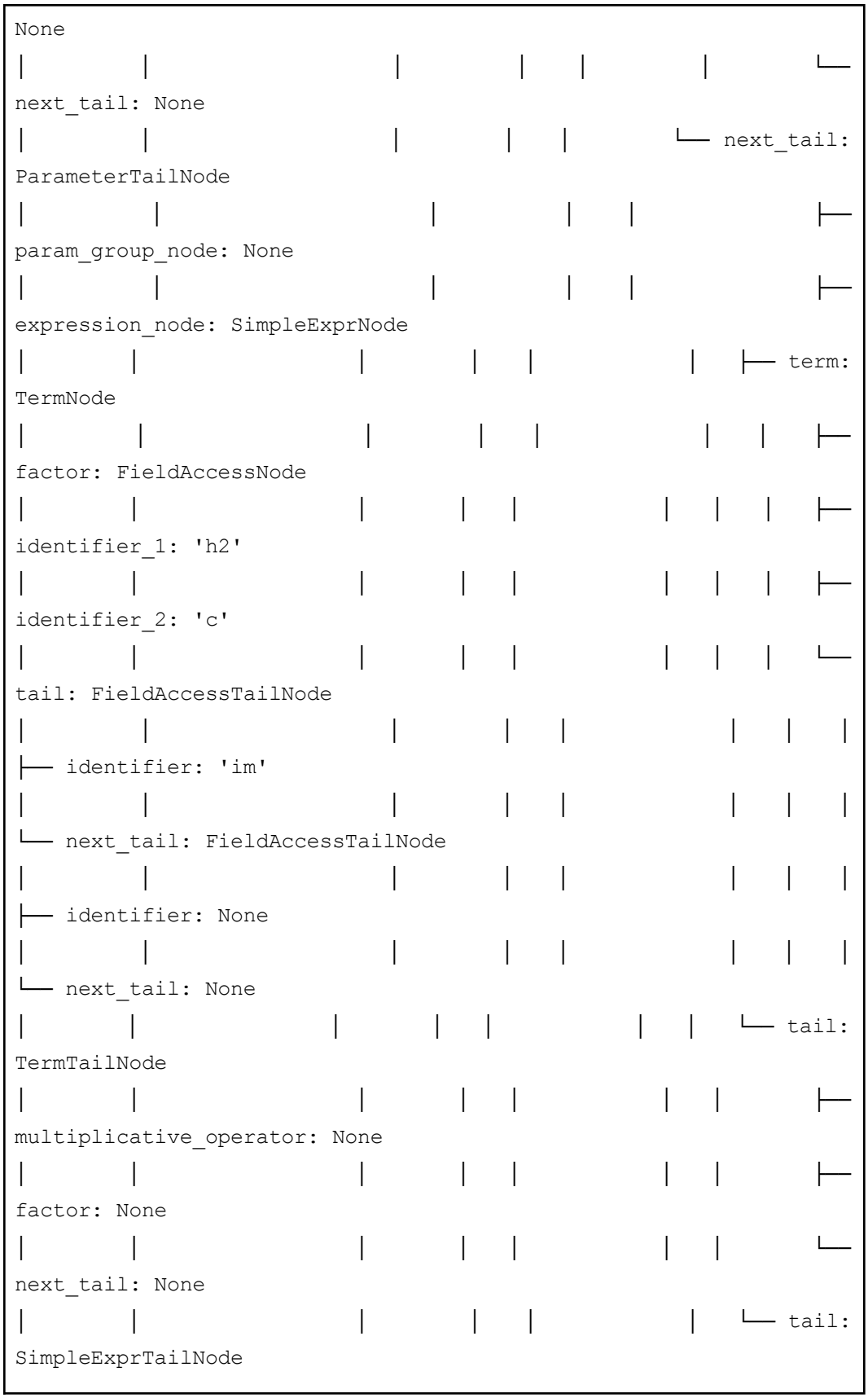

multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
statement: AssignNode
var_node: VarNode
identifier: 'x'
field_access_node: None
expression: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'x'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None

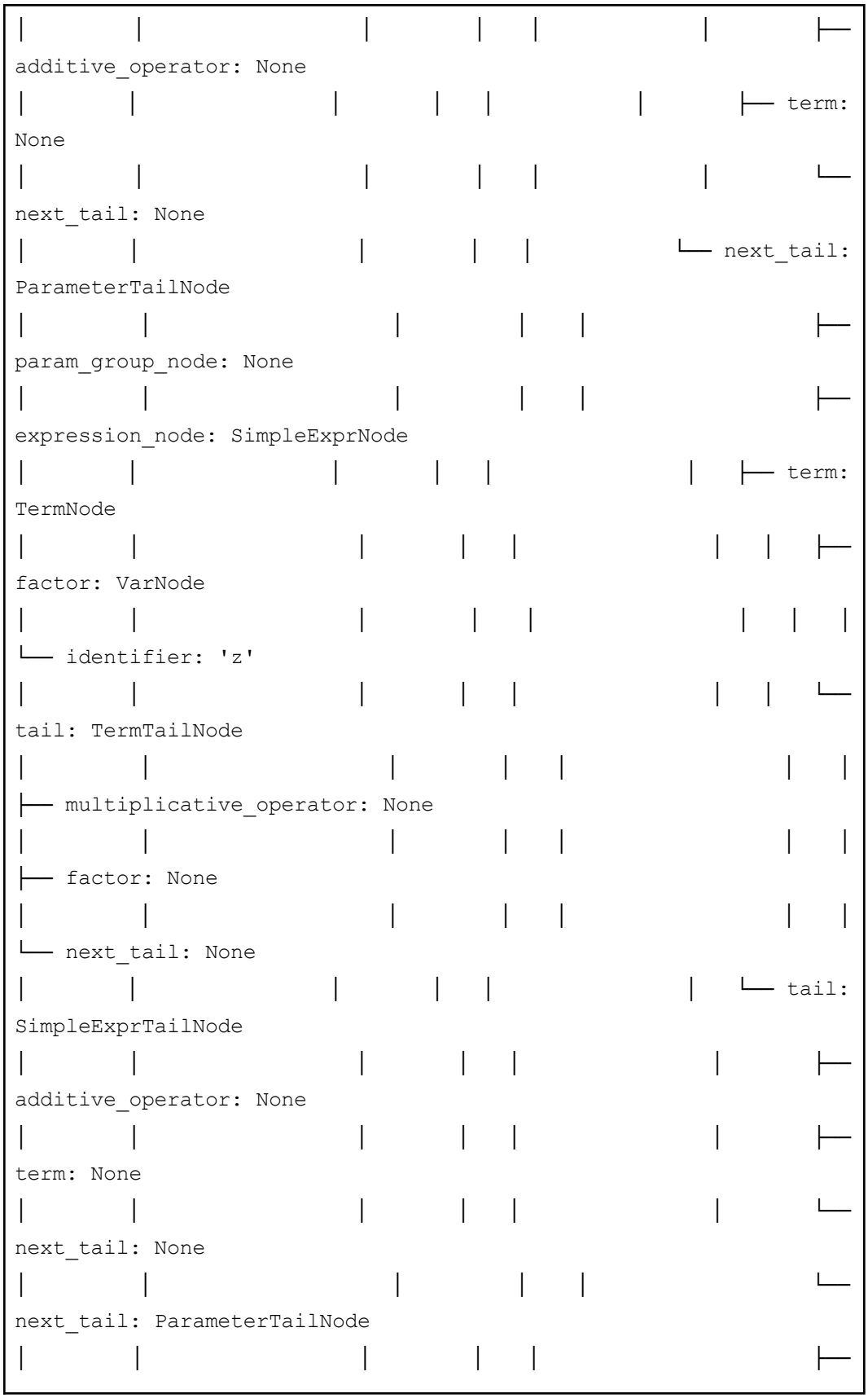

└── tail: SimpleExprTailNode
├── additive_operator: OperatorNode
└── lexeme: '+'
├── term: TermNode
├── factor: NumberNode
└── value: '2'
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── next_tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
└── tail: StatementTailNode
├── statement: AssignNode
├── var_node: VarNode
└── identifier: 'null'
├──

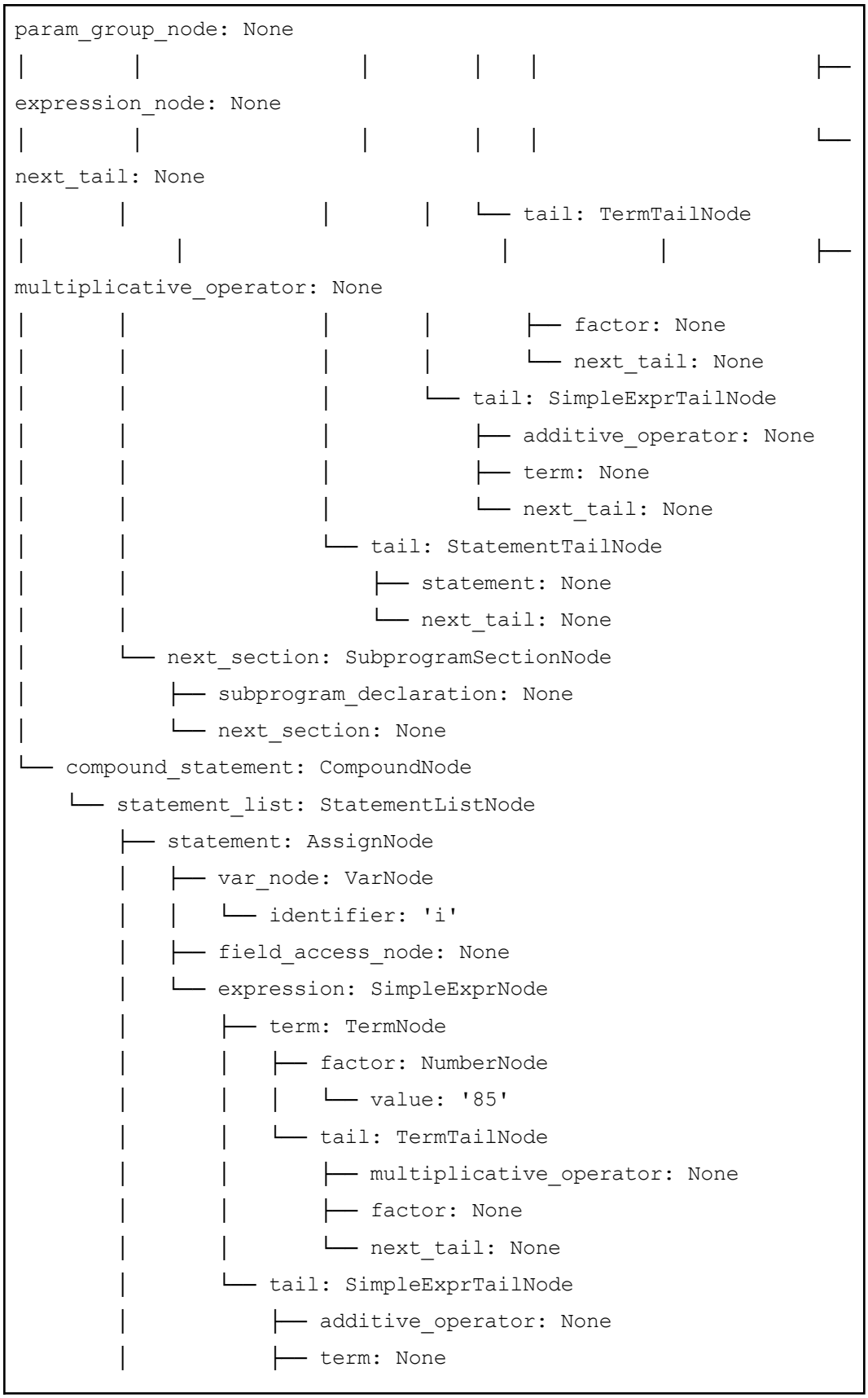

field_access_node: None
└── expression: BinOpNode
├── operator: OperatorNode
│ └── lexeme: '='
├── left: SimpleExprNode
│ ├── term: TermNode
│ │ ├── factor: VarNode
│ │ │ └── identifier: 'x'
│ │ └── tail: TermTailNode
│ │ ├── multiplicative_operator: None
│ │ ├── factor: None
│ │ └── next_tail: None
│ └── tail: SimpleExprTailNode
│ ├── additive_operator: None
│ ├── term: None
│ └── next_tail: None
└── right: SimpleExprNode
├── term: TermNode
│ ├── factor: VarNode

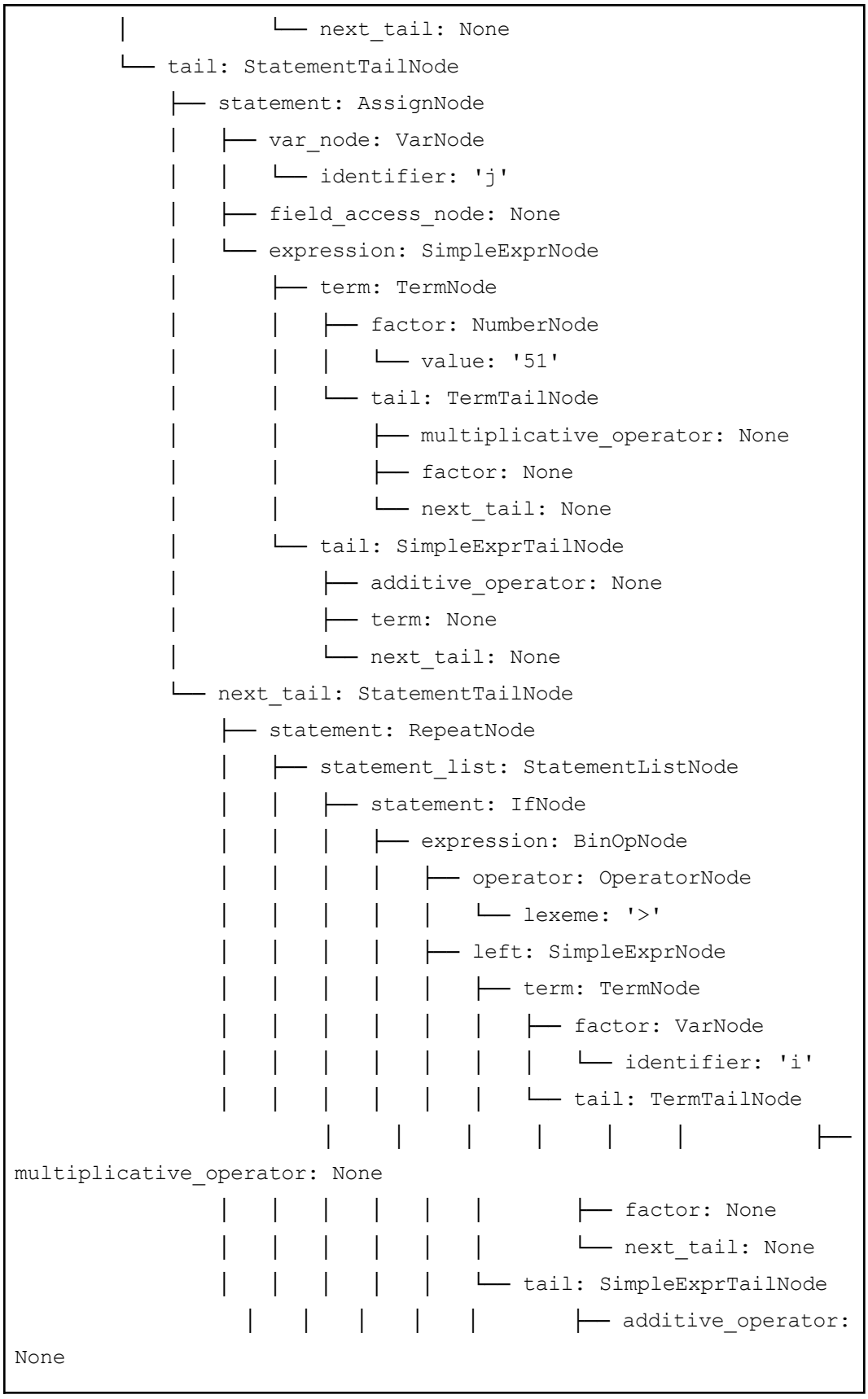

└── identifier: 'y'
tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
└── next_tail: StatementTailNode
├── statement: None
└── next_tail: None
└── next_section: SubprogramSectionNode
├── subprogram_declaration: None
└── next_section: None
└── compound_statement: CompoundNode
└── statement_list: StatementListNode
├── statement: AssignNode
├── var_node: VarNode
└── identifier: 'p'
├── field_access_node: None
└── expression: SimpleExprNode
├── term: TermNode
├── factor: CallNode
├── identifier: 'null'

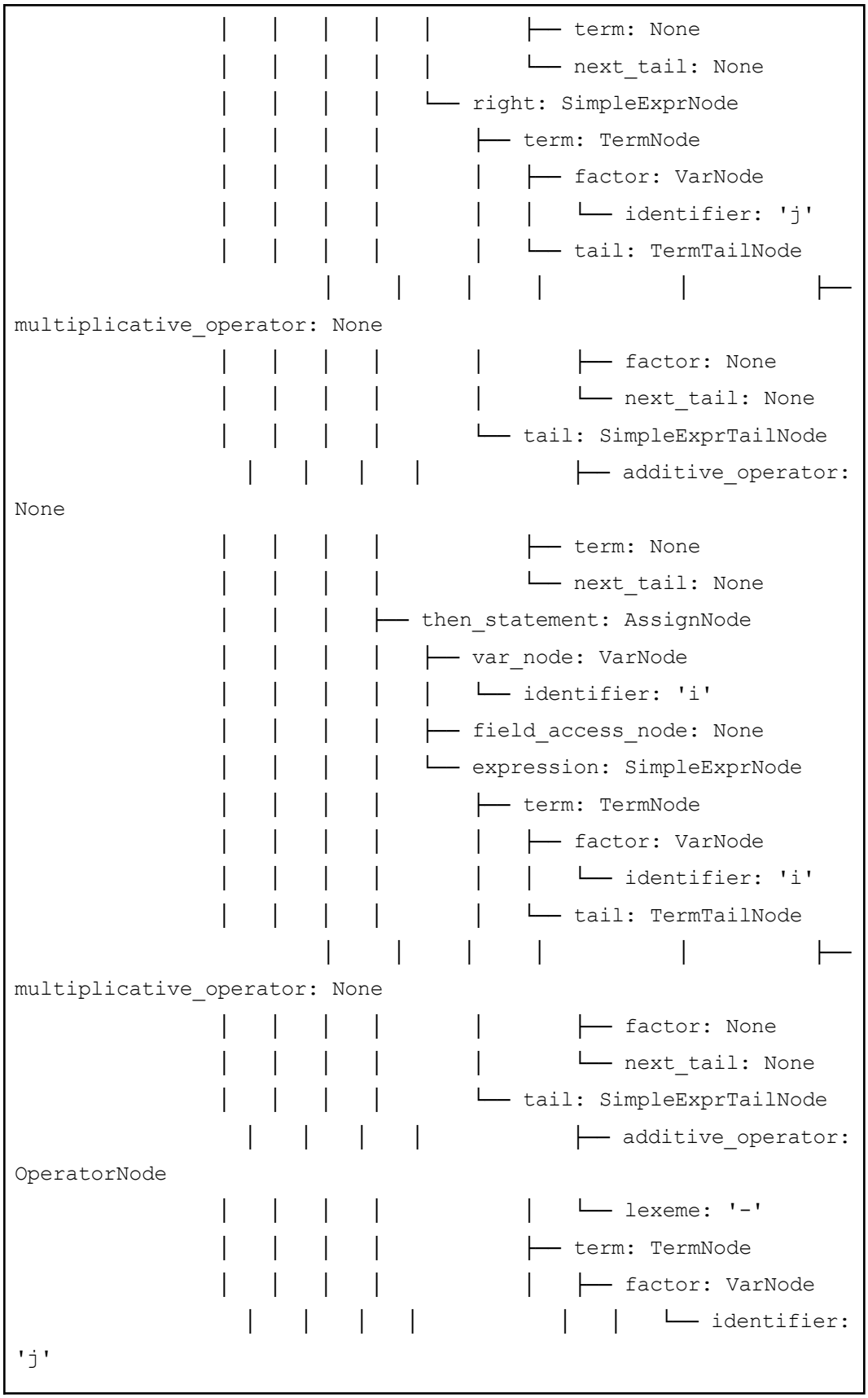
```
|       |                |       |   |   └── parameter_list:
ParameterListNode
|       |             |      |   |      ├── param_group_node:
None
|       |             |      |   |      ├── expression_node:
SimpleExprNode
|       |                |       |   |       |   ├── term:
TermNode
|       |             |      |   |      |   |   ├── factor:
FieldAccessNode
|       |             |       |   |       |   |   |   ├──
identifier_1: 'h1'
|       |             |       |   |       |   |   |   ├──
identifier_2: 'c'
|       |             |      |   |      |   |   |   └── tail:
FieldAccessTailNode
|       |             |      |   |      |   |   |       ├──
identifier: 're'
|       |             |      |   |      |   |   |       └──
next_tail: FieldAccessTailNode
|         |               |       |   |       |   |   |
├── identifier: None
|         |               |       |   |       |   |   |
└── next_tail: None
|       |             |       |   |       |   |   └── tail:
TermTailNode
|       |             |       |   |       |   |       ├──
multiplicative_operator: None
|       |             |       |   |       |   |       ├──
factor: None
|       |             |       |   |       |   |       └──
next_tail: None
|       |               |       |   |       |   └── tail:
SimpleExprTailNode
|       |               |       |   |       |       ├──
additive_operator: None
|       |             |       |   |       |       ├── term:
```

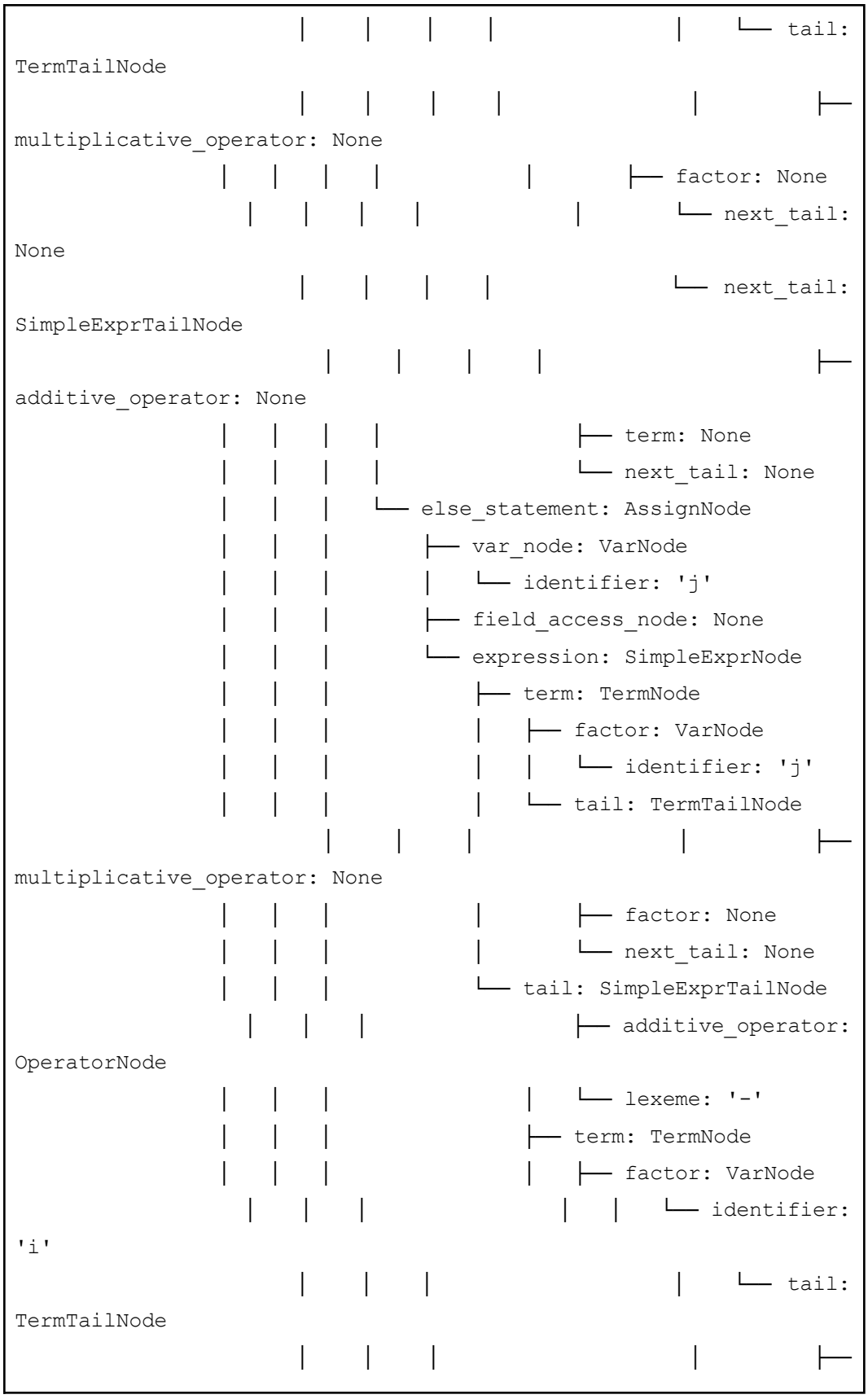

None
next_tail: None
next_tail: ParameterTailNode
param_group_node: None
expression_node: SimpleExprNode
term: TermNode
factor: FieldAccessNode
identifier_1: 'h2'
identifier_2: 'c'
tail: FieldAccessTailNode
identifier: 'im'
next_tail: FieldAccessTailNode
identifier: None
next_tail: None
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode

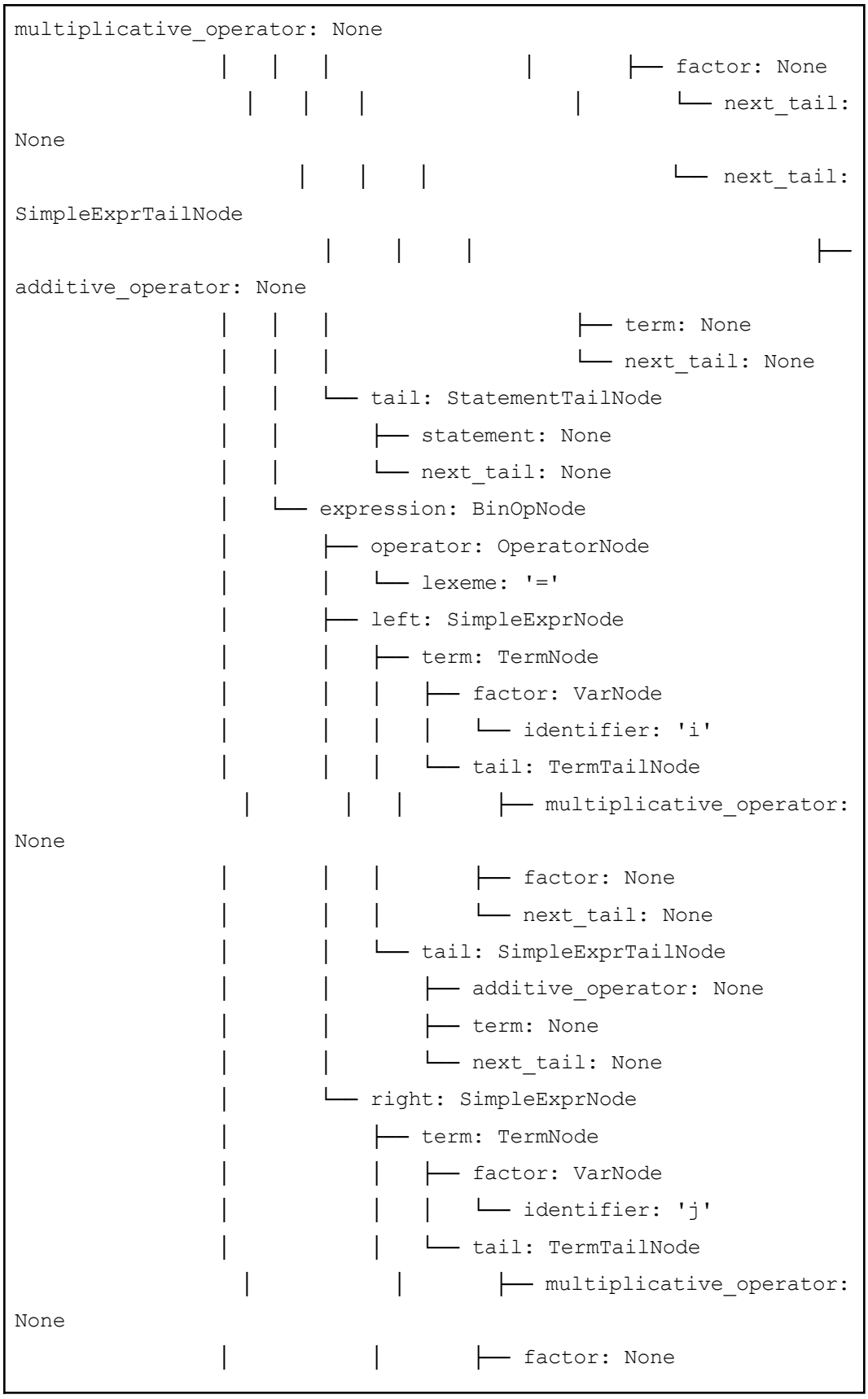

additive_operator: None
term: None
next_tail: None
next_tail: ParameterTailNode
param_group_node: None
expression_node: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'z'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
next_tail: ParameterTailNode

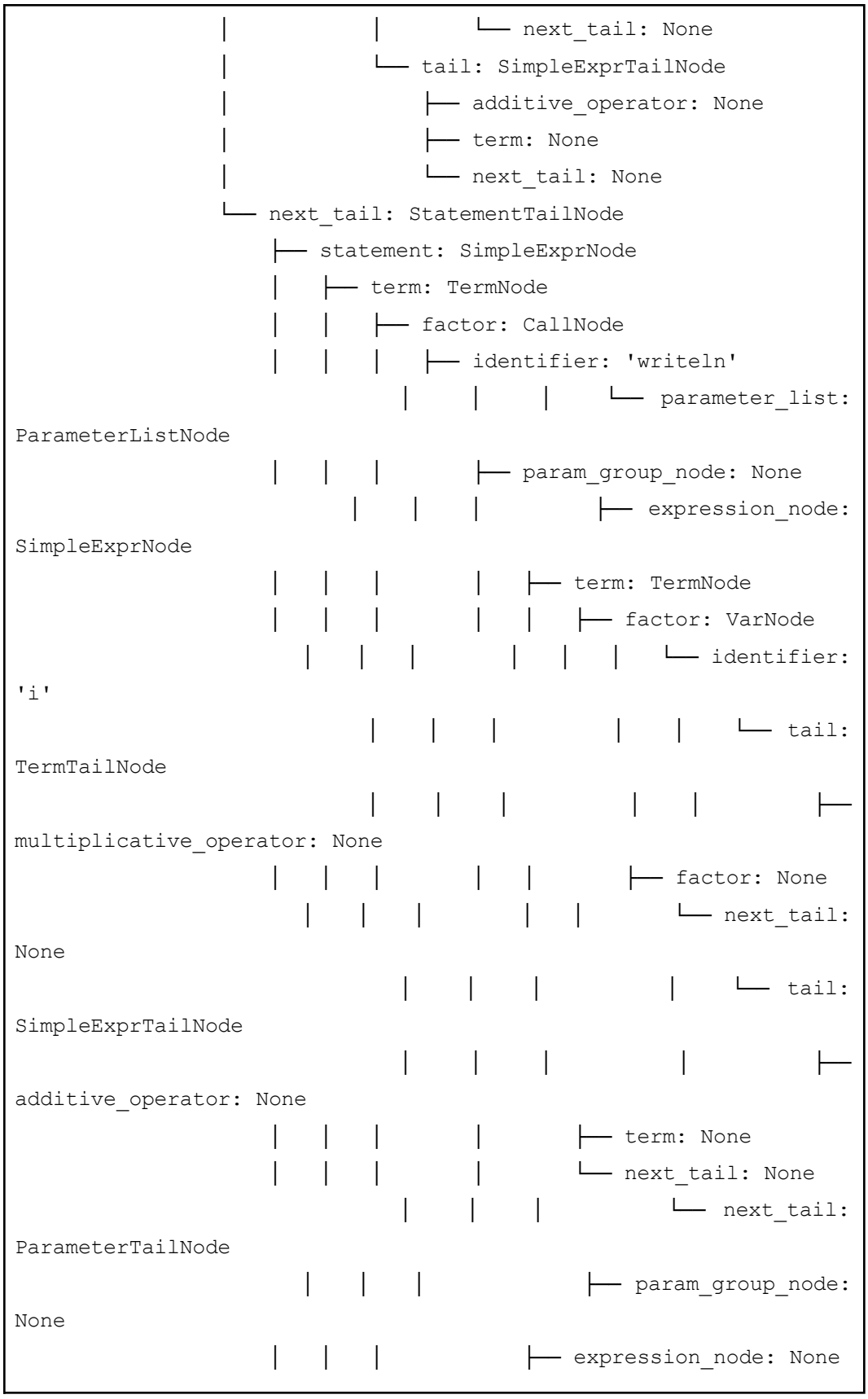

param_group_node: None
expression_node: None
next_tail: None
└── tail: TermTailNode
multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
└── tail: StatementTailNode
├── statement: None
└── next_tail: None
└── next_section: SubprogramSectionNode
├── subprogram_declaration: None
└── next_section: None
└── compound_statement: CompoundNode
└── statement_list: StatementListNode
├── statement: AssignNode
├── var_node: VarNode
└── identifier: 'i'
├── field_access_node: None
└── expression: SimpleExprNode
├── term: TermNode
├── factor: NumberNode
└── value: '85'
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: None
├── term: None


└── next_tail: None
└── tail: StatementTailNode
├── statement: AssignNode
│ ├── var_node: VarNode
│ │ └── identifier: 'j'
│ ├── field_access_node: None
│ └── expression: SimpleExprNode
│ ├── term: TermNode
│ │ ├── factor: NumberNode
│ │ │ └── value: '51'
│ │ └── tail: TermTailNode
│ │ ├── multiplicative_operator: None
│ │ ├── factor: None
│ │ └── next_tail: None
│ └── tail: SimpleExprTailNode
│ ├── additive_operator: None
│ ├── term: None
│ └── next_tail: None
└── next_tail: StatementTailNode
├── statement: RepeatNode
│ ├── statement_list: StatementListNode
│ │ ├── statement: IfNode
│ │ │ ├── expression: BinOpNode
│ │ │ │ ├── operator: OperatorNode
│ │ │ │ │ └── lexeme: '>'
│ │ │ │ ├── left: SimpleExprNode
│ │ │ │ │ ├── term: TermNode
│ │ │ │ │ │ ├── factor: VarNode
│ │ │ │ │ │ │ └── identifier: 'i'
│ │ │ │ │ │ └── tail: TermTailNode
│ │ │ │ │ │ ├── multiplicative_operator: None
│ │ │ │ │ │ ├── factor: None
│ │ │ │ │ │ └── next_tail: None
│ │ │ │ │ └── tail: SimpleExprTailNode
│ │ │ │ │ ├── additive_operator: None


term: None
next_tail: None
right: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'j'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
then_statement: AssignNode
var_node: VarNode
identifier: 'i'
field_access_node: None
expression: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'i'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: OperatorNode
lexeme: '-'
term: TermNode
factor: VarNode
identifier: 'j'


| | | | | └── tail: TermTailNode
| | | | | ├── multiplicative_operator: None
| | | | | ├── factor: None
| | | | | └── next_tail: None
| | | | └── next_tail: SimpleExprTailNode
| | | | ├── additive_operator: None
| | | | ├── term: None
| | | | └── next_tail: None
| | | └── else_statement: AssignNode
| | | ├── var_node: VarNode
| | | | └── identifier: 'j'
| | | ├── field_access_node: None
| | | └── expression: SimpleExprNode
| | | ├── term: TermNode
| | | | ├── factor: VarNode
| | | | | └── identifier: 'j'
| | | | └── tail: TermTailNode
| | | | ├── multiplicative_operator: None
| | | | ├── factor: None
| | | | └── next_tail: None
| | | └── tail: SimpleExprTailNode
| | | ├── additive_operator: OperatorNode
| | | | └── lexeme: '-'
| | | ├── term: TermNode
| | | | ├── factor: VarNode
| | | | | └── identifier: 'i'
| | | | └── tail: TermTailNode
| | | | ├──


multiplicative_operator: None
factor: None
next_tail:
None
next_tail:
SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
tail: StatementTailNode
statement: None
next_tail: None
expression: BinOpNode
operator: OperatorNode
lexeme: '='
left: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'i'
tail: TermTailNode
multiplicative_operator:
None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
right: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'j'
tail: TermTailNode
multiplicative_operator:
None
factor: None


└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
└── next_tail: StatementTailNode
├── statement: SimpleExprNode
│ ├── term: TermNode
│ │ ├── factor: CallNode
│ │ │ ├── identifier: 'writeln'
│ │ │ └── parameter_list: ParameterListNode
│ │ │ ├── param_group_node: None
│ │ │ ├── expression_node: SimpleExprNode
│ │ │ ├── term: TermNode
│ │ │ │ ├── factor: VarNode
│ │ │ │ │ └── identifier: 'i'
│ │ │ │ └── tail: TermTailNode
│ │ │ │ ├── multiplicative_operator: None
│ │ │ │ ├── factor: None
│ │ │ │ └── next_tail: None
│ │ │ └── tail: SimpleExprTailNode
│ │ │ ├── additive_operator: None
│ │ │ ├── term: None
│ │ │ └── next_tail: None
│ │ └── next_tail: ParameterTailNode
│ │ ├── param_group_node: None
│ │ └── expression_node: None

```
                    |   |   |           └── next_tail: None
                    |   |   └── tail: TermTailNode
                    |   |       ├── multiplicative_operator: None
                    |   |       ├── factor: None
                    |   |       └── next_tail: None
                    |   └── tail: SimpleExprTailNode
                    |       ├── additive_operator: None
                    |       ├── term: None
                    |       └── next_tail: None
                    └── next_tail: StatementTailNode
                        ├── statement: None
                        └── next_tail: None

================================================
Tahap: Semantic Analysis (Symbol Tables)
================================================

--- SYMBOL TABLE (tab) ---
Idx  | Identifier   | Link | Obj | Typ | Ref | Nrm | Lev | Adr |
-----------------------------------------------------------------
--------------------
1    | array        | 0    | 0   | 0   | 0   | 0   | 0   | 0   |
2    | begin        | 0    | 0   | 0   | 0   | 0   | 0   | 0   |
3    | case         | 0    | 0   | 0   | 0   | 0   | 0   | 0   |
4    | const        | 0    | 0   | 0   | 0   | 0   | 0   | 0   |
5    | div          | 0    | 0   | 0   | 0   | 0   | 0   | 0   |
6    | do           | 0    | 0   | 0   | 0   | 0   | 0   | 0   |
7    | downto       | 0    | 0   | 0   | 0   | 0   | 0   | 0   |
8    | else         | 0    | 0   | 0   | 0   | 0   | 0   | 0   |
9    | end          | 0    | 0   | 0   | 0   | 0   | 0   | 0   |
10   | for          | 0    | 0   | 0   | 0   | 0   | 0   | 0   |
11   | function     | 0    | 0   | 0   | 0   | 0   | 0   | 0   |
12   | if           | 0    | 0   | 0   | 0   | 0   | 0   | 0   |
13   | mod          | 0    | 0   | 0   | 0   | 0   | 0   | 0   |
14   | not          | 0    | 0   | 0   | 0   | 0   | 0   | 0   |
15   | of           | 0    | 0   | 0   | 0   | 0   | 0   | 0   |
16   | or           | 0    | 0   | 0   | 0   | 0   | 0   | 0   |
```

17	packed	0	0	0	0	0	0	0
18	procedure	0	0	0	0	0	0	0
19	program	0	0	0	0	0	0	0
20	record	0	0	0	0	0	0	0
21	repeat	0	0	0	0	0	0	0
22	string	0	2	0	0	0	0	0
23	then	0	0	0	0	0	0	0
24	to	0	0	0	0	0	0	0
25	type	0	0	0	0	0	0	0
26	until	0	0	0	0	0	0	0
27	var	0	0	0	0	0	0	0
28	while	0	0	0	0	0	0	0
29	test0	28	5	0	0	1	1	0
30	ten	29	0	1	0	0	1	10
31	plus	30	0	4	0	0	1	43
32	row	31	2	5	1	1	1	10
33	complex	32	2	6	2	1	1	2
34	re	0	1	2	0	1	2	0
35	im	34	1	2	0	1	2	1
36	i	33	1	1	0	1	1	0
37	j	36	1	1	0	1	1	1
38	p	37	1	3	0	1	1	2
39	z	38	1	6	2	1	1	3
40	matrix	39	1	5	2	1	1	5
41	pattern	40	1	5	4	1	1	75
42	dummy	41	3	0	0	1	1	0
43	i	41	6	1	0	0	2	0
44	z	43	6	33	0	0	2	1
45	u	44	1	5	1	1	2	0
46	v	45	1	5	1	1	2	10
47	c	0	1	6	2	1	3	0
48	r	47	1	5	1	1	3	0
49	h1	46	1	6	4	1	2	20
50	h2	49	1	6	4	1	2	32
51	null	50	4	3	0	1	2	0
52	x	50	6	2	0	0	3	0
53	y	52	6	2	0	0	3	1

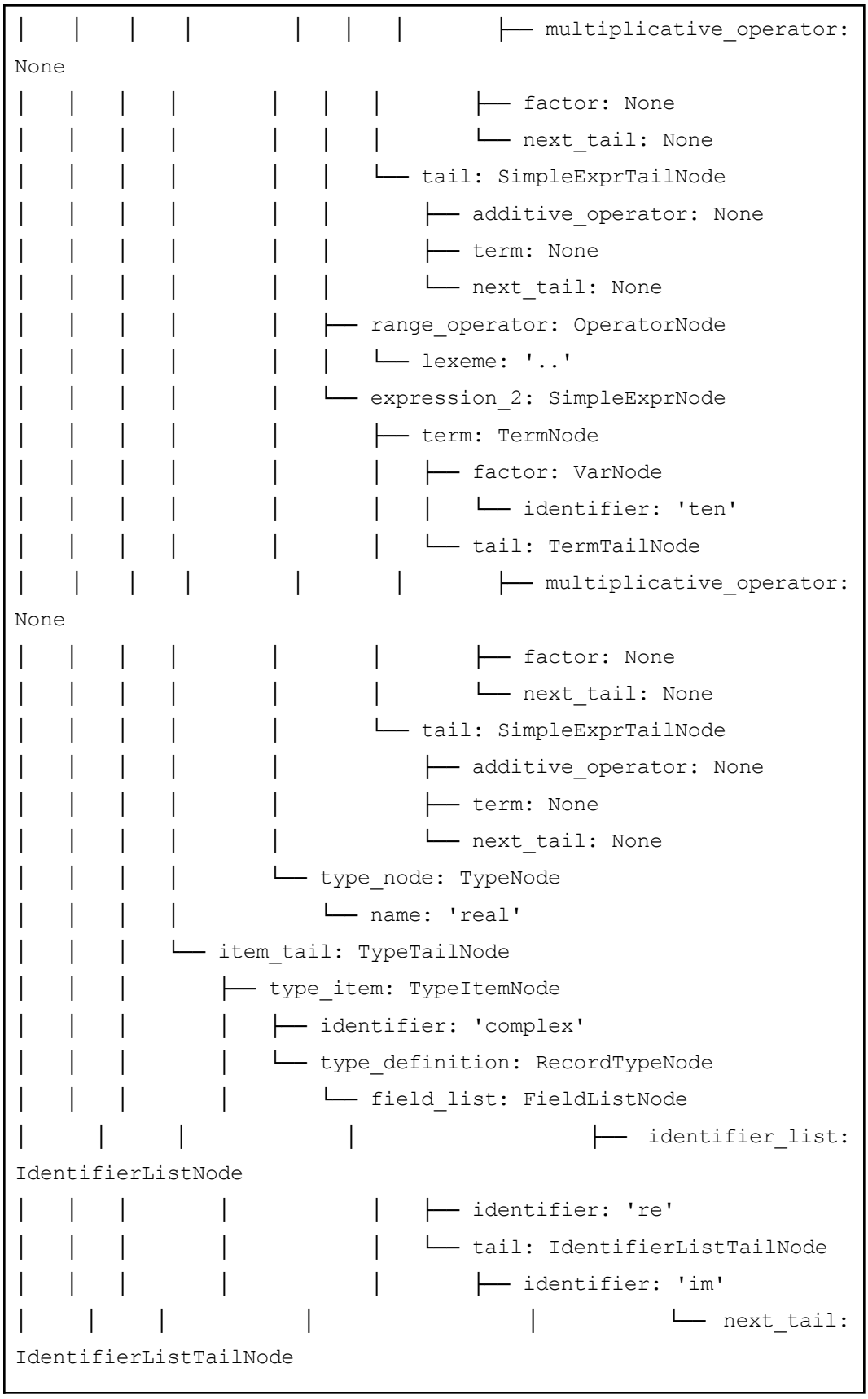
```
54   | z            | 53   | 6   | 33  | 0   | 0   | 3   | 2   |
55   | a            | 54   | 1   | 5   | 5   | 1   | 3   | 0   |
56   | u            | 55   | 1   | 4   | 0   | 1   | 3   | 52  |

--- BLOCK TABLE (btab) ---
Idx  | Last | Lpar | P.Sz | V.Sz |
---------------------------------------
0    | 28   | 0    | 0    | 0    |
1    | 42   | 0    | 0    | 95   |
2    | 35   | 0    | 0    | 2    |
3    | 51   | 44   | 2    | 44   |
4    | 48   | 0    | 0    | 12   |
5    | 56   | 54   | 3    | 53   |

--- ARRAY TABLE (atab) ---
Idx  | Xtyp | Etyp | Eref | Low  | High | ElSz | Size |
-----------------------------------------------------------------
-----
1    | 1    | 2    | 0    | 1    | 10   | 1    | 10   |
2    | 1    | 5    | 1    | -3   | 3    | 10   | 70   |
3    | 1    | 4    | 0    | 2    | 5    | 1    | 4    |
4    | 1    | 5    | 3    | 1    | 5    | 4    | 20   |
5    | 1    | 6    | 2    | 97   | 122  | 2    | 52   |
```

Pembahasan: Berdasarkan hasil uji di atas, Semantic Analyzer berhasil memproses kode program `test1_complex.pas` yang memuat berbagai struktur kompleks, mulai dari deklarasi konstanta, tipe bentukan, hingga prosedur dan fungsi bersarang. Output Abstract Syntax Tree (AST) menunjukkan bahwa parser telah mentransformasi kode sumber menjadi struktur pohon hierarkis yang benar, merepresentasikan logika program secara akurat. Selanjutnya, keberhasilan pembentukan Symbol Table (TAB) dengan 56 entri membuktikan bahwa semua identifier, baik keyword bawaan maupun variabel pengguna, telah tercatat dengan atribut yang tepat.

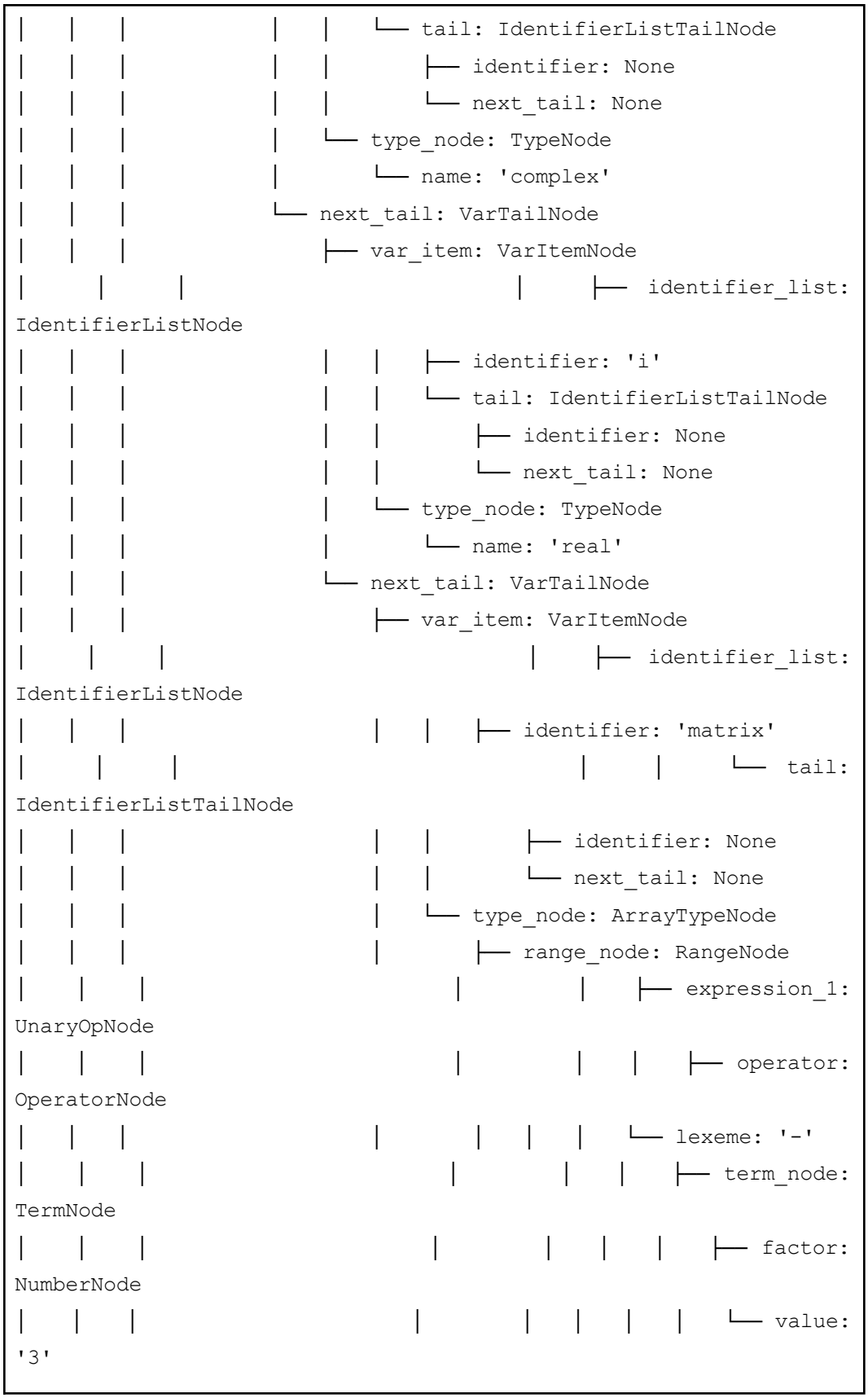
3.2.2 Hasil Uji 2

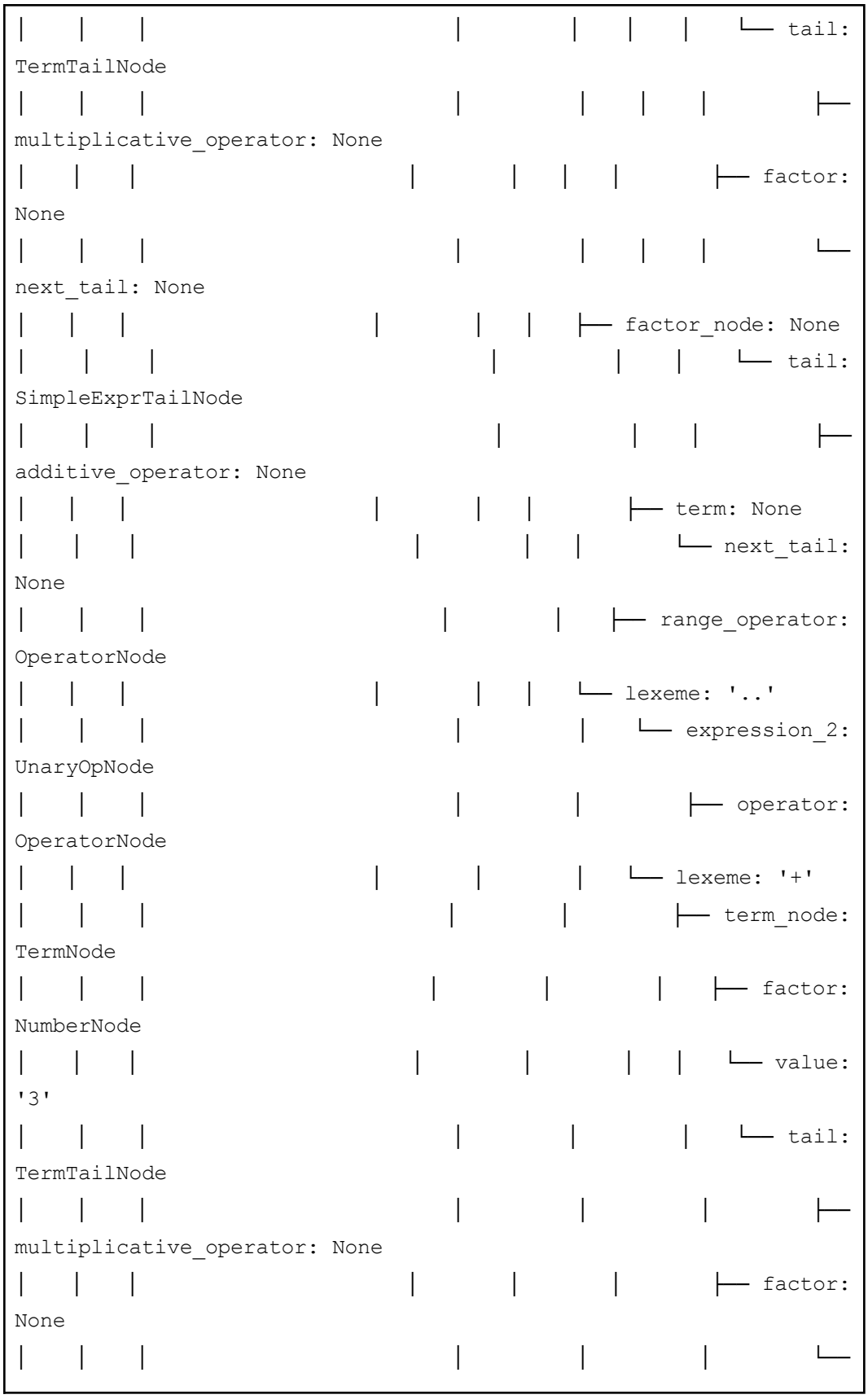
```
================================================
Tahap: AST Generation (Abstract Syntax Tree)
```

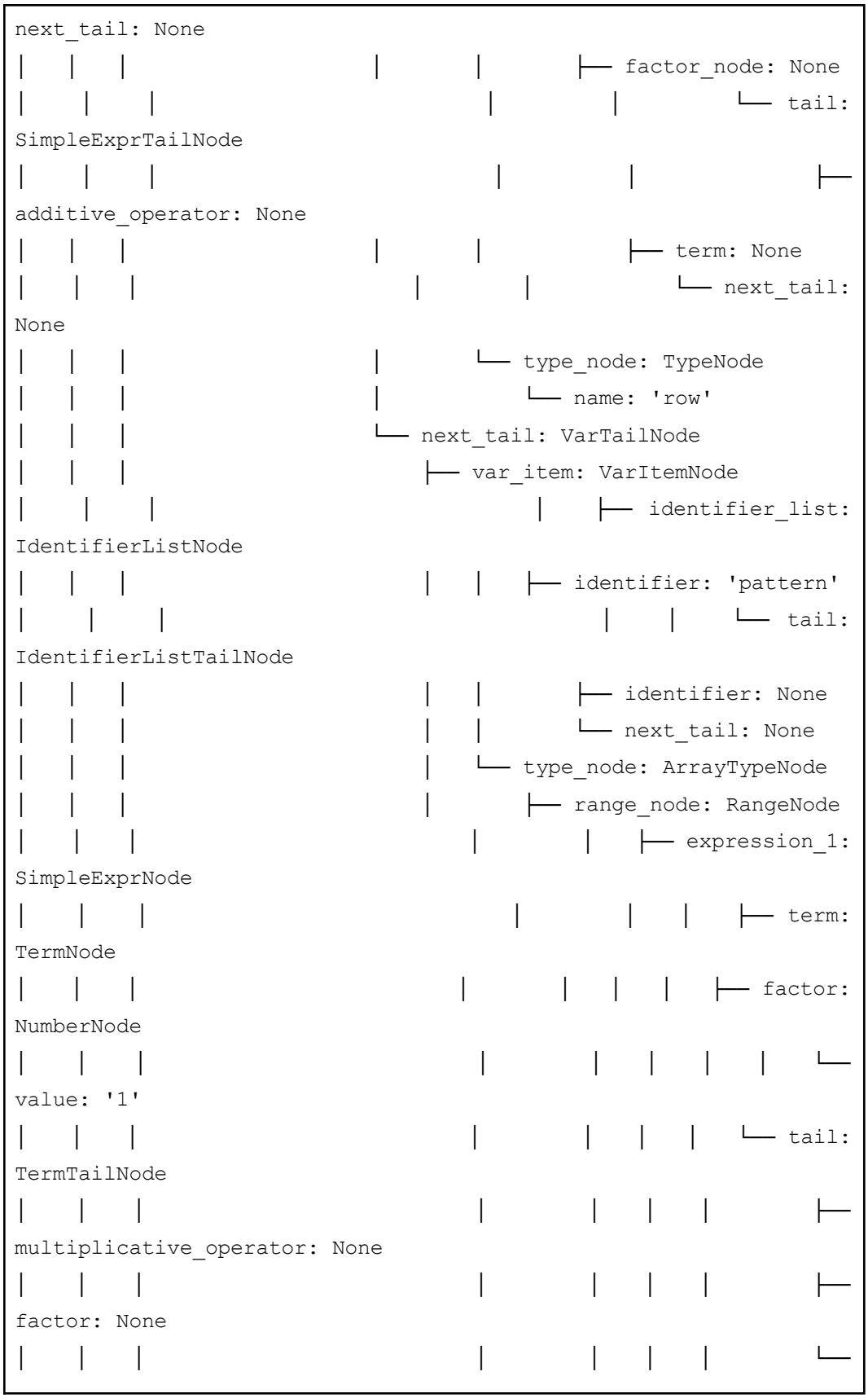
```
================================================
AST Structure:
------------------------------
ProgramNode
├── program_header: ProgramHeaderNode
│   └── identifier: 'test0'
├── declaration_part: DeclarationPartNode
│   ├── const_section: ConstSectionNode
│   │   ├── const_declaration: ConstDeclNode
│   │   │   ├── const_item: ConstItemNode
│   │   │   │   ├── identifier: 'ten'
│   │   │   │   ├── assign_operator: '='
│   │   │   │   └── value_node: NumberNode
│   │   │   │       └── value: '10'
│   │   │   └── item_tail: ConstTailNode
│   │   │       ├── const_item: ConstItemNode
│   │   │       │   ├── identifier: 'plus'
│   │   │       │   ├── assign_operator: '='
│   │   │       │   └── value_node: StringNode
│   │   │       │       └── value: ''+''
│   │   │       └── next_tail: ConstTailNode
│   │   │           ├── const_item: None
│   │   │           └── next_tail: None
│   │   └── next_section: ConstSectionNode
│   │       ├── const_declaration: None
│   │       └── next_section: None
│   ├── type_section: TypeSectionNode
│   │   ├── type_declaration: TypeDeclNode
│   │   │   ├── type_item: TypeItemNode
│   │   │   │   ├── identifier: 'row'
│   │   │   │   └── type_definition: ArrayTypeNode
│   │   │   │       ├── range_node: RangeNode
│   │   │   │       │   ├── expression_1: SimpleExprNode
│   │   │   │       │   │   ├── term: TermNode
│   │   │   │       │   │   │   ├── factor: NumberNode
│   │   │   │       │   │   │   │   └── value: '1'
│   │   │   │       │   │   │   └── tail: TermTailNode
```

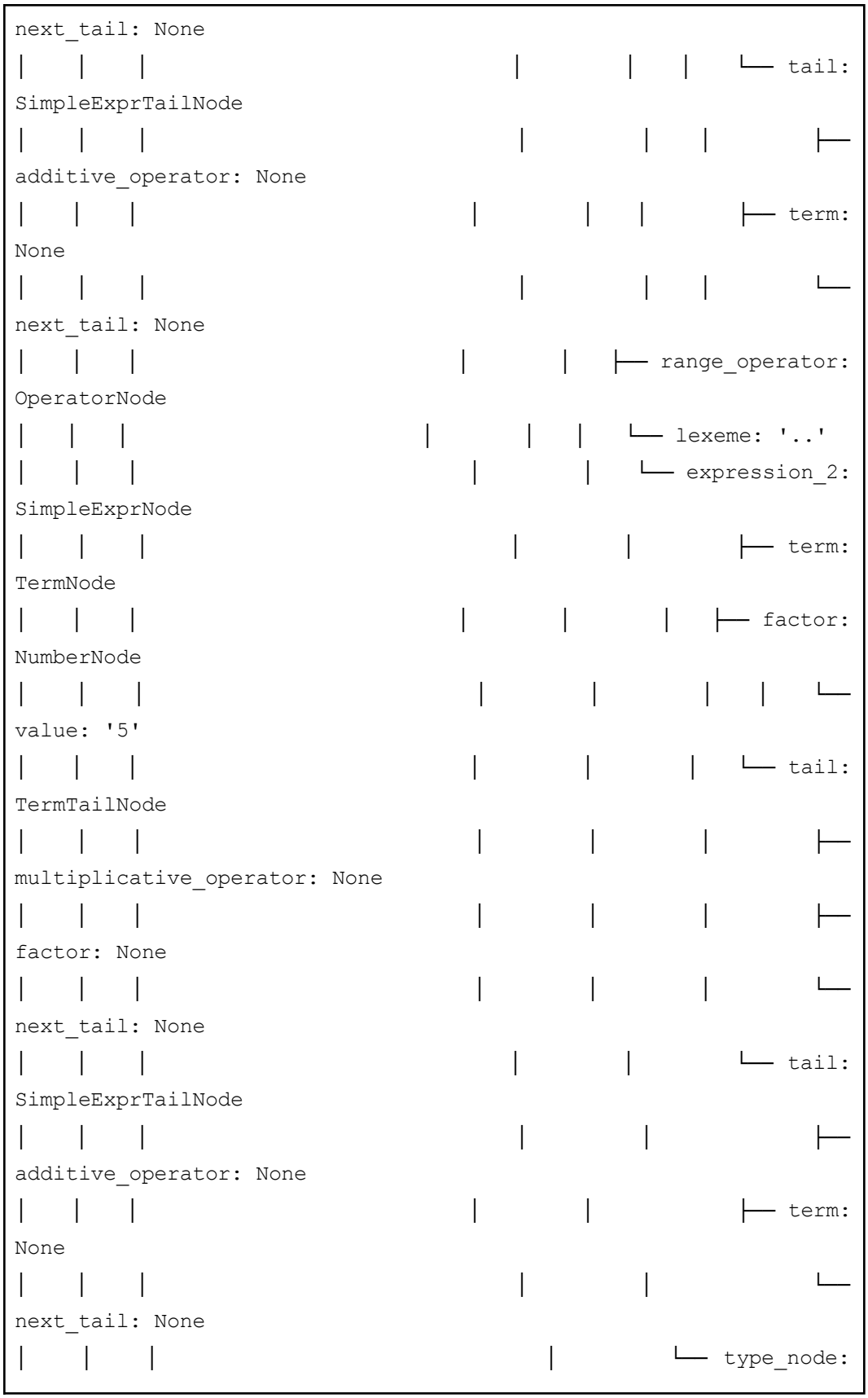

| | | | | | | ├── multiplicative_operator:
None
| | | | | | | ├── factor: None
| | | | | | | └── next_tail: None
| | | | | | └── tail: SimpleExprTailNode
| | | | | | ├── additive_operator: None
| | | | | | ├── term: None
| | | | | | └── next_tail: None
| | | | | ├── range_operator: OperatorNode
| | | | | | └── lexeme: '..'
| | | | | └── expression_2: SimpleExprNode
| | | | | ├── term: TermNode
| | | | | | ├── factor: VarNode
| | | | | | | └── identifier: 'ten'
| | | | | | └── tail: TermTailNode
| | | | | | ├── multiplicative_operator:
None
| | | | | | ├── factor: None
| | | | | | └── next_tail: None
| | | | | └── tail: SimpleExprTailNode
| | | | | ├── additive_operator: None
| | | | | ├── term: None
| | | | | └── next_tail: None
| | | | └── type_node: TypeNode
| | | | └── name: 'real'
| | | └── item_tail: TypeTailNode
| | | ├── type_item: TypeItemNode
| | | | ├── identifier: 'complex'
| | | | └── type_definition: RecordTypeNode
| | | | └── field_list: FieldListNode
| | | | ├── identifier_list:
IdentifierListNode
| | | | ├── identifier: 're'
| | | | └── tail: IdentifierListTailNode
| | | | ├── identifier: 'im'
| | | | └── next_tail:
IdentifierListTailNode

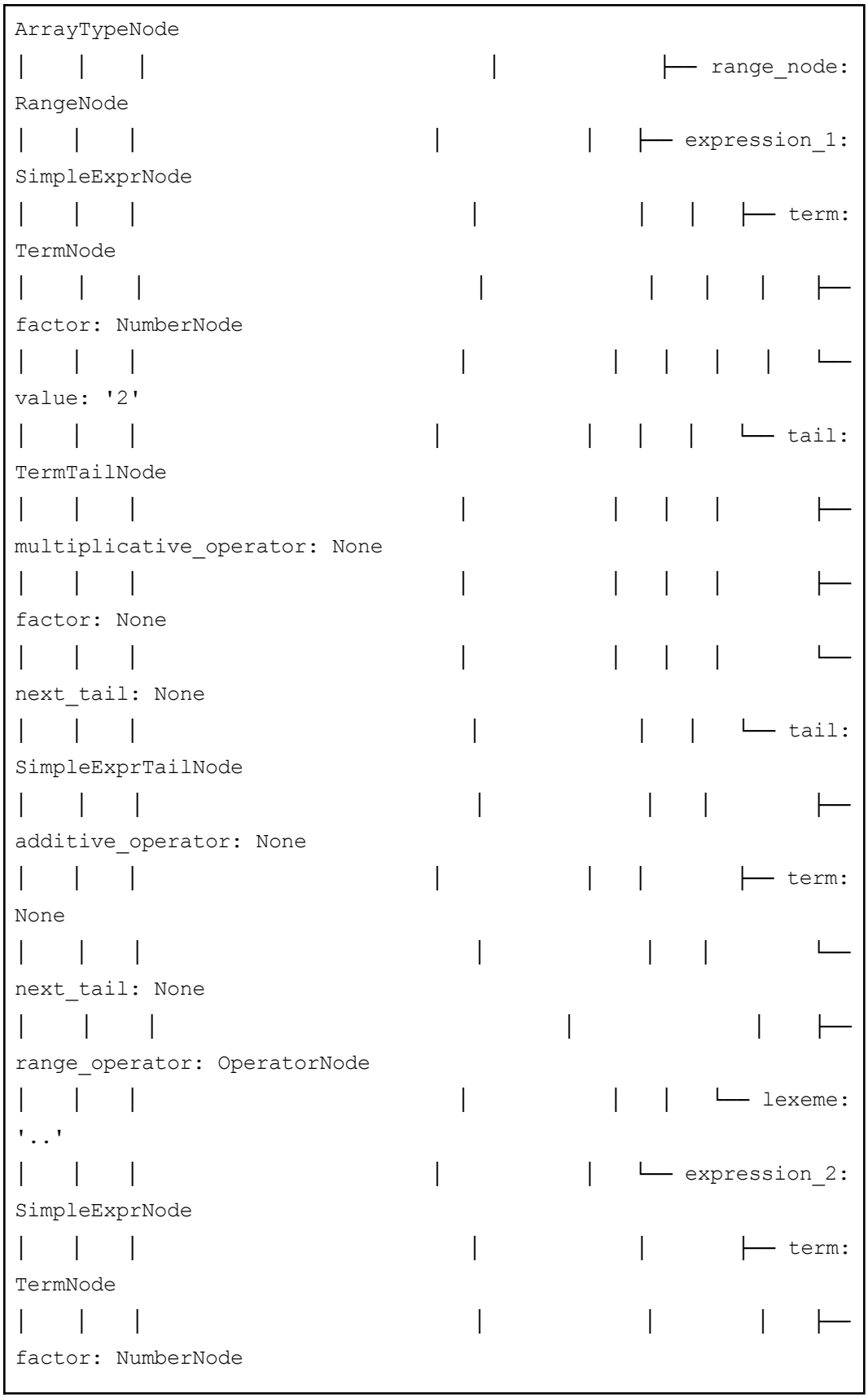
```
|   |   |       |           |           ├── identifier: None
|   |   |       |           |           └── next_tail: None
|   |   |       |           ├── type_definition: TypeNode
|   |   |       |           |   └── name: 'real'
|     |     |           |                   └── field_list_tail:
FieldListTailNode
|   |   |       └── next_tail: TypeTailNode
|   |   |           ├── type_item: None
|   |   |           └── next_tail: None
|   |   └── next_section: TypeSectionNode
|   |       ├── type_declaration: None
|   |       └── next_section: None
|   ├── var_section: VarSectionNode
|   |   ├── var_declaration: VarDeclNode
|   |   |   ├── var_item: VarItemNode
|   |   |   |   ├── identifier_list: IdentifierListNode
|   |   |   |   |   ├── identifier: 'i'
|   |   |   |   |   └── tail: IdentifierListTailNode
|   |   |   |   |       ├── identifier: 'j'
|   |   |   |   |       └── next_tail: IdentifierListTailNode
|   |   |   |   |           ├── identifier: None
|   |   |   |   |           └── next_tail: None
|   |   |   |   └── type_node: TypeNode
|   |   |   |       └── name: 'integer'
|   |   |   └── item_tail: VarTailNode
|   |   |       ├── var_item: VarItemNode
|   |   |       |   ├── identifier_list: IdentifierListNode
|   |   |       |   |   ├── identifier: 'p'
|   |   |       |   |   └── tail: IdentifierListTailNode
|   |   |       |   |       ├── identifier: None
|   |   |       |   |       └── next_tail: None
|   |   |       |   └── type_node: TypeNode
|   |   |       |       └── name: 'boolean'
|   |   |       └── next_tail: VarTailNode
|   |   |           ├── var_item: VarItemNode
|   |   |           |   ├── identifier_list: IdentifierListNode
|   |   |           |   |   ├── identifier: 'z'
```

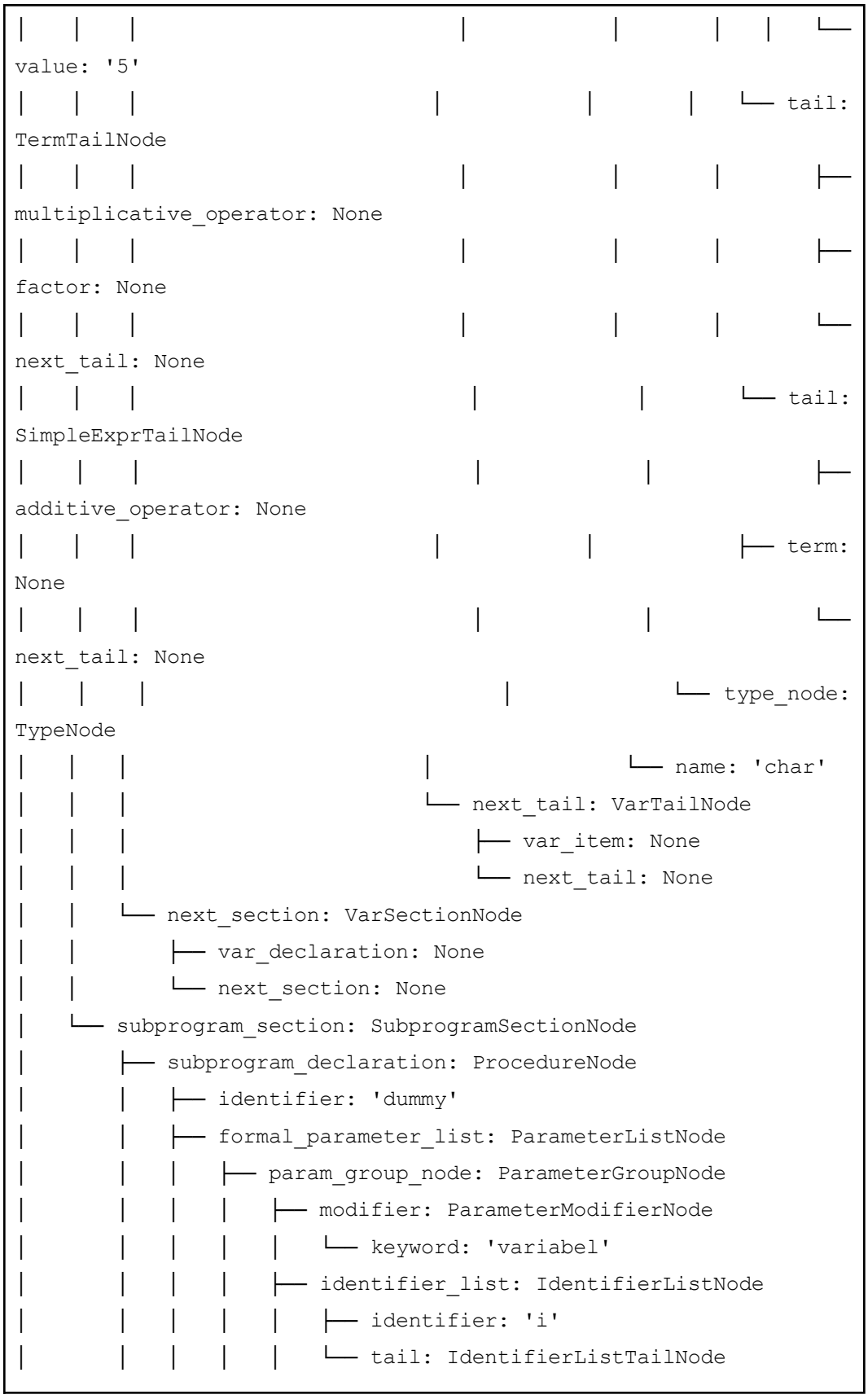

└── tail: IdentifierListTailNode
├── identifier: None
└── next_tail: None
└── type_node: TypeNode
└── name: 'complex'
└── next_tail: VarTailNode
├── var_item: VarItemNode
├── identifier_list: IdentifierListNode
├── identifier: 'i'
└── tail: IdentifierListTailNode
├── identifier: None
└── next_tail: None
└── type_node: TypeNode
└── name: 'real'
└── next_tail: VarTailNode
├── var_item: VarItemNode
├── identifier_list: IdentifierListNode
├── identifier: 'matrix'
└── tail: IdentifierListTailNode
├── identifier: None
└── next_tail: None
└── type_node: ArrayTypeNode
├── range_node: RangeNode
├── expression_1: UnaryOpNode
├── operator: OperatorNode
└── lexeme: '-'
├── term_node: TermNode
├── factor: NumberNode
└── value: '3'

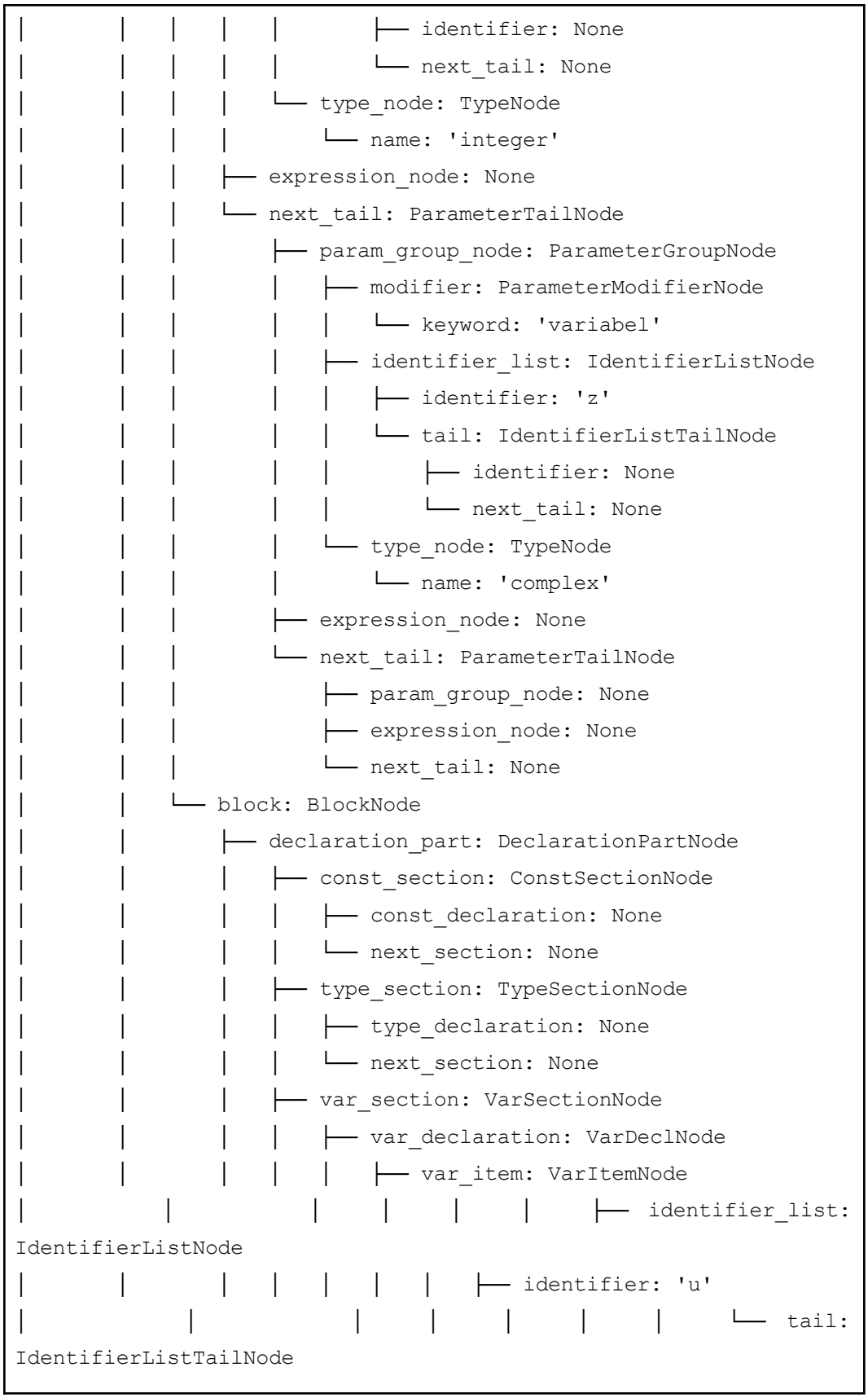

| | | | | | | └── tail:
TermTailNode
| | | | | | | ├──
multiplicative_operator: None
| | | | | | | ├── factor:
None
| | | | | | | └──
next_tail: None
| | | | | | ├── factor_node: None
| | | | | | └── tail:
SimpleExprTailNode
| | | | | | ├──
additive_operator: None
| | | | | | ├── term: None
| | | | | | └── next_tail:
None
| | | | | ├── range_operator:
OperatorNode
| | | | | | └── lexeme: '..'
| | | | | └── expression_2:
UnaryOpNode
| | | | | ├── operator:
OperatorNode
| | | | | | └── lexeme: '+'
| | | | | ├── term_node:
TermNode
| | | | | | ├── factor:
NumberNode
| | | | | | | └── value:
'3'
| | | | | └── tail:
TermTailNode
| | | | | ├──
multiplicative_operator: None
| | | | | ├── factor:
None
| | | | | └──

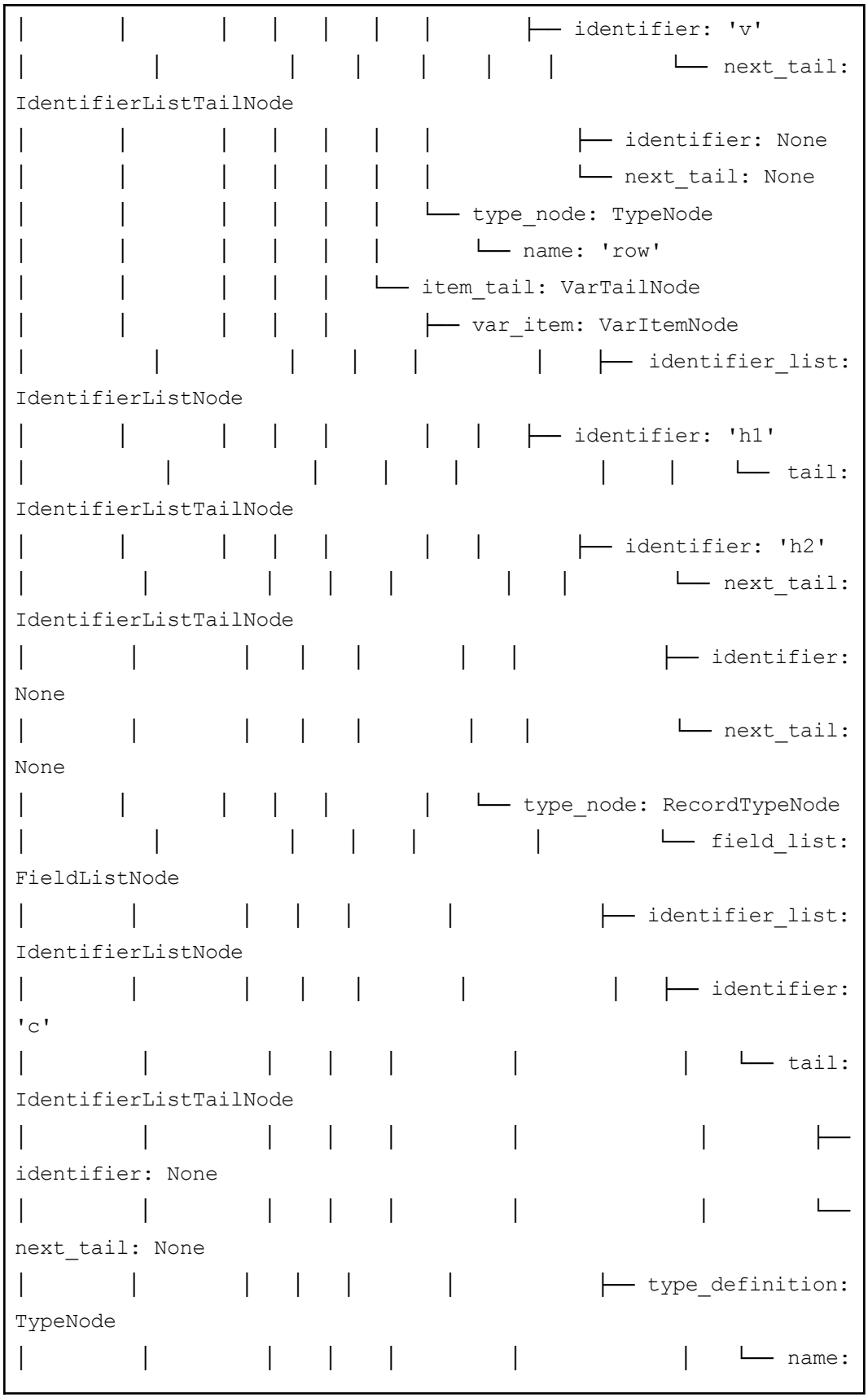

next_tail: None
| | | | | ├── factor_node: None
| | | | | └── tail:
SimpleExprTailNode
| | | | | ├──
additive_operator: None
| | | | | ├── term: None
| | | | | └── next_tail:
None
| | | | └── type_node: TypeNode
| | | | └── name: 'row'
| | | | └── next_tail: VarTailNode
| | | | ├── var_item: VarItemNode
| | | | ├── identifier_list:
IdentifierListNode
| | | | | ├── identifier: 'pattern'
| | | | | └── tail:
IdentifierListTailNode
| | | | | ├── identifier: None
| | | | | └── next_tail: None
| | | | └── type_node: ArrayTypeNode
| | | | ├── range_node: RangeNode
| | | | ├── expression_1:
SimpleExprNode
| | | | ├── term:
TermNode
| | | | ├── factor:
NumberNode
| | | | └──
value: '1'
| | | | └── tail:
TermTailNode
| | | | ├──
multiplicative_operator: None
| | | | ├──
factor: None
| | | | └──

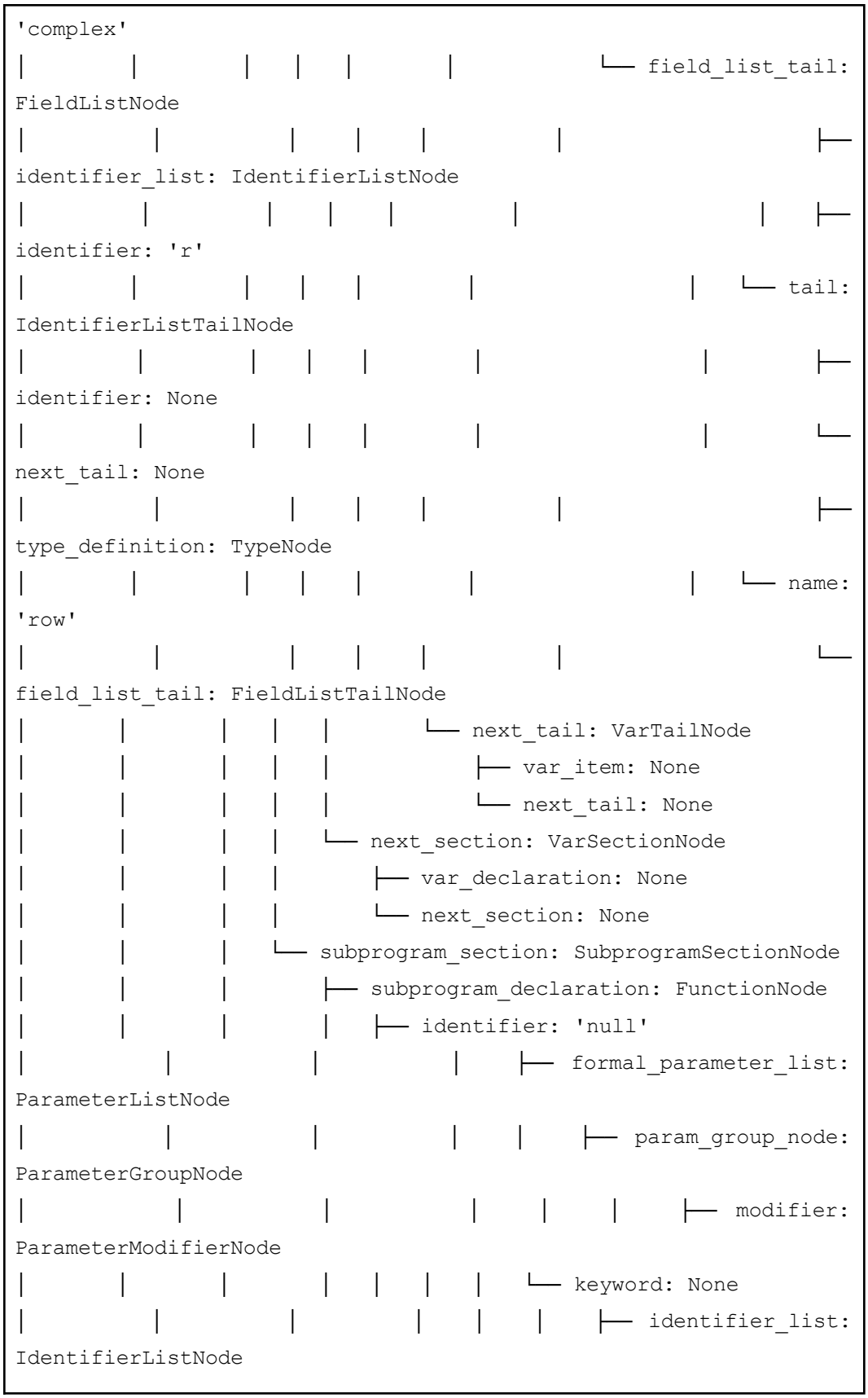

next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
range_operator: OperatorNode
lexeme: '..'
expression_2: SimpleExprNode
term: TermNode
factor: NumberNode
value: '5'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
type_node:

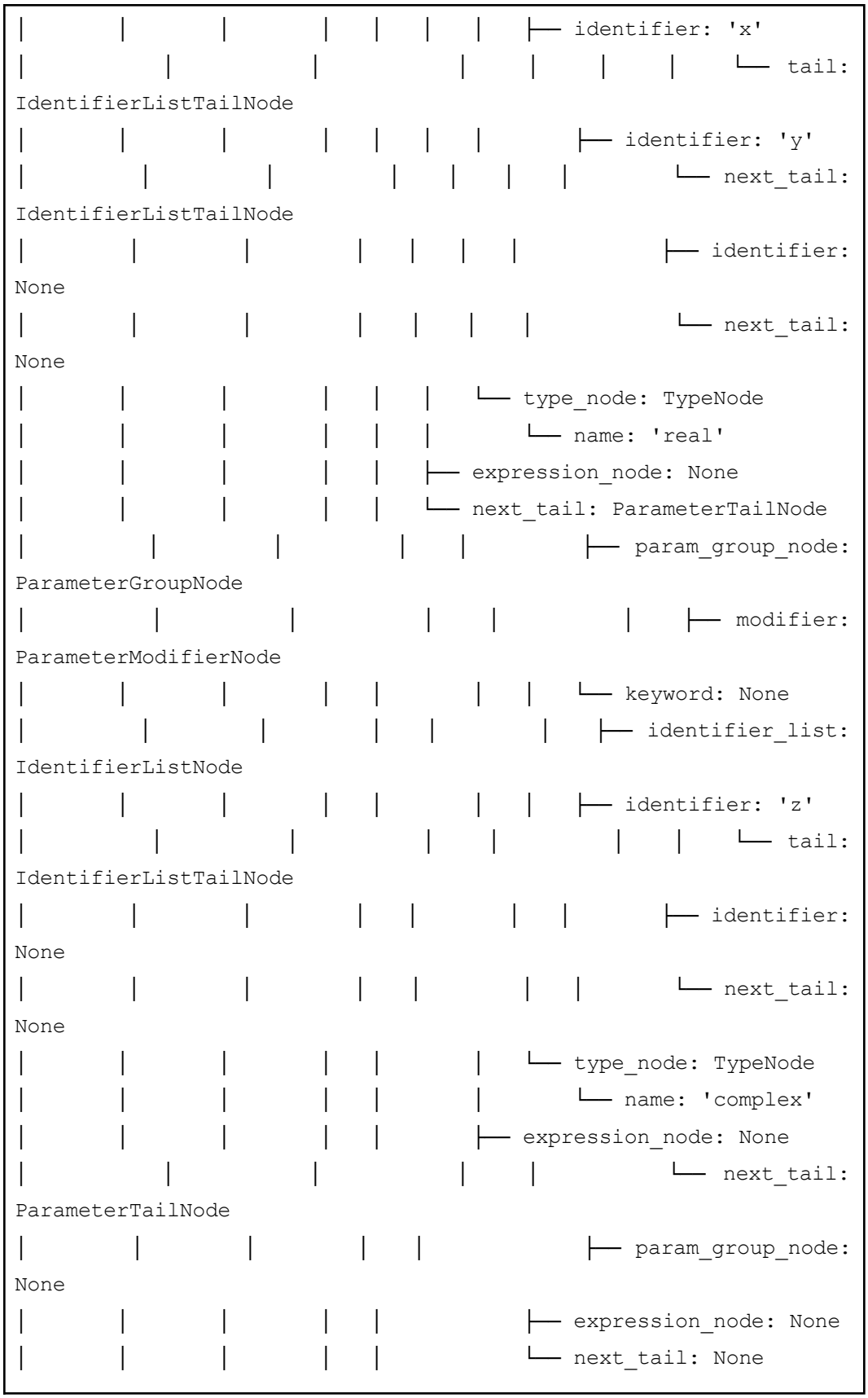

ArrayTypeNode
├─ range_node: RangeNode
│ ├─ expression_1: SimpleExprNode
│ │ ├─ term: TermNode
│ │ │ ├─ factor: NumberNode
│ │ │ │ └─ value: '2'
│ │ │ └─ tail: TermTailNode
│ │ │   ├─ multiplicative_operator: None
│ │ │   ├─ factor: None
│ │ │   └─ next_tail: None
│ │ └─ tail: SimpleExprTailNode
│ │   ├─ additive_operator: None
│ │   ├─ term: None
│ │   └─ next_tail: None
│ ├─ range_operator: OperatorNode
│ │ └─ lexeme: '..'
│ └─ expression_2: SimpleExprNode
│   ├─ term: TermNode
│   │ ├─ factor: NumberNode

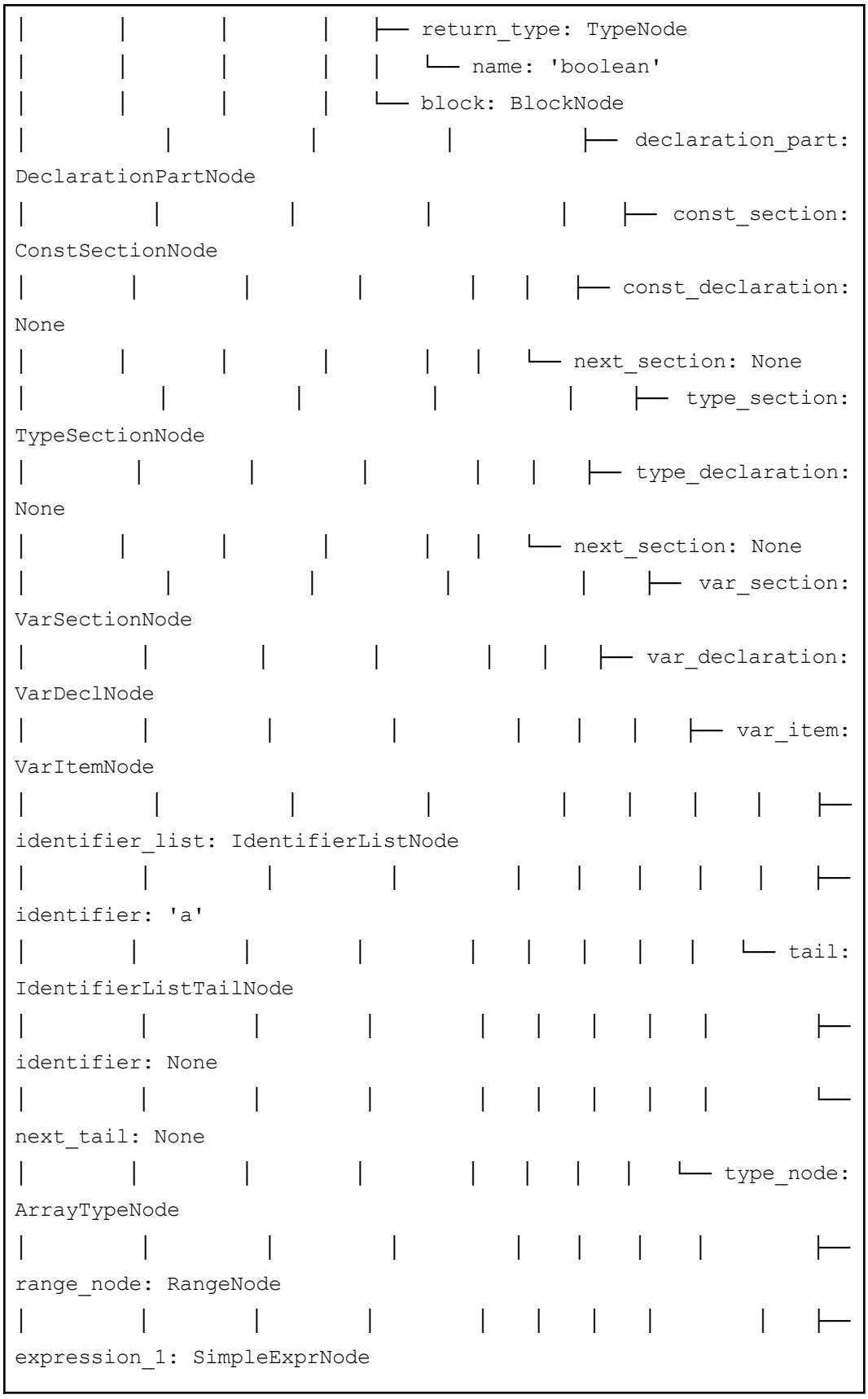

value: '5'
TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
type_node: TypeNode
name: 'char'
next_tail: VarTailNode
var_item: None
next_tail: None
next_section: VarSectionNode
var_declaration: None
next_section: None
subprogram_section: SubprogramSectionNode
subprogram_declaration: ProcedureNode
identifier: 'dummy'
formal_parameter_list: ParameterListNode
param_group_node: ParameterGroupNode
modifier: ParameterModifierNode
keyword: 'variabel'
identifier_list: IdentifierListNode
identifier: 'i'
tail: IdentifierListTailNode

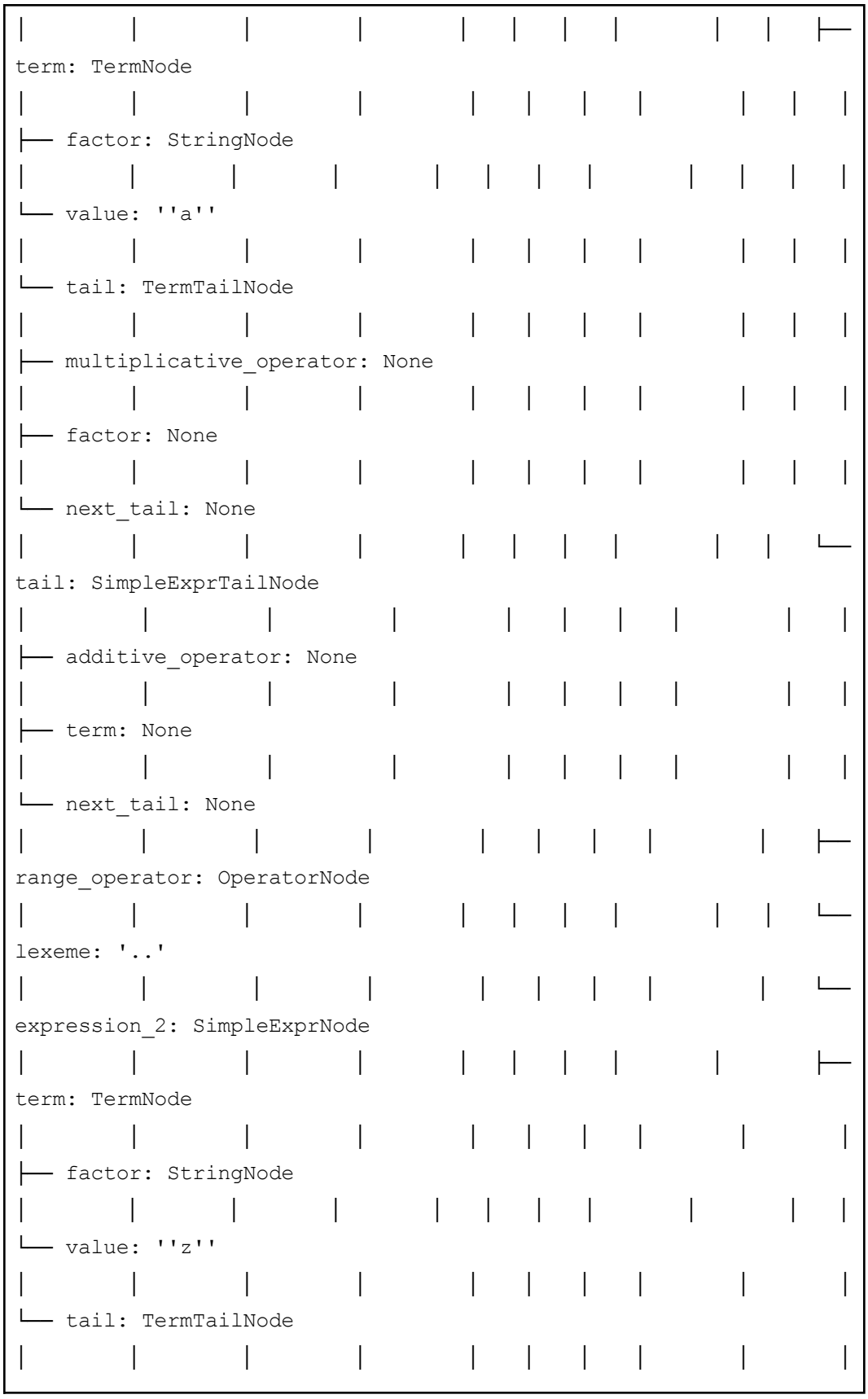

identifier: None
next_tail: None
type_node: TypeNode
name: 'integer'
expression_node: None
next_tail: ParameterTailNode
param_group_node: ParameterGroupNode
modifier: ParameterModifierNode
keyword: 'variabel'
identifier_list: IdentifierListNode
identifier: 'z'
tail: IdentifierListTailNode
identifier: None
next_tail: None
type_node: TypeNode
name: 'complex'
expression_node: None
next_tail: ParameterTailNode
param_group_node: None
expression_node: None
next_tail: None
block: BlockNode
declaration_part: DeclarationPartNode
const_section: ConstSectionNode
const_declaration: None
next_section: None
type_section: TypeSectionNode
type_declaration: None
next_section: None
var_section: VarSectionNode
var_declaration: VarDeclNode
var_item: VarItemNode
identifier_list: IdentifierListNode
identifier: 'u'
tail: IdentifierListTailNode

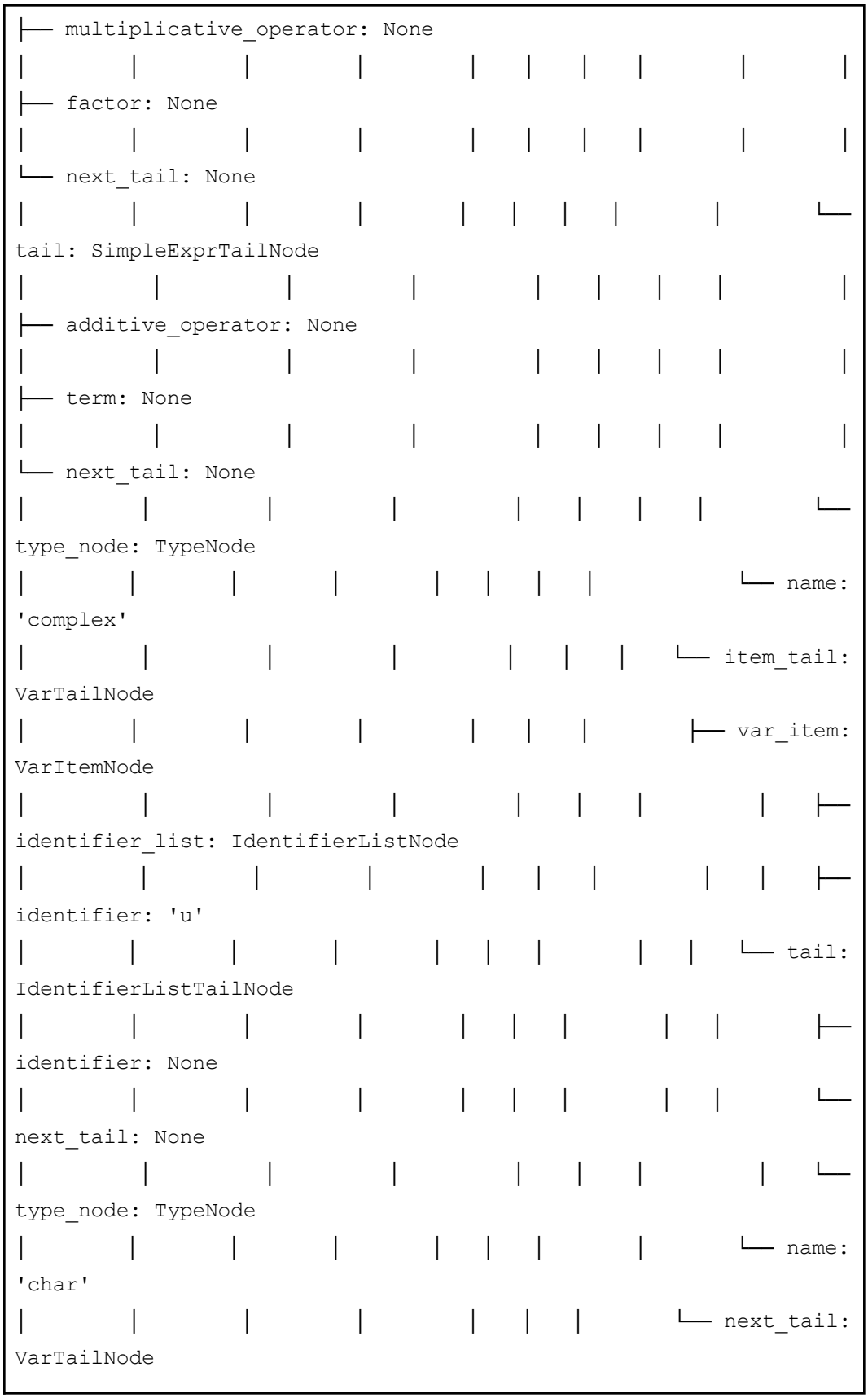

identifier: 'v'
next_tail:
IdentifierListTailNode
identifier: None
next_tail: None
type_node: TypeNode
name: 'row'
item_tail: VarTailNode
var_item: VarItemNode
identifier_list:
IdentifierListNode
identifier: 'h1'
tail:
IdentifierListTailNode
identifier: 'h2'
next_tail:
IdentifierListTailNode
identifier:
None
next_tail:
None
type_node: RecordTypeNode
field_list:
FieldListNode
identifier_list:
IdentifierListNode
identifier:
'c'
tail:
IdentifierListTailNode
identifier: None
next_tail: None
type_definition:
TypeNode
name:

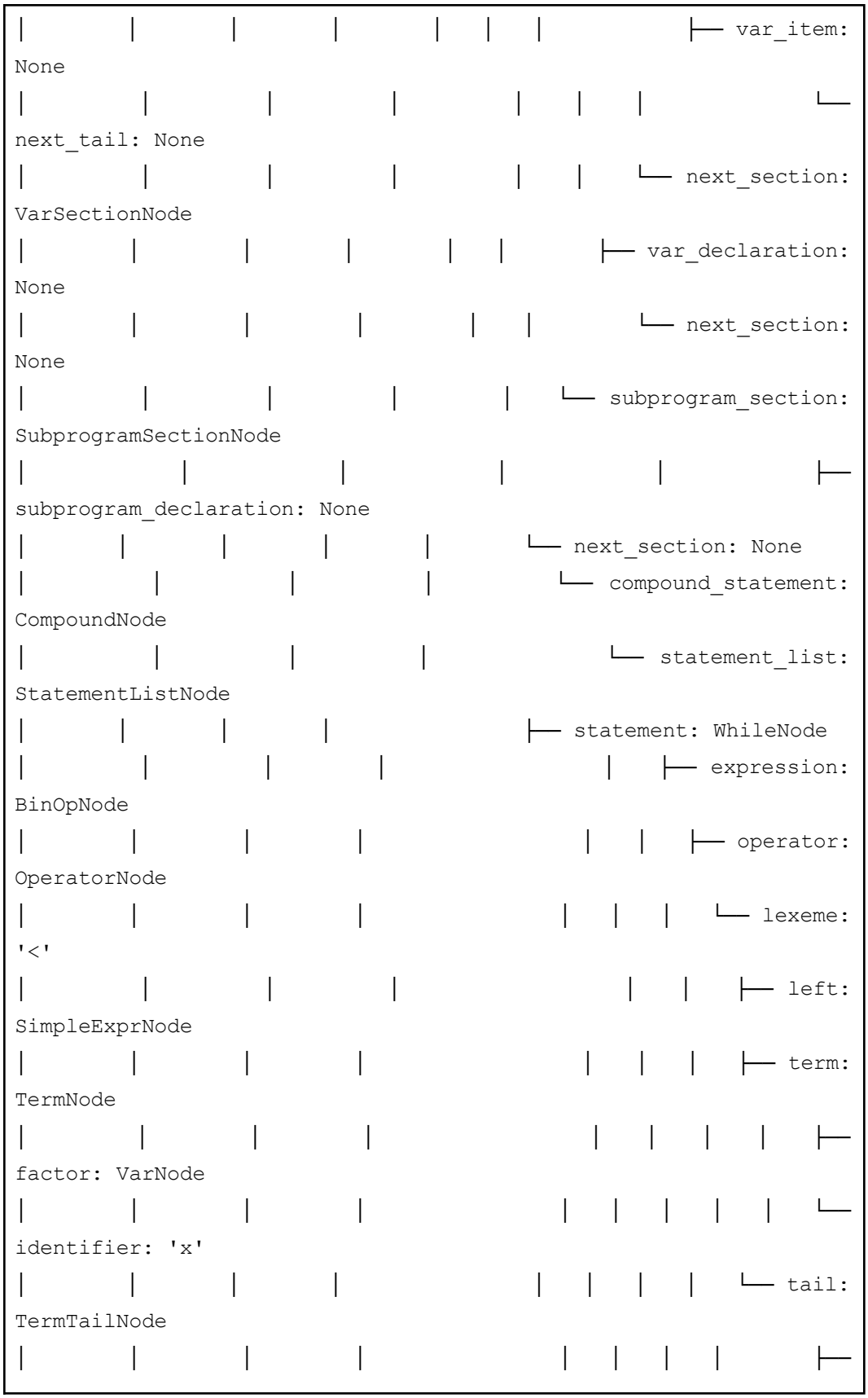

'complex'
└── field_list_tail: FieldListNode
├── identifier_list: IdentifierListNode
│ ├── identifier: 'r'
│ └── tail: IdentifierListTailNode
│ ├── identifier: None
│ └── next_tail: None
├── type_definition: TypeNode
│ └── name: 'row'
└── field_list_tail: FieldListTailNode
└── next_tail: VarTailNode
├── var_item: None
└── next_tail: None
└── next_section: VarSectionNode
├── var_declaration: None
└── next_section: None
└── subprogram_section: SubprogramSectionNode
├── subprogram_declaration: FunctionNode
│ ├── identifier: 'null'
│ ├── formal_parameter_list: ParameterListNode
│ ├── param_group_node: ParameterGroupNode
│ ├── modifier: ParameterModifierNode
│ └── keyword: None
├── identifier_list: IdentifierListNode

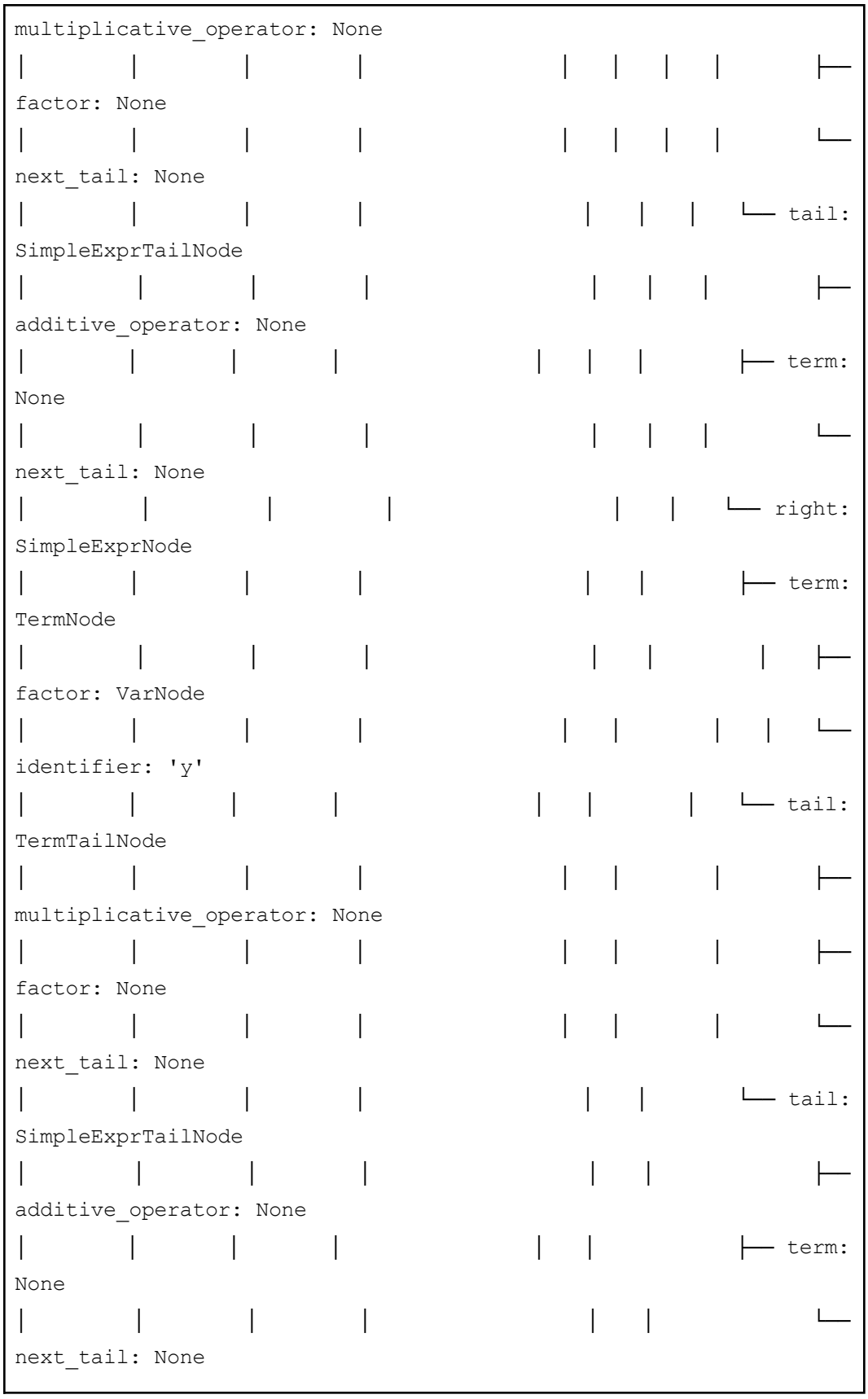

| | | | | | | ├── identifier: 'x'
| | | | | | | └── tail: IdentifierListTailNode
| | | | | | | ├── identifier: 'y'
| | | | | | | └── next_tail: IdentifierListTailNode
| | | | | | | ├── identifier: None
| | | | | | | └── next_tail: None
| | | | | | └── type_node: TypeNode
| | | | | | └── name: 'real'
| | | | | ├── expression_node: None
| | | | | └── next_tail: ParameterTailNode
| | | | | ├── param_group_node: ParameterGroupNode
| | | | | | ├── modifier: ParameterModifierNode
| | | | | | └── keyword: None
| | | | | | ├── identifier_list: IdentifierListNode
| | | | | | ├── identifier: 'z'
| | | | | | | └── tail: IdentifierListTailNode
| | | | | | ├── identifier: None
| | | | | | └── next_tail: None
| | | | | └── type_node: TypeNode
| | | | | └── name: 'complex'
| | | | ├── expression_node: None
| | | | └── next_tail: ParameterTailNode
| | | | ├── param_group_node: None
| | | | ├── expression_node: None
| | | | └── next_tail: None

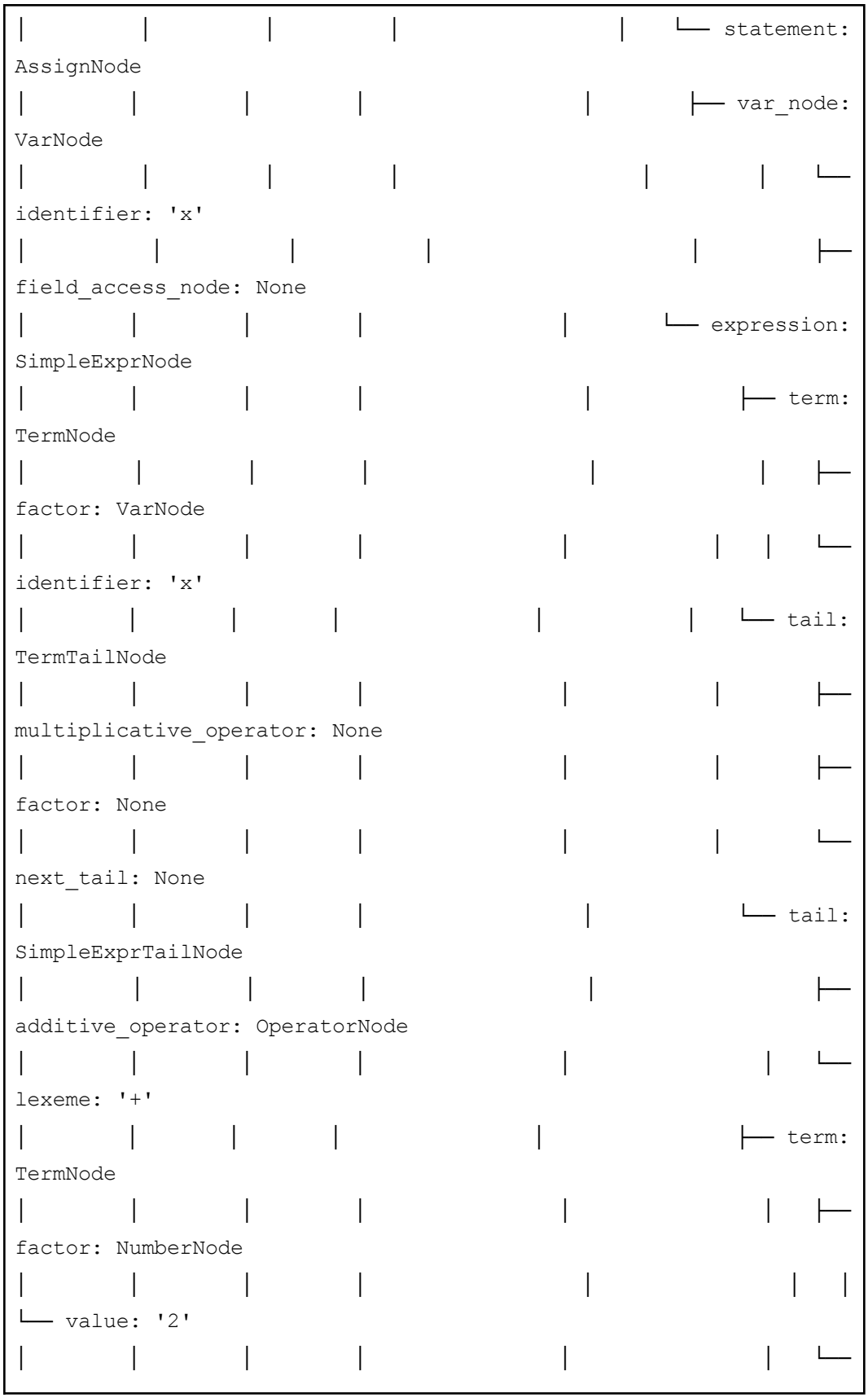

| | | | ├── return_type: TypeNode
| | | | │ └── name: 'boolean'
| | | | └── block: BlockNode
| | | | ├── declaration_part: DeclarationPartNode
| | | | | ├── const_section: ConstSectionNode
| | | | | | ├── const_declaration: None
| | | | | | └── next_section: None
| | | | | ├── type_section: TypeSectionNode
| | | | | | ├── type_declaration: None
| | | | | | └── next_section: None
| | | | | ├── var_section: VarSectionNode
| | | | | | ├── var_declaration: VarDeclNode
| | | | | | | ├── var_item: VarItemNode
| | | | | | | | ├── identifier_list: IdentifierListNode
| | | | | | | | | ├── identifier: 'a'
| | | | | | | | | └── tail: IdentifierListTailNode
| | | | | | | | | ├── identifier: None
| | | | | | | | | └── next_tail: None
| | | | | | | | └── type_node: ArrayTypeNode
| | | | | | | | ├── range_node: RangeNode
| | | | | | | | | ├── expression_1: SimpleExprNode

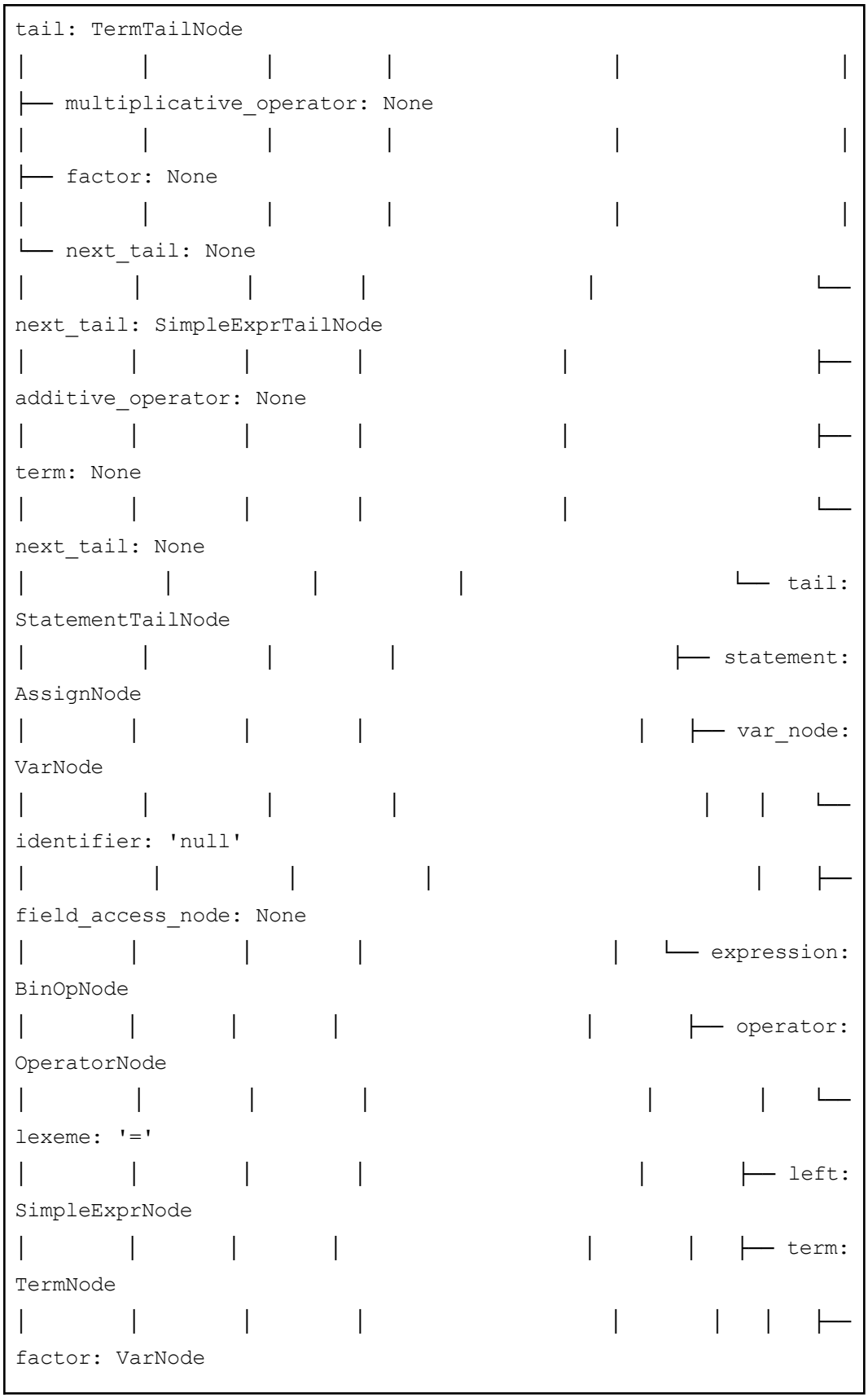

term: TermNode
├── factor: StringNode
└── value: ''a''
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
range_operator: OperatorNode
└── lexeme: '..'
expression_2: SimpleExprNode
term: TermNode
├── factor: StringNode
└── value: ''z''
└── tail: TermTailNode

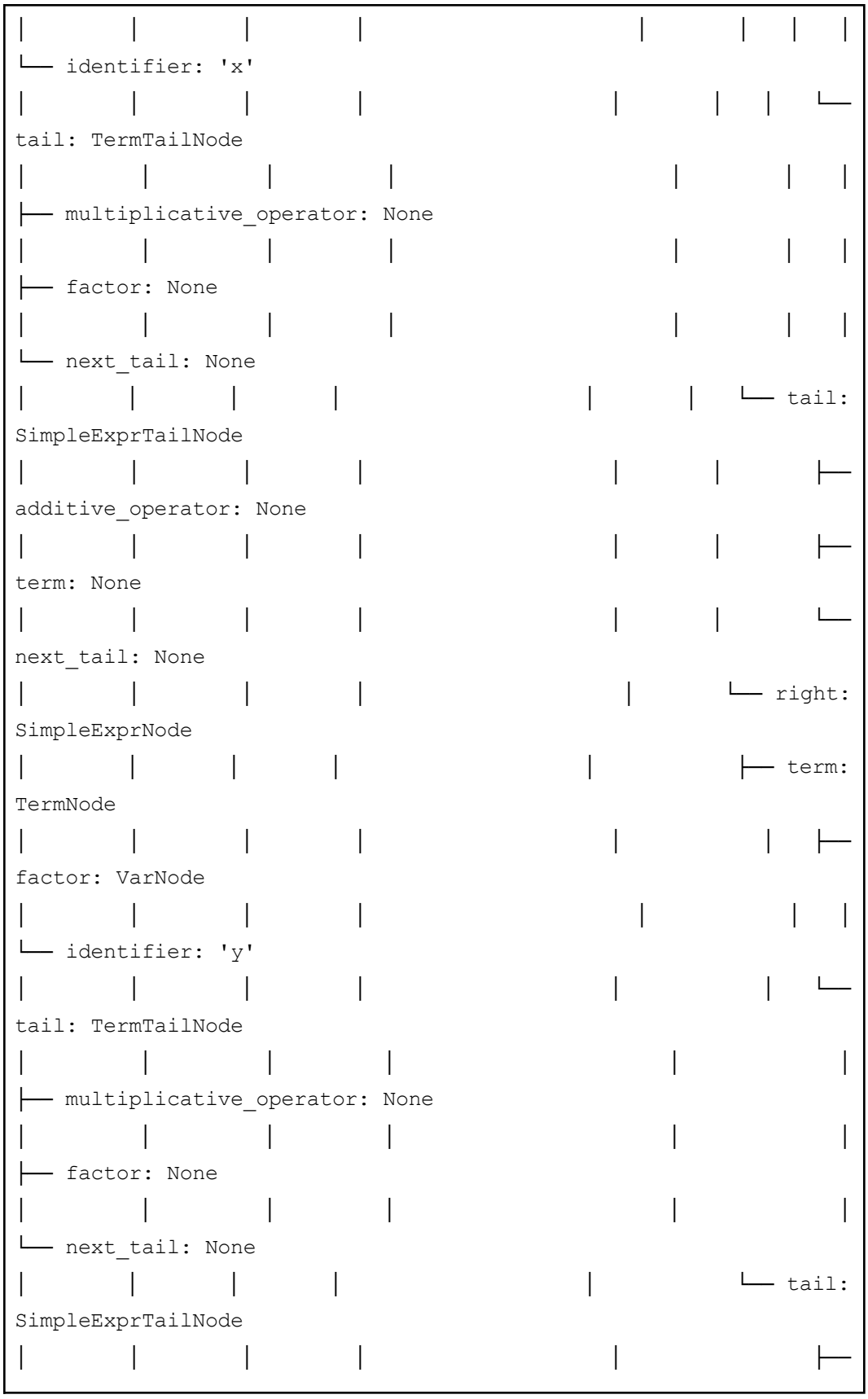

├── multiplicative_operator: None
│ │ │ │ │ │ │ │ │
├── factor: None
│ │ │ │ │ │ │ │ │
└── next_tail: None
│ │ │ │ │ │ │ │ └──
tail: SimpleExprTailNode
│ │ │ │ │ │ │ │
├── additive_operator: None
│ │ │ │ │ │ │ │
├── term: None
│ │ │ │ │ │ │ │
└── next_tail: None
│ │ │ │ │ │ │ └──
type_node: TypeNode
│ │ │ │ │ │ │ └── name:
'complex'
│ │ │ │ │ │ └── item_tail:
VarTailNode
│ │ │ │ │ │ ├── var_item:
VarItemNode
│ │ │ │ │ │ │ ├──
identifier_list: IdentifierListNode
│ │ │ │ │ │ │ │ ├──
identifier: 'u'
│ │ │ │ │ │ │ │ └── tail:
IdentifierListTailNode
│ │ │ │ │ │ │ │ ├──
identifier: None
│ │ │ │ │ │ │ │ └──
next_tail: None
│ │ │ │ │ │ │ └──
type_node: TypeNode
│ │ │ │ │ │ │ └── name:
'char'
│ │ │ │ │ │ └── next_tail:
VarTailNode

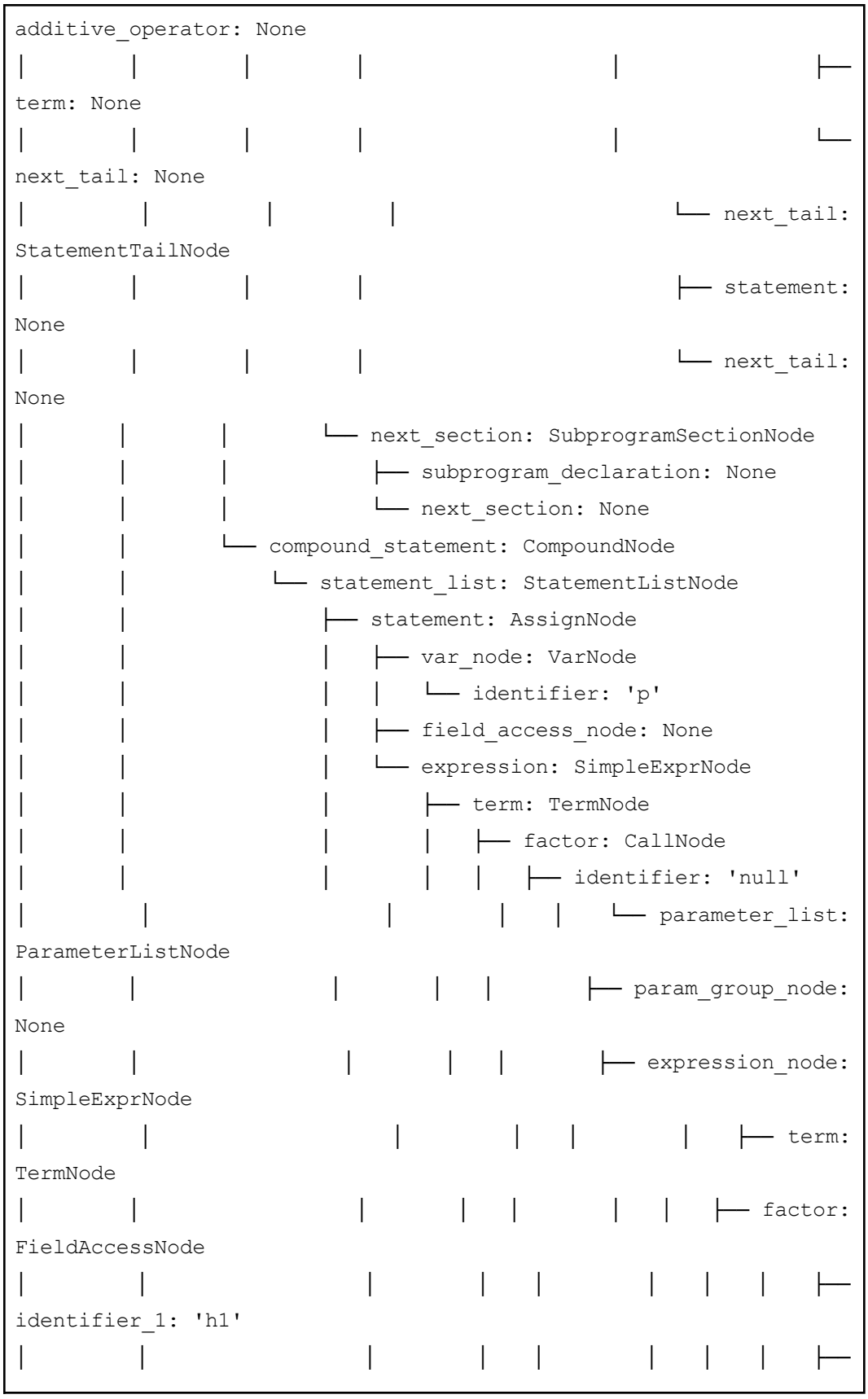

| | | | | | | ├── var_item: None
| | | | | | | └── next_tail: None
| | | | | | └── next_section: VarSectionNode
| | | | | | ├── var_declaration: None
| | | | | | └── next_section: None
| | | | | └── subprogram_section: SubprogramSectionNode
| | | | | | ├── subprogram_declaration: None
| | | | | └── next_section: None
| | | | └── compound_statement: CompoundNode
| | | | └── statement_list: StatementListNode
| | | | ├── statement: WhileNode
| | | | | ├── expression: BinOpNode
| | | | | | ├── operator: OperatorNode
| | | | | | | └── lexeme: '<'
| | | | | | ├── left: SimpleExprNode
| | | | | | | ├── term: TermNode
| | | | | | | | ├── factor: VarNode
| | | | | | | | | └── identifier: 'x'
| | | | | | | | └── tail: TermTailNode
| | | | | | | | ├──

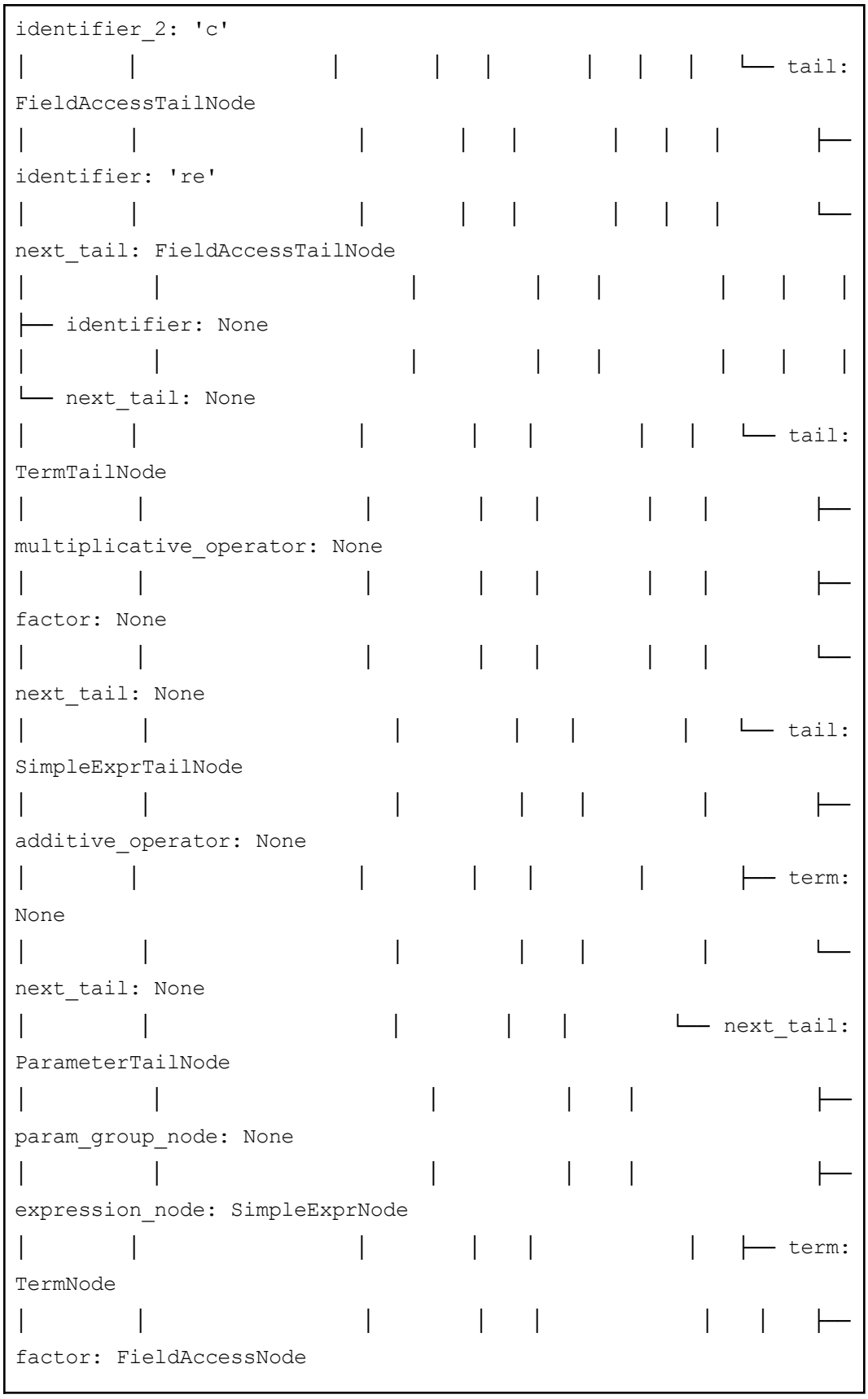

multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
right: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'y'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None

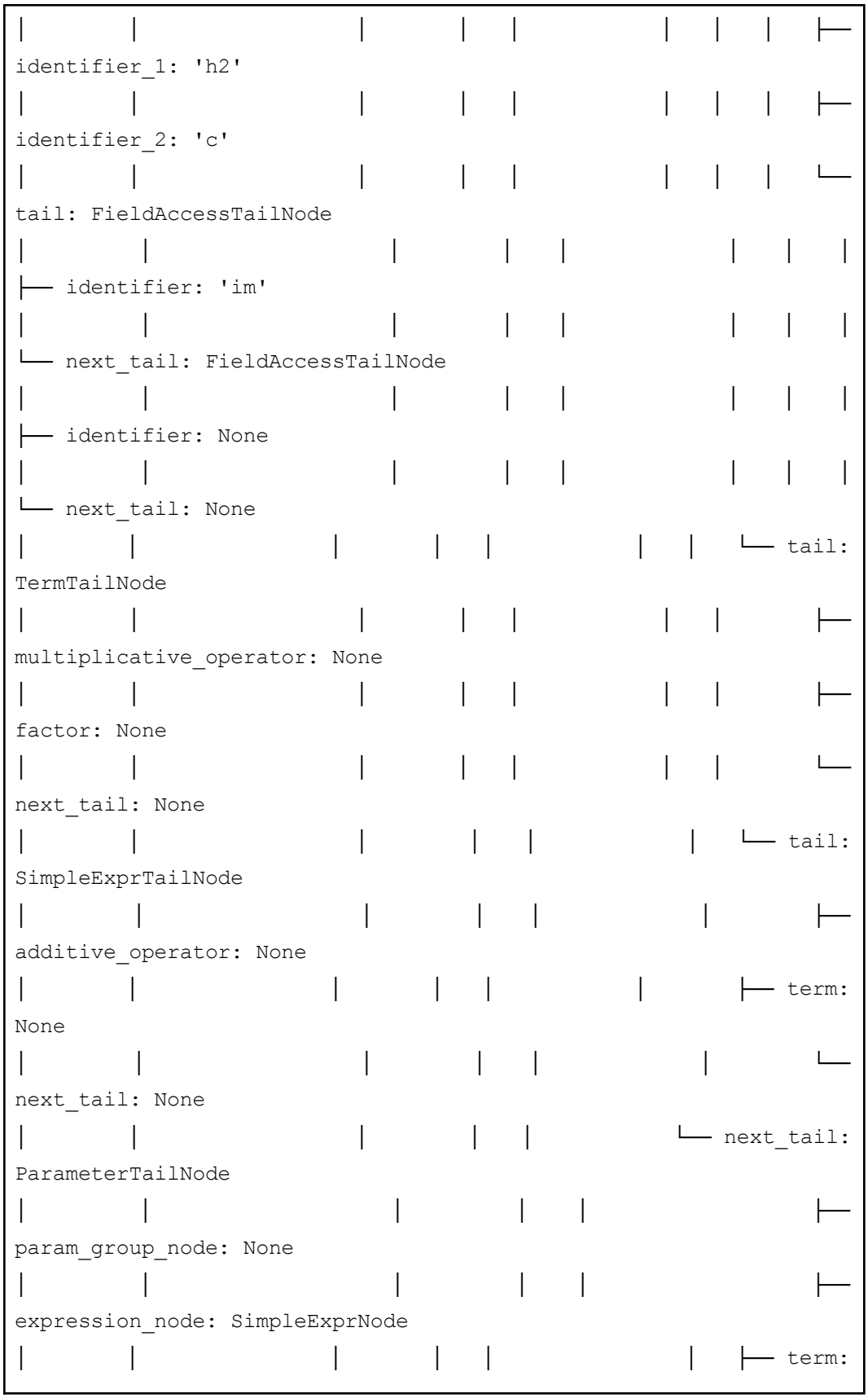

└── statement: AssignNode
├── var_node: VarNode
└── identifier: 'x'
├── field_access_node: None
└── expression: SimpleExprNode
├── term: TermNode
├── factor: VarNode
└── identifier: 'x'
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: OperatorNode
└── lexeme: '+'
├── term: TermNode
├── factor: NumberNode
└── value: '2'
└──

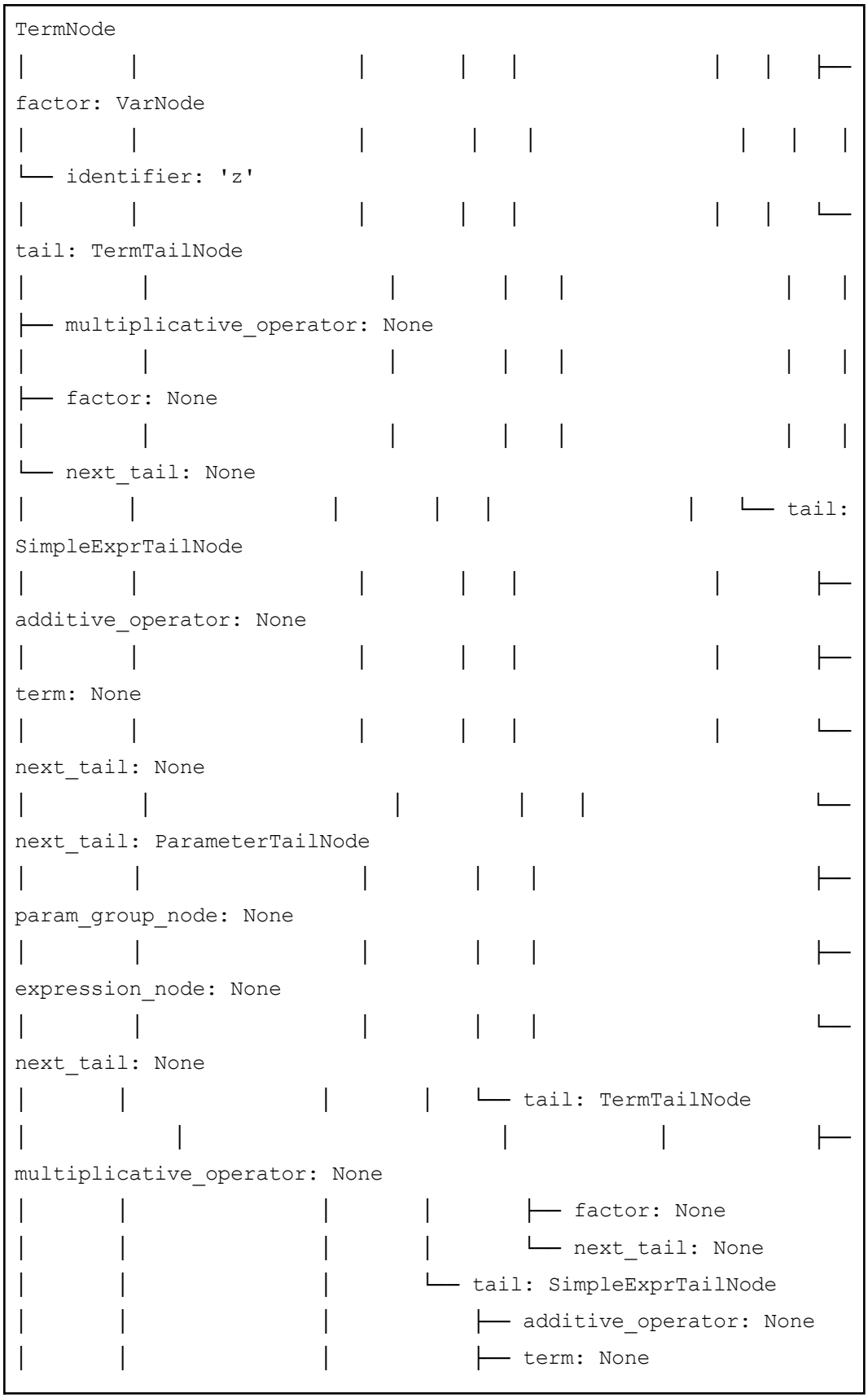

tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
next_tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
tail: StatementTailNode
statement: AssignNode
var_node: VarNode
identifier: 'null'
field_access_node: None
expression: BinOpNode
operator: OperatorNode
lexeme: '='
left: SimpleExprNode
term: TermNode
factor: VarNode

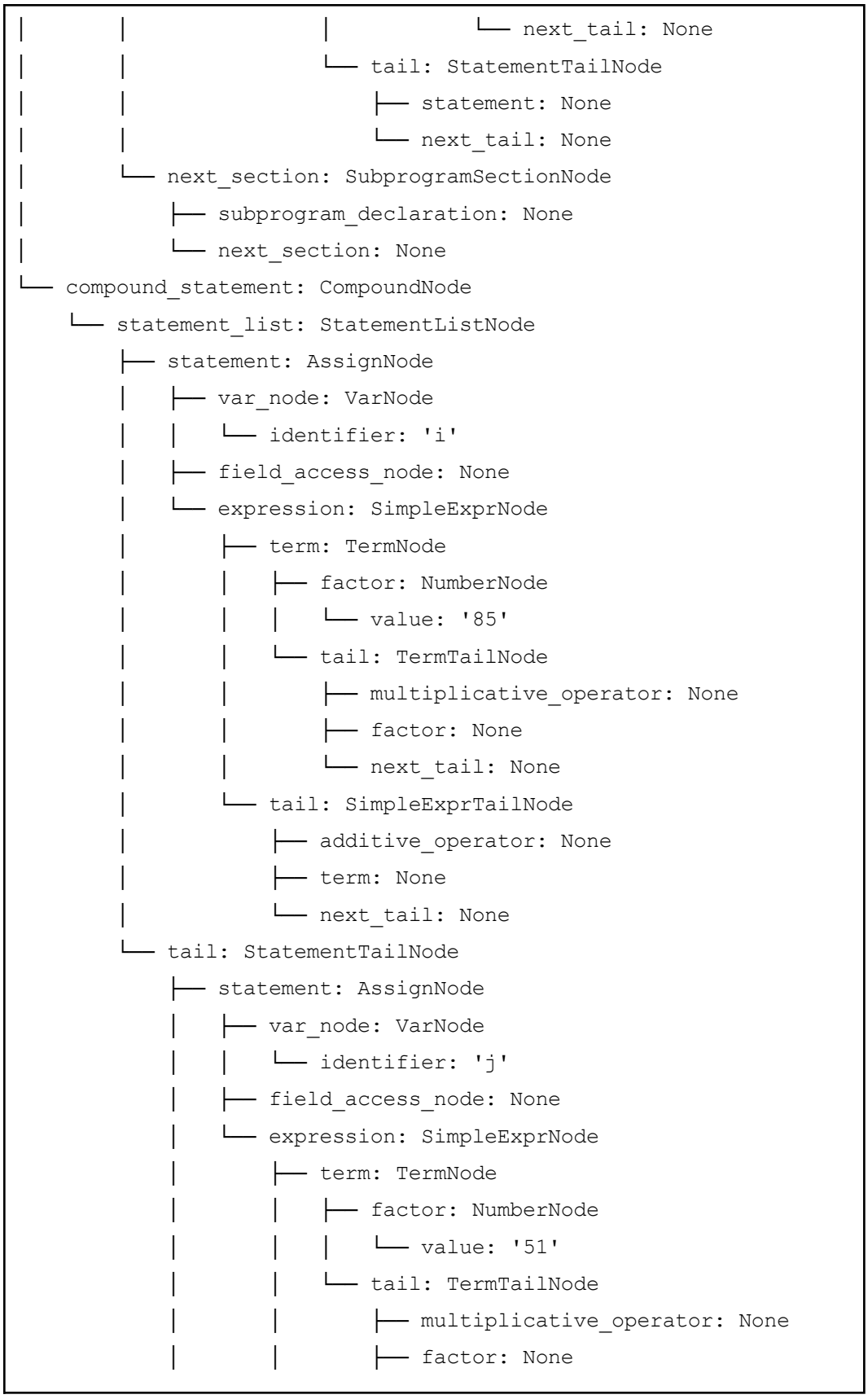

└── identifier: 'x'
tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail:
SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
└── right:
SimpleExprNode
├── term:
TermNode
factor: VarNode
└── identifier: 'y'
tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail:
SimpleExprTailNode
├──

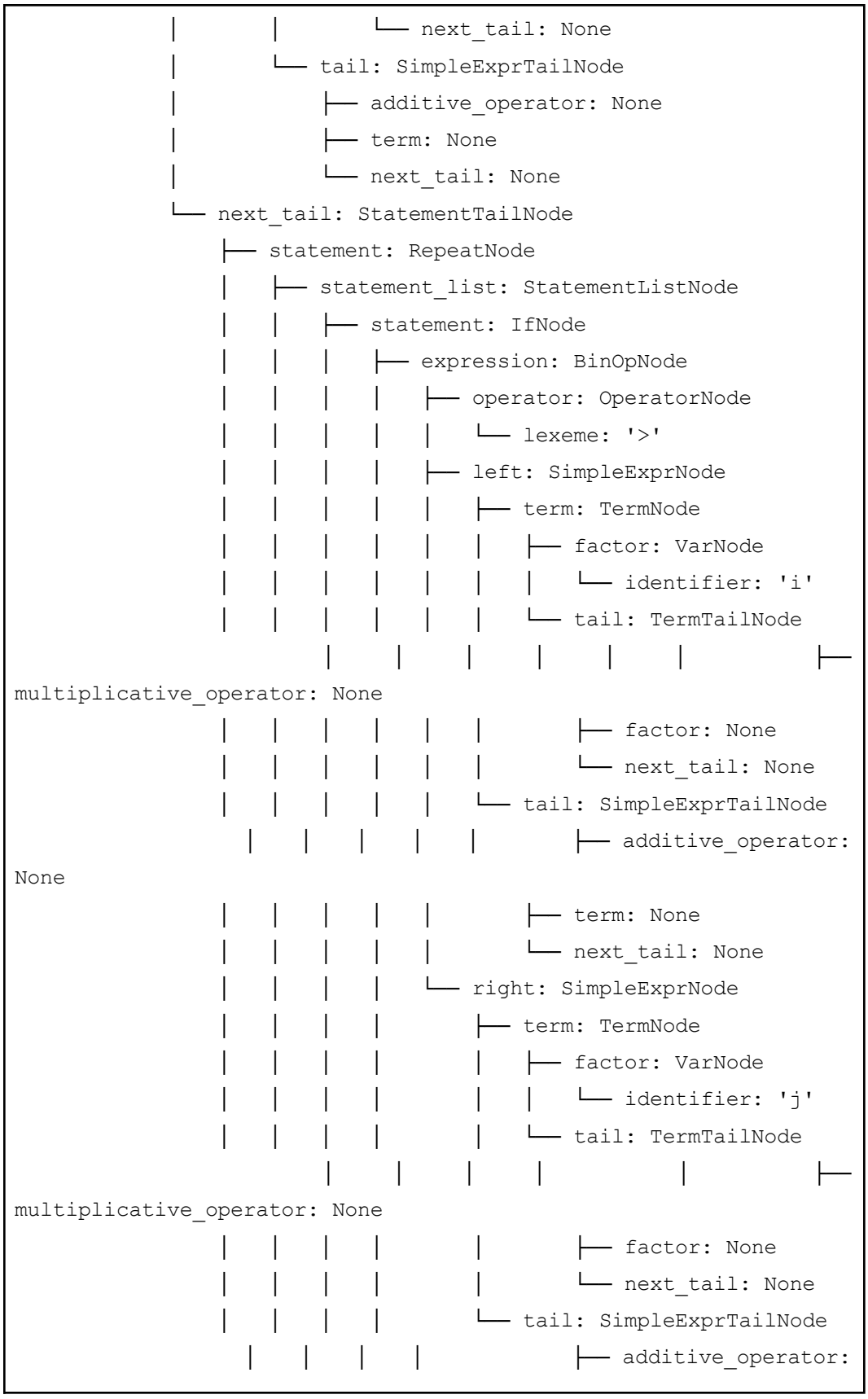

additive_operator: None
term: None
next_tail: None
next_tail: StatementTailNode
statement: None
next_tail: None
next_section: SubprogramSectionNode
subprogram_declaration: None
next_section: None
compound_statement: CompoundNode
statement_list: StatementListNode
statement: AssignNode
var_node: VarNode
identifier: 'p'
field_access_node: None
expression: SimpleExprNode
term: TermNode
factor: CallNode
identifier: 'null'
parameter_list: ParameterListNode
param_group_node: None
expression_node: SimpleExprNode
term: TermNode
factor: FieldAccessNode
identifier_1: 'h1'

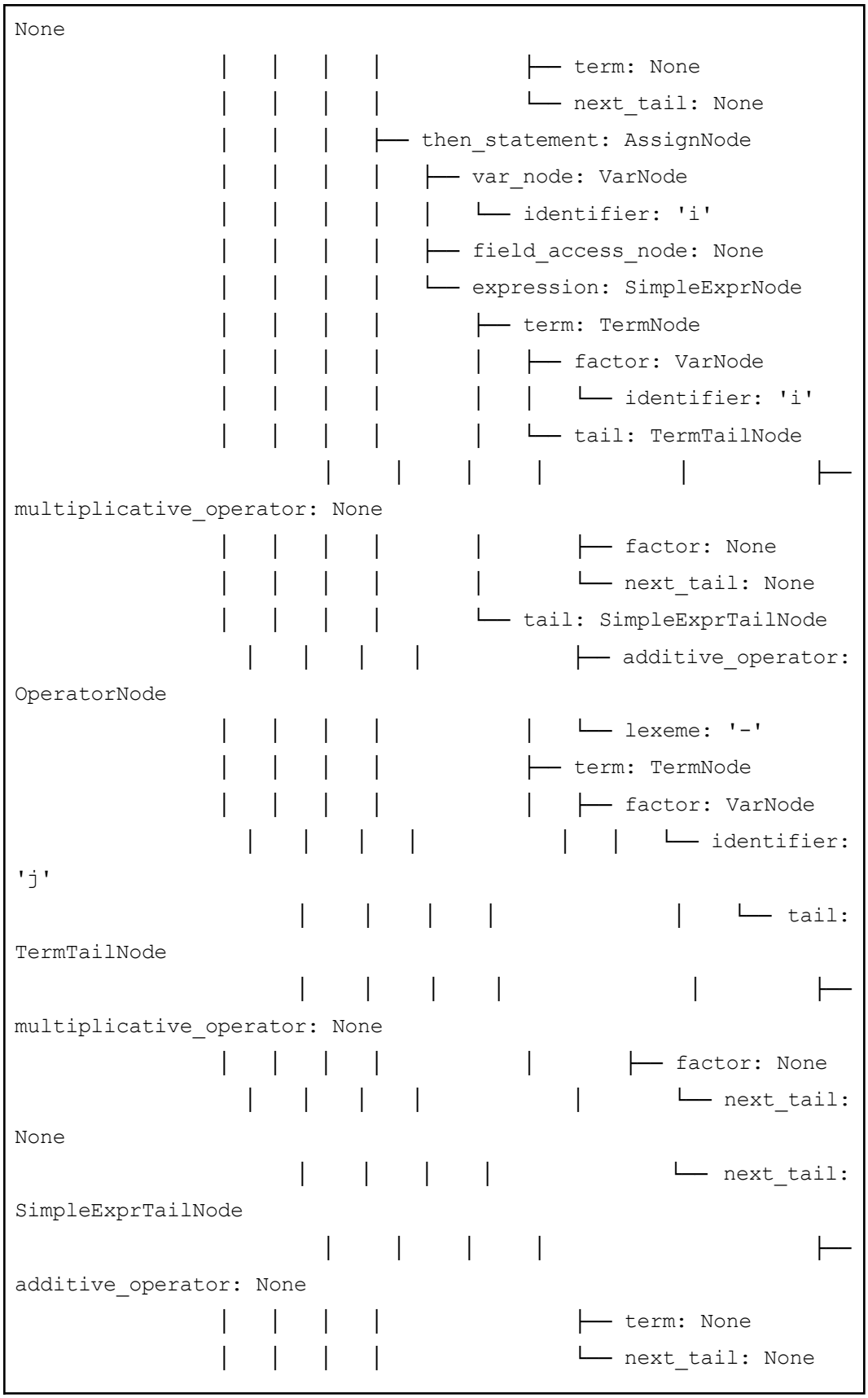

identifier_2: 'c'
└── tail: FieldAccessTailNode
├── identifier: 're'
└── next_tail: FieldAccessTailNode
├── identifier: None
└── next_tail: None
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
└── next_tail: ParameterTailNode
├── param_group_node: None
├── expression_node: SimpleExprNode
├── term: TermNode
├── factor: FieldAccessNode

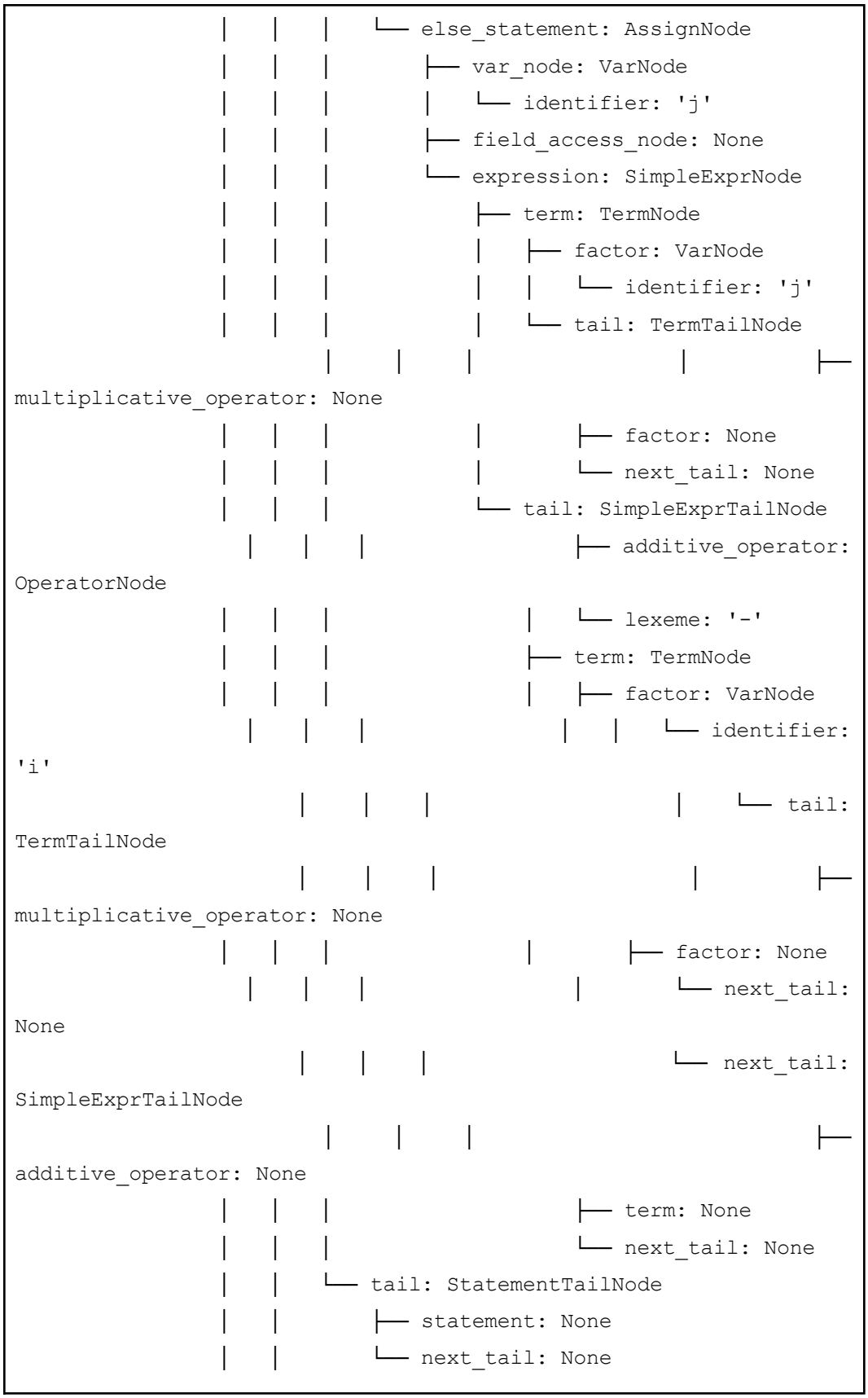

identifier_1: 'h2'
identifier_2: 'c'
tail: FieldAccessTailNode
├── identifier: 'im'
└── next_tail: FieldAccessTailNode
├── identifier: None
└── next_tail: None
└── tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
└── tail: SimpleExprTailNode
additive_operator: None
├── term: None
next_tail: None
└── next_tail: ParameterTailNode
param_group_node: None
expression_node: SimpleExprNode
├── term:


TermNode
factor: VarNode
└── identifier: 'z'
tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
next_tail: ParameterTailNode
param_group_node: None
expression_node: None
next_tail: None
└── tail: TermTailNode
multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: None
├── term: None

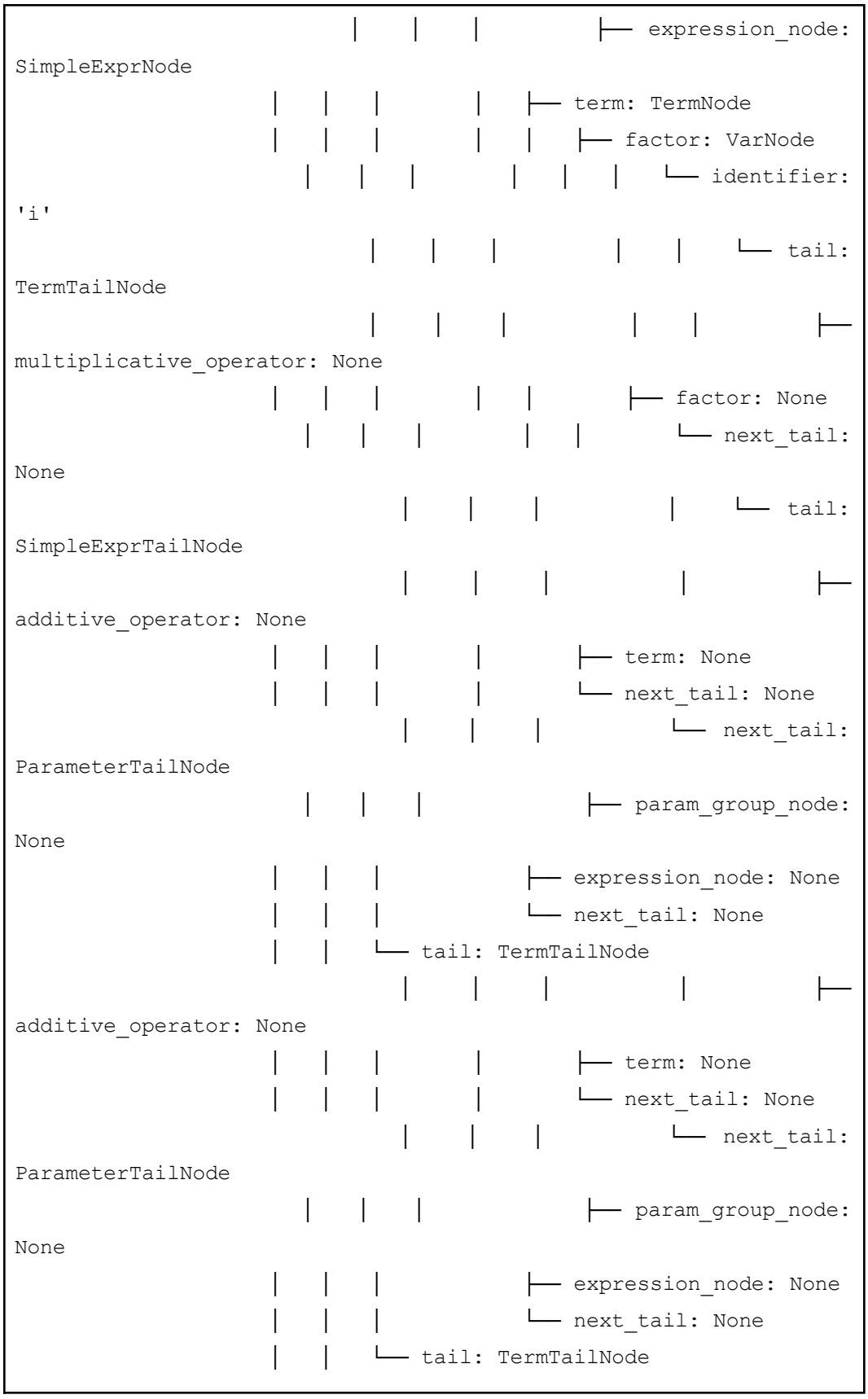

└── next_tail: None
└── tail: StatementTailNode
├── statement: None
└── next_tail: None
└── next_section: SubprogramSectionNode
├── subprogram_declaration: None
└── next_section: None
└── compound_statement: CompoundNode
└── statement_list: StatementListNode
├── statement: AssignNode
├── var_node: VarNode
└── identifier: 'i'
├── field_access_node: None
└── expression: SimpleExprNode
├── term: TermNode
├── factor: NumberNode
└── value: '85'
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
└── tail: StatementTailNode
├── statement: AssignNode
├── var_node: VarNode
└── identifier: 'j'
├── field_access_node: None
└── expression: SimpleExprNode
├── term: TermNode
├── factor: NumberNode
└── value: '51'
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None

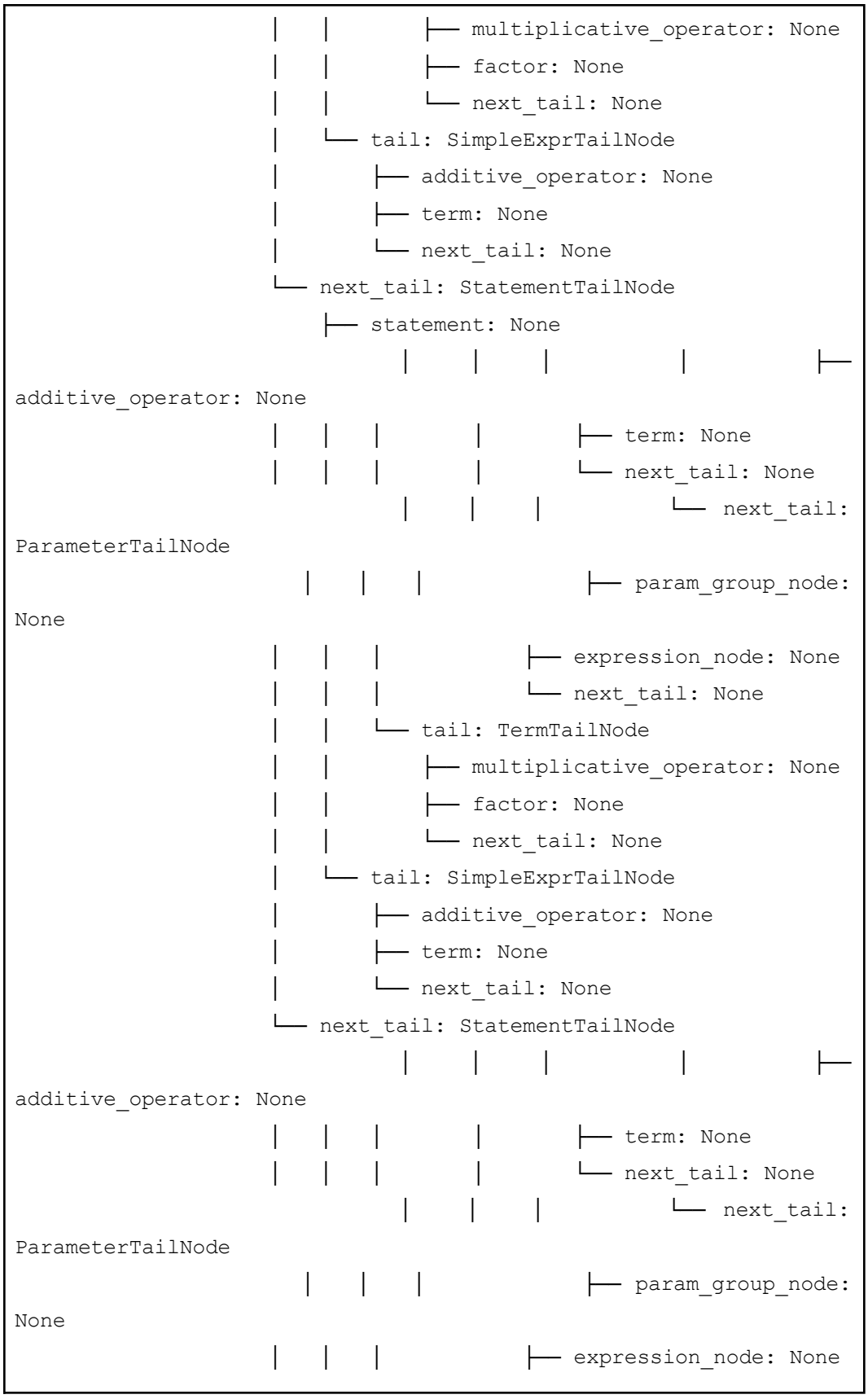

└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
└── next_tail: StatementTailNode
├── statement: RepeatNode
├── statement_list: StatementListNode
├── statement: IfNode
├── expression: BinOpNode
├── operator: OperatorNode
└── lexeme: '>'
├── left: SimpleExprNode
├── term: TermNode
├── factor: VarNode
└── identifier: 'i'
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
└── right: SimpleExprNode
├── term: TermNode
├── factor: VarNode
└── identifier: 'j'
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator:


None
| | | | ├── term: None
| | | | └── next_tail: None
| | | ├── then_statement: AssignNode
| | | | ├── var_node: VarNode
| | | | | └── identifier: 'i'
| | | | ├── field_access_node: None
| | | | └── expression: SimpleExprNode
| | | | ├── term: TermNode
| | | | | ├── factor: VarNode
| | | | | | └── identifier: 'i'
| | | | | └── tail: TermTailNode
| | | | | ├──
multiplicative_operator: None
| | | | | ├── factor: None
| | | | | └── next_tail: None
| | | | └── tail: SimpleExprTailNode
| | | | ├── additive_operator:
OperatorNode
| | | | | └── lexeme: '-'
| | | | ├── term: TermNode
| | | | | ├── factor: VarNode
| | | | | | └── identifier:
'j'
| | | | | └── tail:
TermTailNode
| | | | | ├──
multiplicative_operator: None
| | | | ├── factor: None
| | | | | └── next_tail:
None
| | | | └── next_tail:
SimpleExprTailNode
| | | | ├──
additive_operator: None
| | | | ├── term: None
| | | | └── next_tail: None


└── else_statement: AssignNode
├── var_node: VarNode
│ └── identifier: 'j'
├── field_access_node: None
└── expression: SimpleExprNode
├── term: TermNode
│ ├── factor: VarNode
│ │ └── identifier: 'j'
│ └── tail: TermTailNode
multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator:
OperatorNode
│ └── lexeme: '-'
├── term: TermNode
│ ├── factor: VarNode
│ │ └── identifier:
'i'
│ └── tail:
TermTailNode
multiplicative_operator: None
├── factor: None
└── next_tail:
None
└── next_tail:
SimpleExprTailNode
additive_operator: None
├── term: None
└── next_tail: None
└── tail: StatementTailNode
├── statement: None
└── next_tail: None

```
                │   └── expression: BinOpNode
                │       ├── operator: OperatorNode
                │       │   └── lexeme: '='
                │       ├── left: SimpleExprNode
                │       │   ├── term: TermNode
                │       │   │   ├── factor: VarNode
                │       │   │   │   └── identifier: 'i'
                │       │   │   └── tail: TermTailNode
                  │       │   │       ├── multiplicative_operator:
None
                │       │   │       ├── factor: None
                │       │   │       └── next_tail: None
                │       │   └── tail: SimpleExprTailNode
                │       │       ├── additive_operator: None
                │       │       ├── term: None
                │       │       └── next_tail: None
                │       └── right: SimpleExprNode
                │           ├── term: TermNode
                │           │   ├── factor: VarNode
                │           │   │   └── identifier: 'j'
                │           │   └── tail: TermTailNode
                  │           │       ├── multiplicative_operator:
None
                │           │       ├── factor: None
                │           │       └── next_tail: None
                │           └── tail: SimpleExprTailNode
                │               ├── additive_operator: None
                │               ├── term: None
                │               └── next_tail: None
                └── next_tail: StatementTailNode
                    ├── statement: SimpleExprNode
                    │   ├── term: TermNode
                    │   │   ├── factor: CallNode
                    │   │   │   ├── identifier: 'writeln'
                              │   │   │   └── parameter_list:
ParameterListNode
                    │   │   │       ├── param_group_node: None
```


| | | ├── expression_node:
SimpleExprNode
| | | | ├── term: TermNode
| | | | | ├── factor: VarNode
| | | | | | | └── identifier:
'i'
| | | | | | └── tail:
TermTailNode
| | | | | | ├──
multiplicative_operator: None
| | | | | ├── factor: None
| | | | | └── next_tail:
None
| | | | └── tail:
SimpleExprTailNode
| | | | ├──
additive_operator: None
| | | | ├── term: None
| | | | └── next_tail: None
| | | └── next_tail:
ParameterTailNode
| | | ├── param_group_node:
None
| | | ├── expression_node: None
| | | └── next_tail: None
| | └── tail: TermTailNode
| | | ├──
additive_operator: None
| | | ├── term: None
| | | └── next_tail: None
| | | └── next_tail:
ParameterTailNode
| | | ├── param_group_node:
None
| | | ├── expression_node: None
| | | └── next_tail: None
| | └── tail: TermTailNode

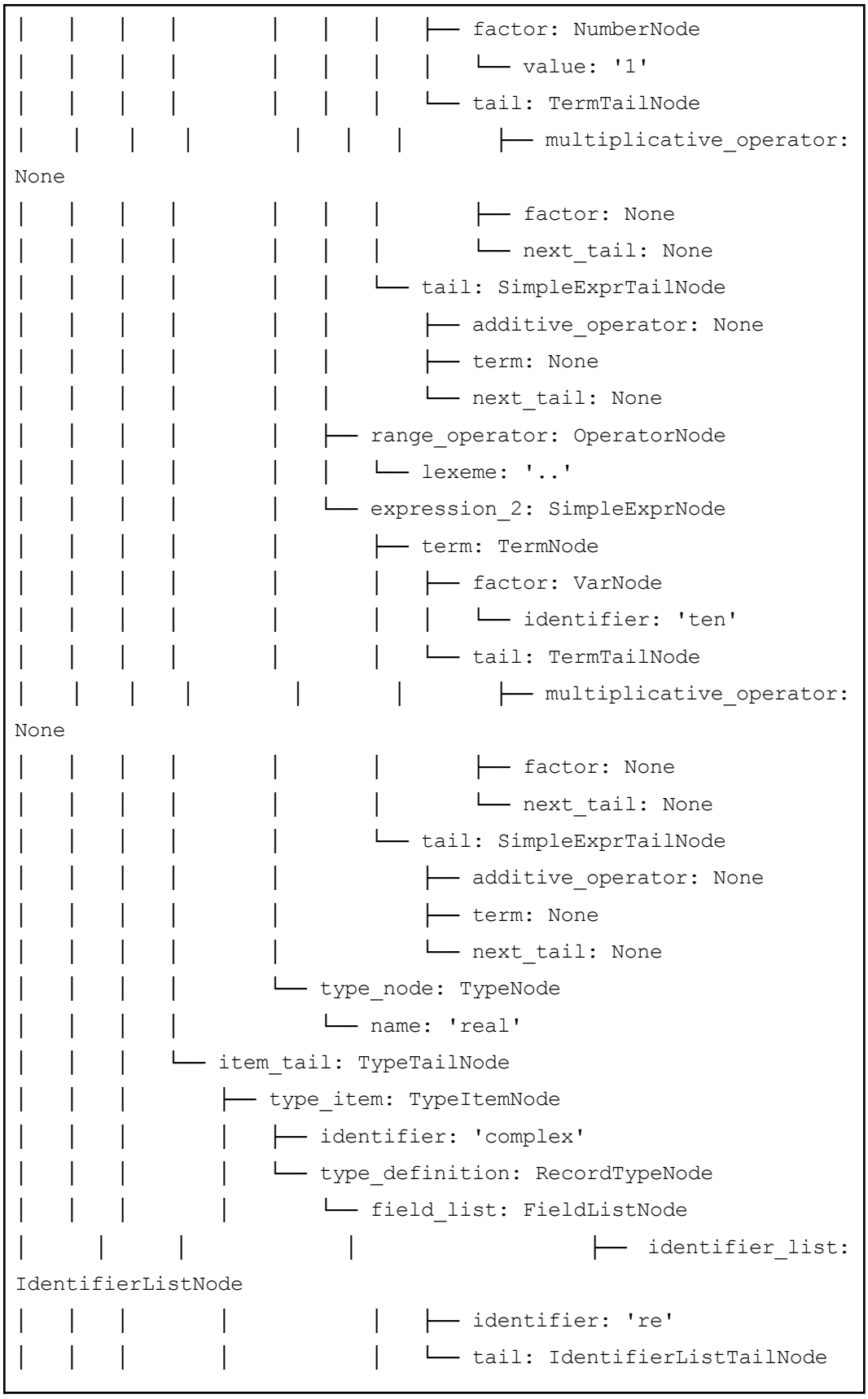

| | ├── multiplicative_operator: None
| | ├── factor: None
| | └── next_tail: None
| └── tail: SimpleExprTailNode
| ├── additive_operator: None
| ├── term: None
| └── next_tail: None
└── next_tail: StatementTailNode
├── statement: None
| | | | ├──
additive_operator: None
| | | | ├── term: None
| | | | └── next_tail: None
| | | └── next_tail:
ParameterTailNode
| | | ├── param_group_node:
None
| | | ├── expression_node: None
| | | └── next_tail: None
| | └── tail: TermTailNode
| | ├── multiplicative_operator: None
| | ├── factor: None
| | └── next_tail: None
| └── tail: SimpleExprTailNode
| ├── additive_operator: None
| ├── term: None
| └── next_tail: None
└── next_tail: StatementTailNode
| | | ├──
additive_operator: None
| | | ├── term: None
| | | └── next_tail: None
| | └── next_tail:
ParameterTailNode
| | ├── param_group_node:
None
| | ├── expression_node: None

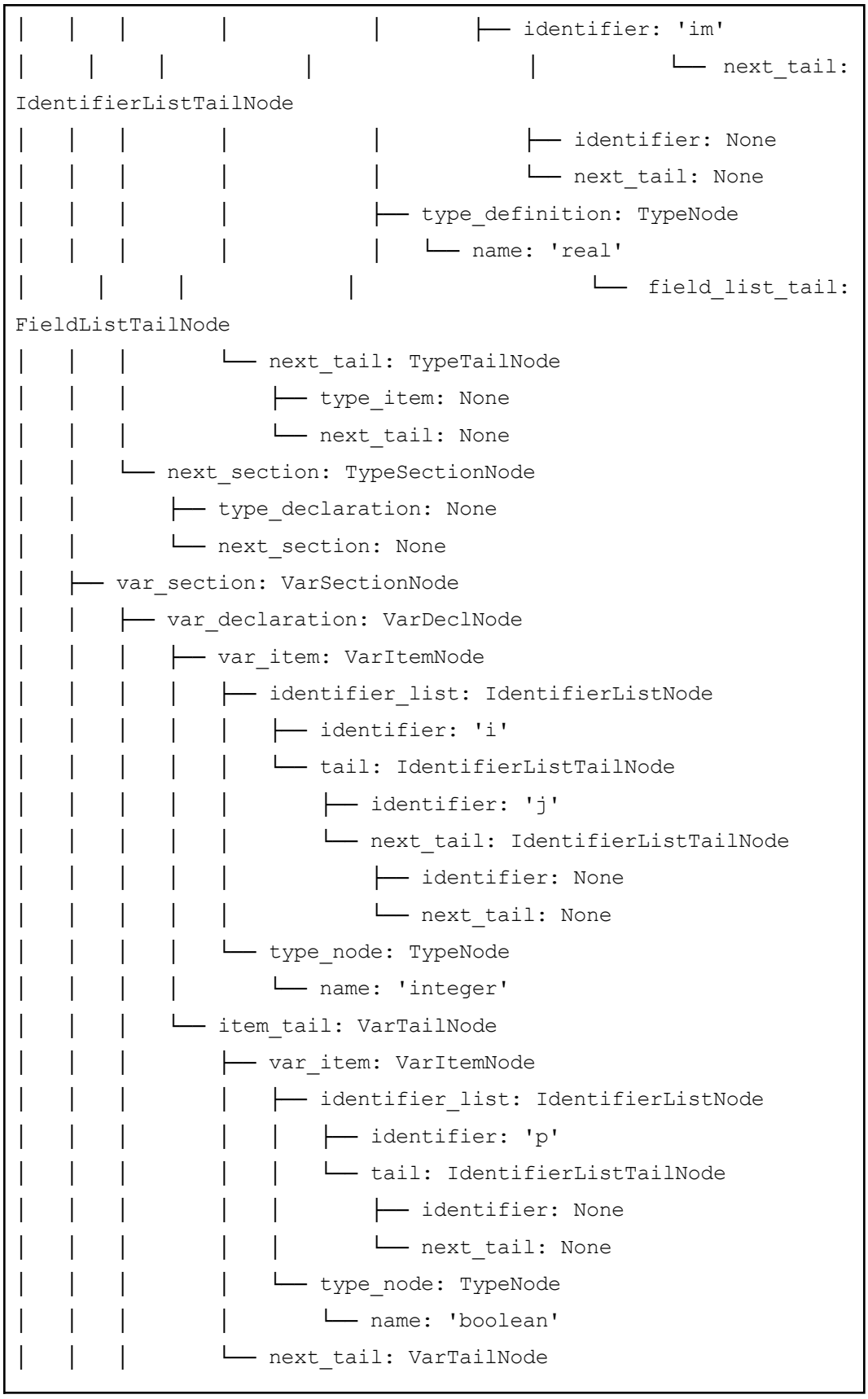
```
                    │   │   │           └── next_tail: None
                    │   │   └── tail: TermTailNode
                    │   │       ├── multiplicative_operator: None
                    │   │   │   │       └── next_tail: None
                              │   │   │           └── next_tail:
ParameterTailNode
                      │   │   │           ├── param_group_node:
None
                    │   │   │           ├── expression_node: None
                    │   │   │           └── next_tail: None
                    │   │   └── tail: TermTailNode
                      │   │   │           ├── param_group_node:
None
                    │   │   │           ├── expression_node: None
                    │   │   │           └── next_tail: None
                    │   │   └── tail: TermTailNode
                    │   │   │           └── next_tail: None
                    │   │   └── tail: TermTailNode
                    │   │   └── tail: TermTailNode
                    │   │       ├── multiplicative_operator: None
                    │   │       ├── factor: None
                    │   │       ├── factor: None
                    │   │       └── next_tail: None
                    │   └── tail: SimpleExprTailNode
                    │       ├── additive_operator: None
                    │       ├── term: None
                    │       └── next_tail: None
                    └── next_tail: StatementTailNode
                        ├── statement: None
                        └── next_tail: None


================================================
Tahap: Semantic Analysis (Symbol Tables)
================================================
Semantic Error: Duplicate declaration of 'i' in scope level 1
```

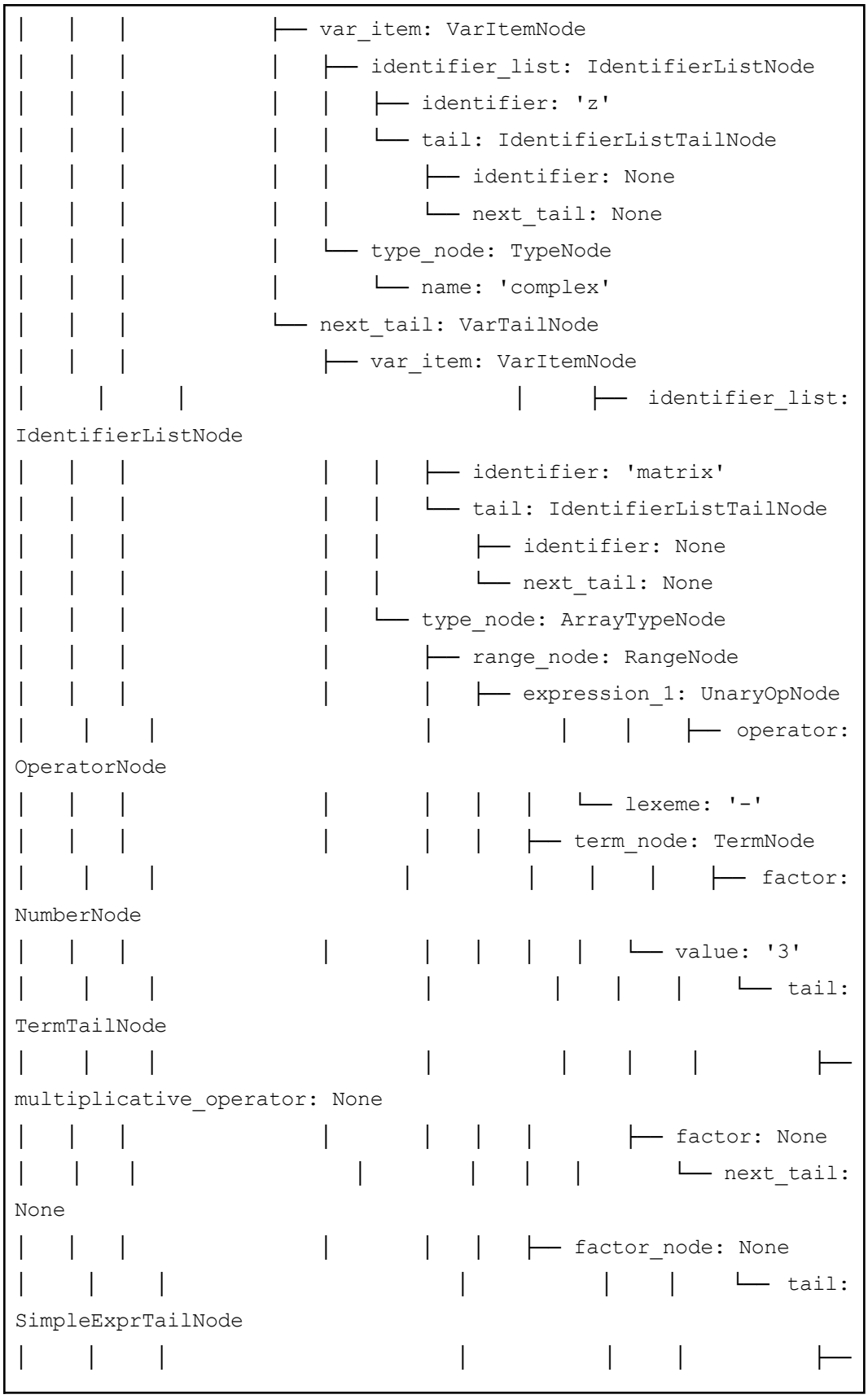
Pembahasan:

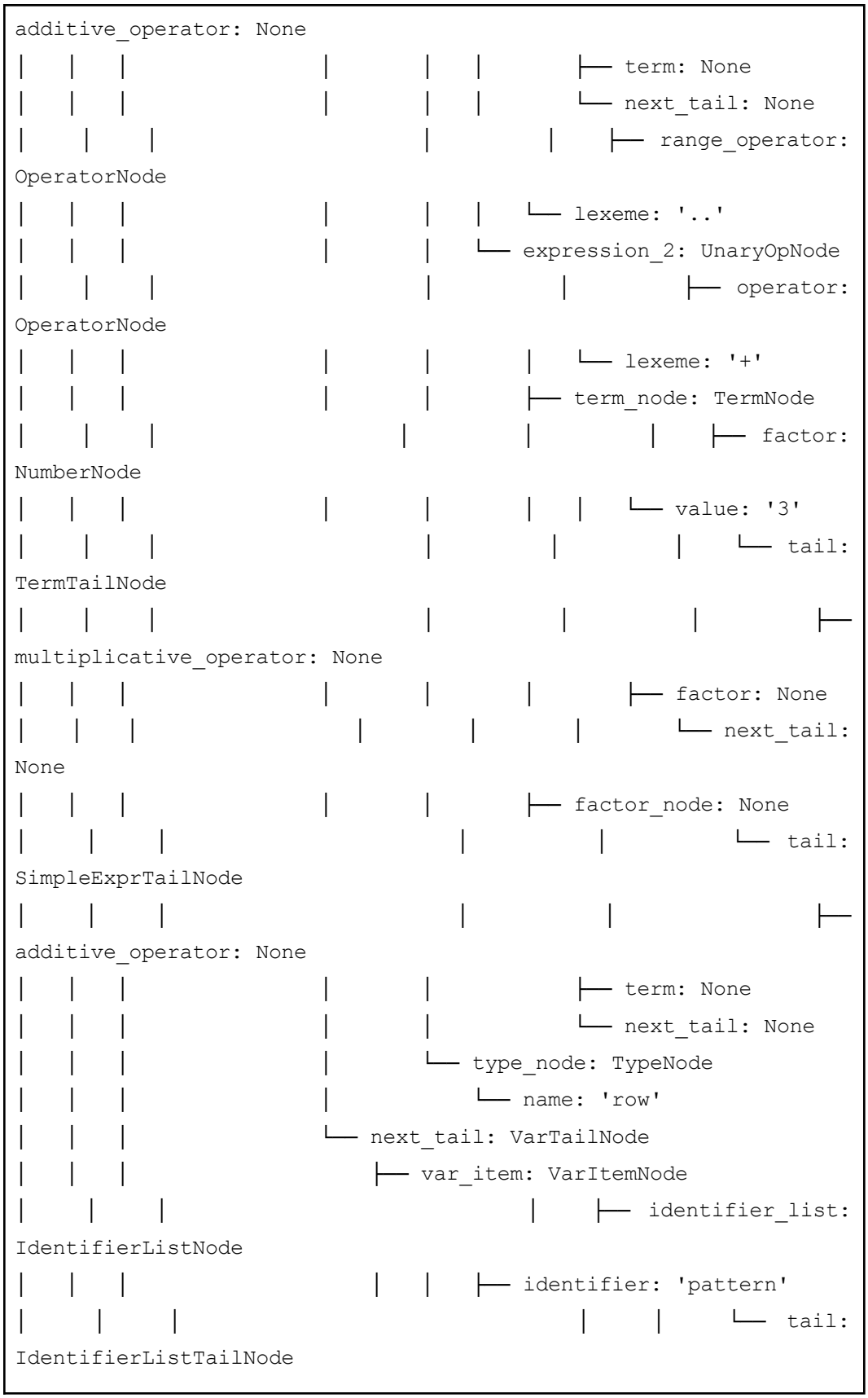
3.2.3 Hasil Uji 3

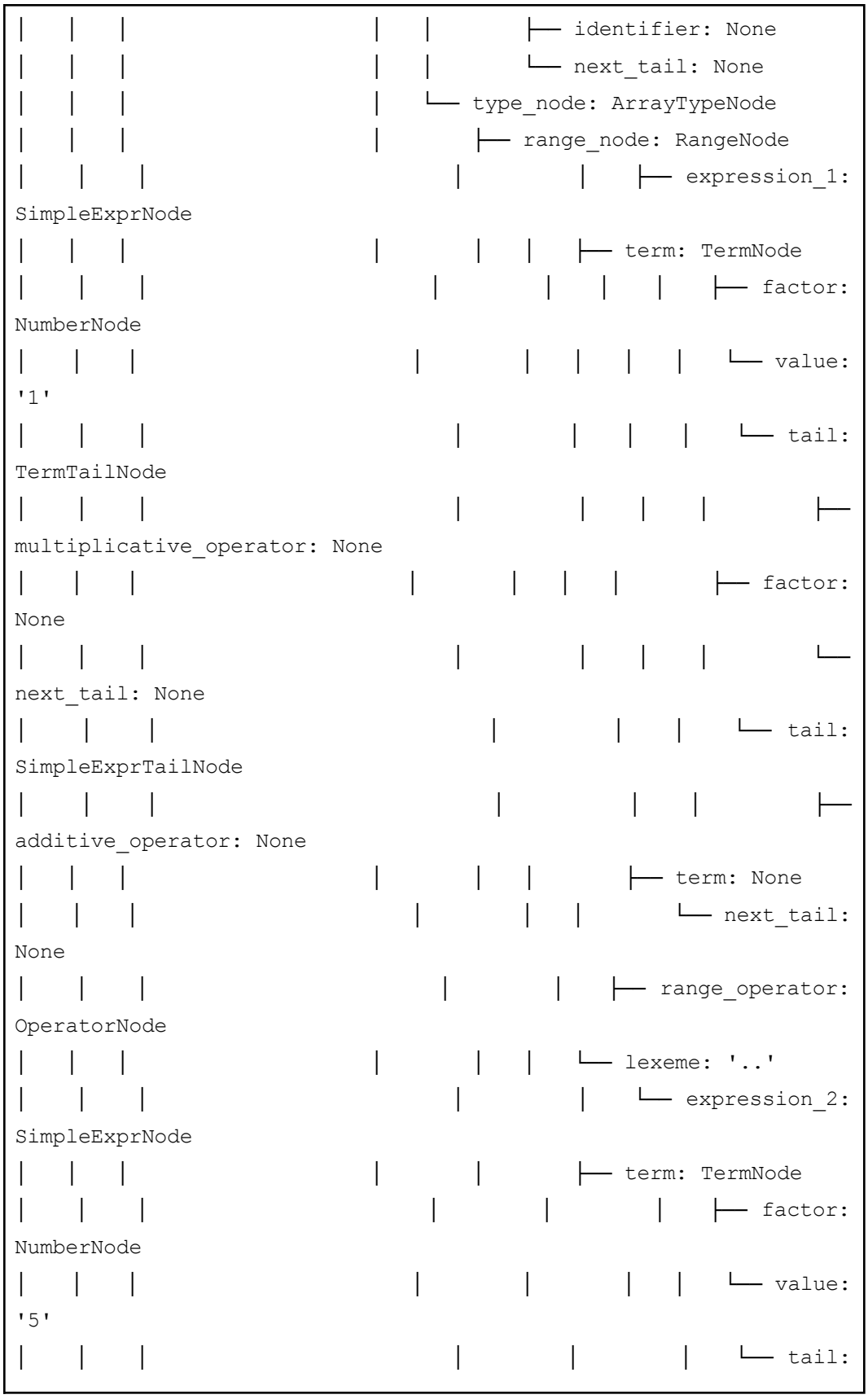
```
==================================================
Tahap: AST Generation (Abstract Syntax Tree)
==================================================
AST Structure:
------------------------------
ProgramNode
├── program_header: ProgramHeaderNode
│   └── identifier: 'test0'
├── declaration_part: DeclarationPartNode
│   ├── const_section: ConstSectionNode
│   │   ├── const_declaration: ConstDeclNode
│   │   │   ├── const_item: ConstItemNode
│   │   │   │   ├── identifier: 'ten'
│   │   │   │   ├── assign_operator: '='
│   │   │   │   └── value_node: NumberNode
│   │   │   │       └── value: '10'
│   │   │   └── item_tail: ConstTailNode
│   │   │       ├── const_item: ConstItemNode
│   │   │       │   ├── identifier: 'plus'
│   │   │       │   ├── assign_operator: '='
│   │   │       │   └── value_node: StringNode
│   │   │       │       └── value: ''+''
│   │   │       └── next_tail: ConstTailNode
│   │   │           ├── const_item: None
│   │   │           └── next_tail: None
│   │   └── next_section: ConstSectionNode
│   │       ├── const_declaration: None
│   │       └── next_section: None
│   ├── type_section: TypeSectionNode
│   │   ├── type_declaration: TypeDeclNode
│   │   │   ├── type_item: TypeItemNode
│   │   │   │   ├── identifier: 'row'
│   │   │   │   └── type_definition: ArrayTypeNode
│   │   │   │       ├── range_node: RangeNode
│   │   │   │       │   ├── expression_1: SimpleExprNode
│   │   │   │       │   │   ├── term: TermNode
```

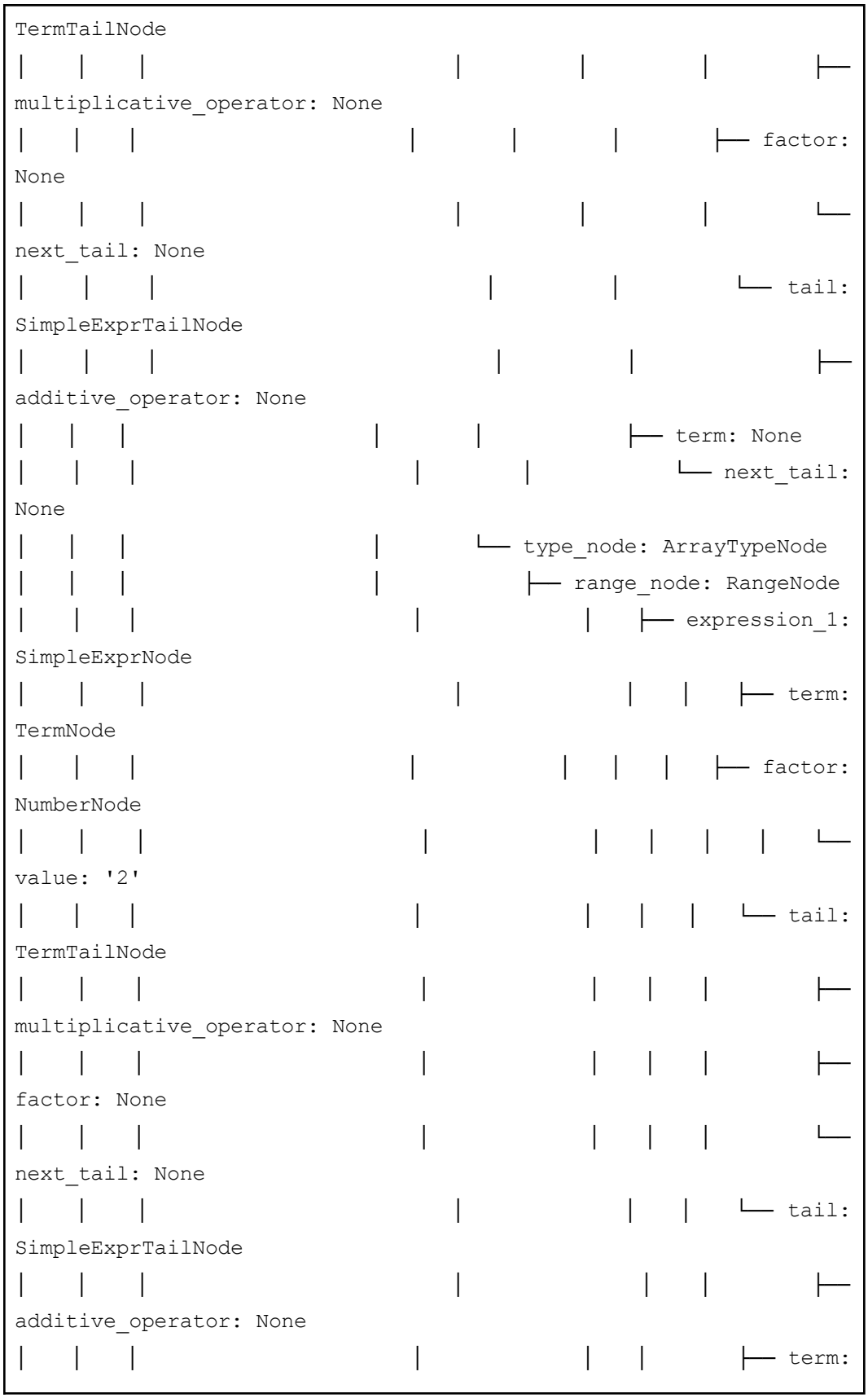

| | | | | | | | ├── factor: NumberNode
| | | | | | | | | └── value: '1'
| | | | | | | | └── tail: TermTailNode
| | | | | | | | ├── multiplicative_operator:
None
| | | | | | | | ├── factor: None
| | | | | | | | └── next_tail: None
| | | | | | | └── tail: SimpleExprTailNode
| | | | | | | ├── additive_operator: None
| | | | | | | ├── term: None
| | | | | | | └── next_tail: None
| | | | | | ├── range_operator: OperatorNode
| | | | | | | └── lexeme: '..'
| | | | | | └── expression_2: SimpleExprNode
| | | | | | ├── term: TermNode
| | | | | | | ├── factor: VarNode
| | | | | | | | └── identifier: 'ten'
| | | | | | | └── tail: TermTailNode
| | | | | | | ├── multiplicative_operator:
None
| | | | | | | ├── factor: None
| | | | | | | └── next_tail: None
| | | | | | └── tail: SimpleExprTailNode
| | | | | | ├── additive_operator: None
| | | | | | ├── term: None
| | | | | | └── next_tail: None
| | | | | └── type_node: TypeNode
| | | | | └── name: 'real'
| | | | └── item_tail: TypeTailNode
| | | | ├── type_item: TypeItemNode
| | | | | ├── identifier: 'complex'
| | | | | └── type_definition: RecordTypeNode
| | | | | └── field_list: FieldListNode
| | | | | ├── identifier_list:
IdentifierListNode
| | | | | | ├── identifier: 're'
| | | | | | └── tail: IdentifierListTailNode

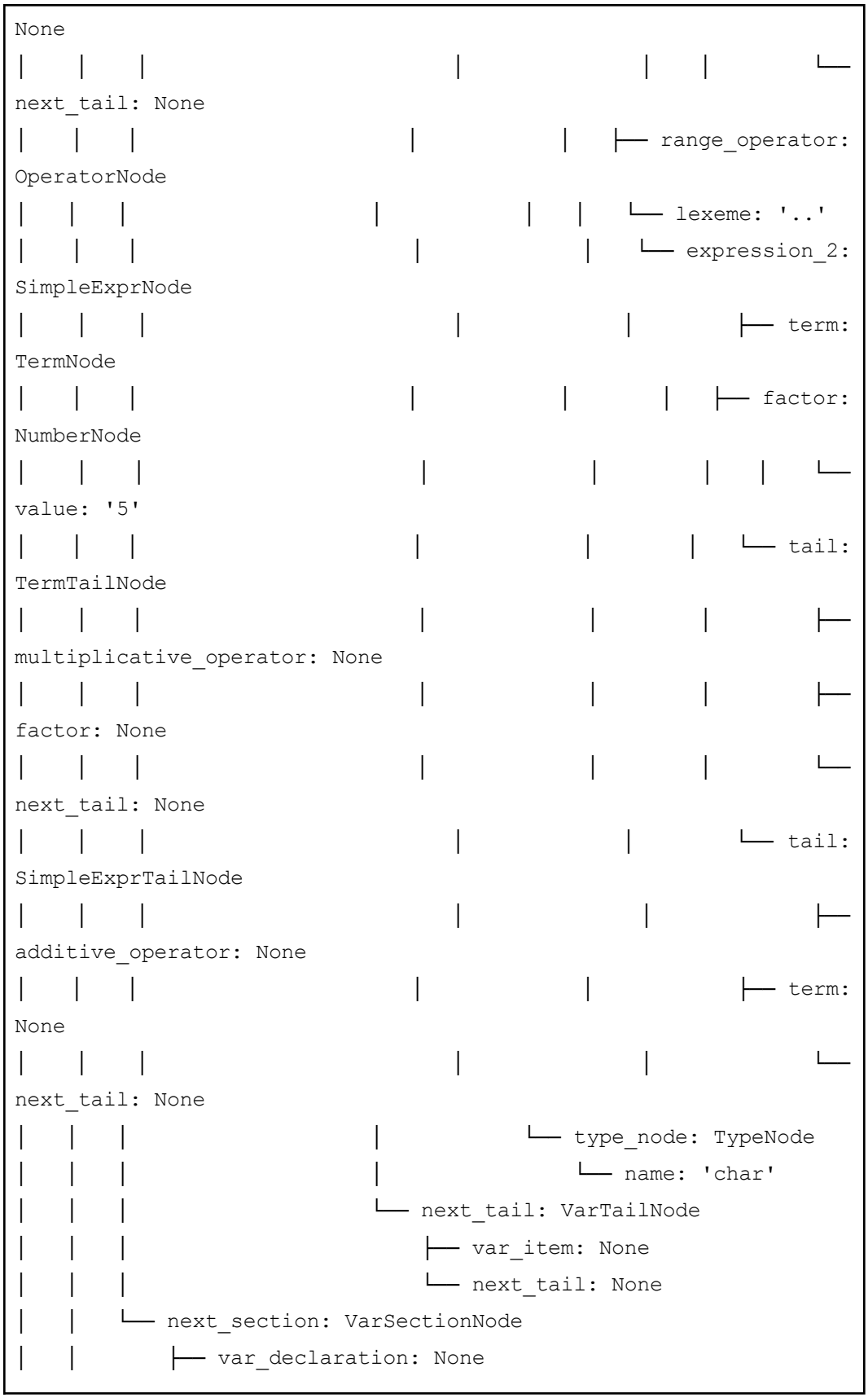

| | | | | | ├── identifier: 'im'
| | | | | | └── next_tail:
IdentifierListTailNode
| | | | | | ├── identifier: None
| | | | | | └── next_tail: None
| | | | | ├── type_definition: TypeNode
| | | | | | └── name: 'real'
| | | | | └── field_list_tail:
FieldListTailNode
| | | | └── next_tail: TypeTailNode
| | | | ├── type_item: None
| | | | └── next_tail: None
| | | └── next_section: TypeSectionNode
| | | ├── type_declaration: None
| | | └── next_section: None
| | ├── var_section: VarSectionNode
| | | ├── var_declaration: VarDeclNode
| | | | ├── var_item: VarItemNode
| | | | | ├── identifier_list: IdentifierListNode
| | | | | | ├── identifier: 'i'
| | | | | | └── tail: IdentifierListTailNode
| | | | | | ├── identifier: 'j'
| | | | | | └── next_tail: IdentifierListTailNode
| | | | | | ├── identifier: None
| | | | | | └── next_tail: None
| | | | | └── type_node: TypeNode
| | | | | └── name: 'integer'
| | | | └── item_tail: VarTailNode
| | | | ├── var_item: VarItemNode
| | | | | ├── identifier_list: IdentifierListNode
| | | | | | ├── identifier: 'p'
| | | | | | └── tail: IdentifierListTailNode
| | | | | | ├── identifier: None
| | | | | | └── next_tail: None
| | | | | └── type_node: TypeNode
| | | | | └── name: 'boolean'
| | | | └── next_tail: VarTailNode


├── var_item: VarItemNode
│   ├── identifier_list: IdentifierListNode
│   │   ├── identifier: 'z'
│   │   └── tail: IdentifierListTailNode
│   │   ├── identifier: None
│   │   └── next_tail: None
│   └── type_node: TypeNode
│   └── name: 'complex'
└── next_tail: VarTailNode
    ├── var_item: VarItemNode
    │   ├── identifier_list: IdentifierListNode
    │   │   ├── identifier: 'matrix'
    │   │   └── tail: IdentifierListTailNode
    │   │   ├── identifier: None
    │   │   └── next_tail: None
    │   └── type_node: ArrayTypeNode
    │   ├── range_node: RangeNode
    │   │   ├── expression_1: UnaryOpNode
    │   │   │   ├── operator: OperatorNode
    │   │   │   │   └── lexeme: '-'
    │   │   │   ├── term_node: TermNode
    │   │   │   │   ├── factor: NumberNode
    │   │   │   │   │   └── value: '3'
    │   │   │   │   └── tail: TermTailNode
    │   │   │   │   ├── multiplicative_operator: None
    │   │   │   │   ├── factor: None
    │   │   │   │   └── next_tail: None
    │   │   │   └── factor_node: None
    │   │   │   └── tail: SimpleExprTailNode
    │   │   │   ├──

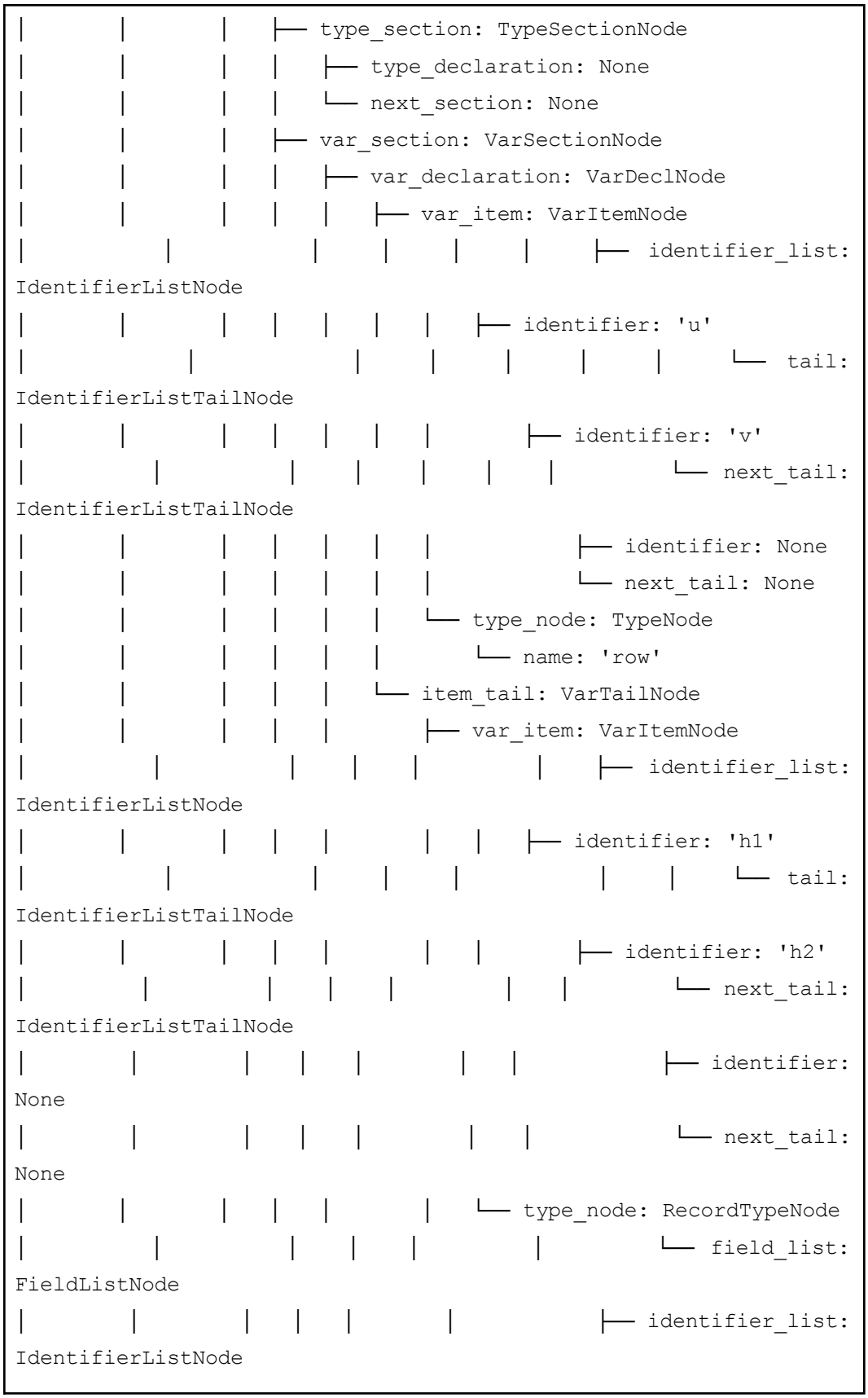

additive_operator: None
term: None
next_tail: None
range_operator: OperatorNode
lexeme: '..'
expression_2: UnaryOpNode
operator: OperatorNode
lexeme: '+'
term_node: TermNode
factor: NumberNode
value: '3'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
factor_node: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
type_node: TypeNode
name: 'row'
next_tail: VarTailNode
var_item: VarItemNode
identifier_list: IdentifierListNode
identifier: 'pattern'
tail: IdentifierListTailNode

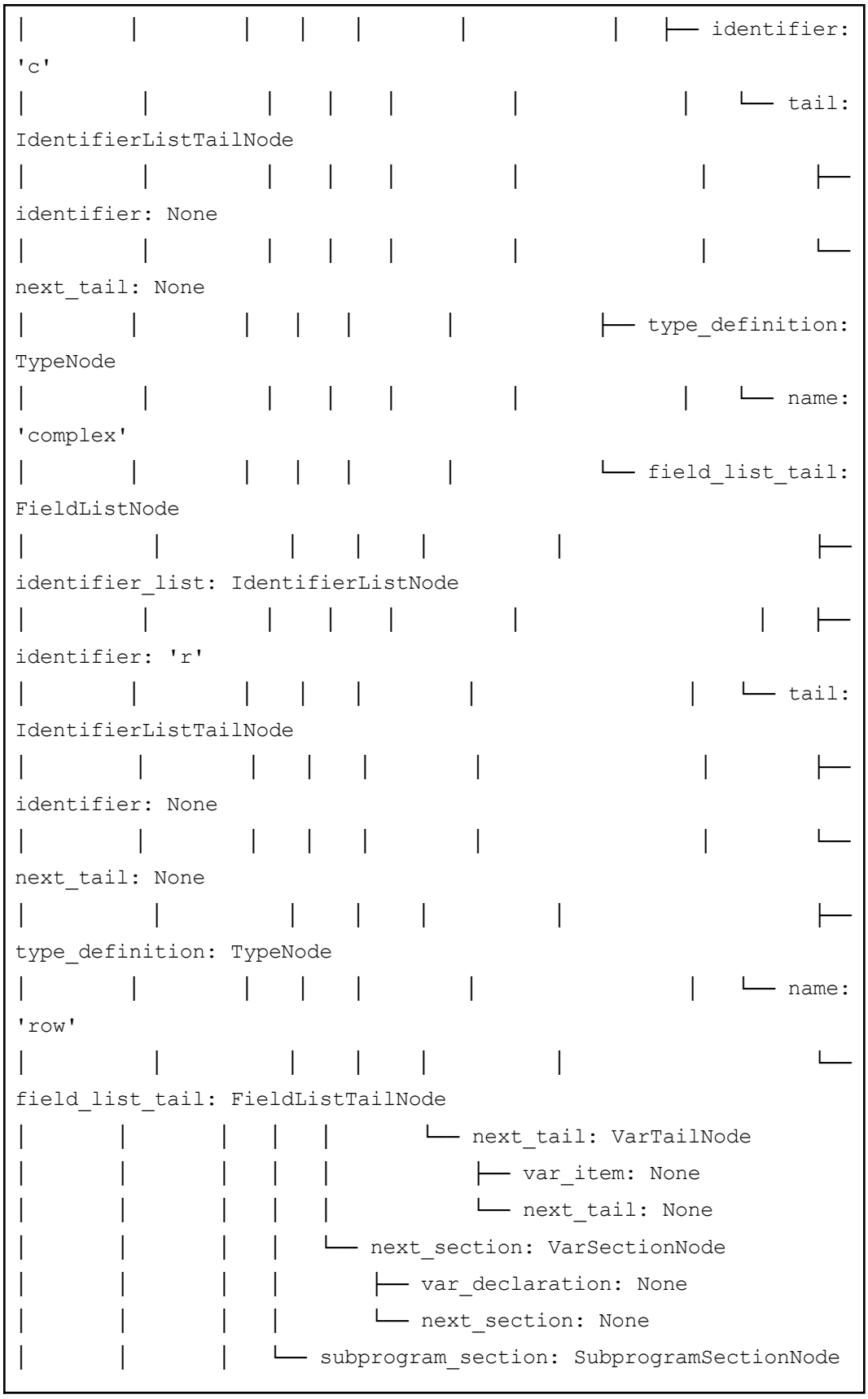

| | | | | | ├── identifier: None
| | | | | | └── next_tail: None
| | | | | └── type_node: ArrayTypeNode
| | | | | ├── range_node: RangeNode
| | | | | | ├── expression_1: SimpleExprNode
| | | | | | | ├── term: TermNode
| | | | | | | | ├── factor: NumberNode
| | | | | | | | | └── value: '1'
| | | | | | | | └── tail: TermTailNode
| | | | | | | | ├── multiplicative_operator: None
| | | | | | | | ├── factor: None
| | | | | | | | └── next_tail: None
| | | | | | | └── tail: SimpleExprTailNode
| | | | | | | ├── additive_operator: None
| | | | | | | ├── term: None
| | | | | | | └── next_tail: None
| | | | | | ├── range_operator: OperatorNode
| | | | | | | └── lexeme: '..'
| | | | | | └── expression_2: SimpleExprNode
| | | | | | ├── term: TermNode
| | | | | | | ├── factor: NumberNode
| | | | | | | | └── value: '5'
| | | | | | | └── tail:

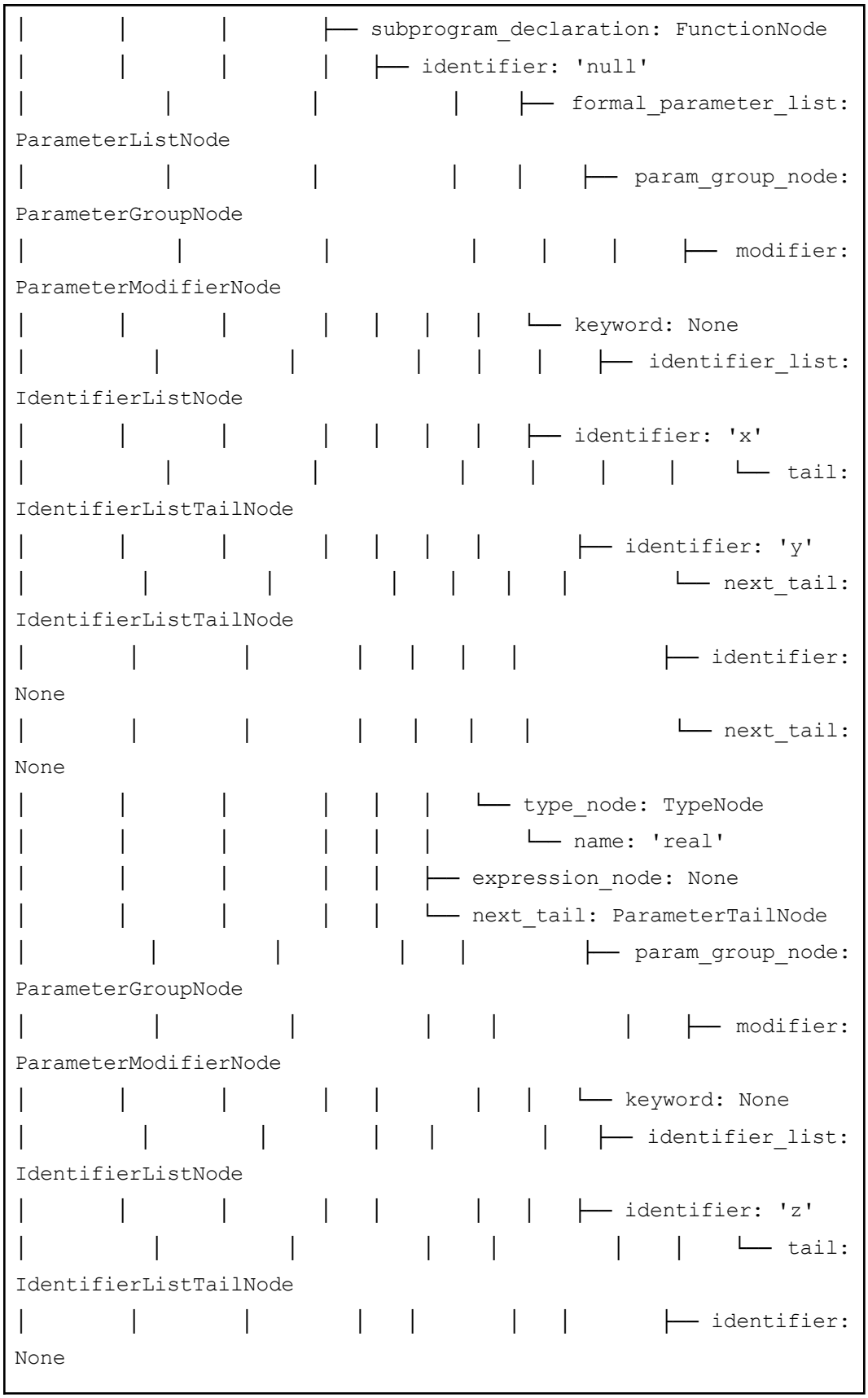

TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
type_node: ArrayTypeNode
range_node: RangeNode
expression_1: SimpleExprNode
term: TermNode
factor: NumberNode
value: '2'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term:

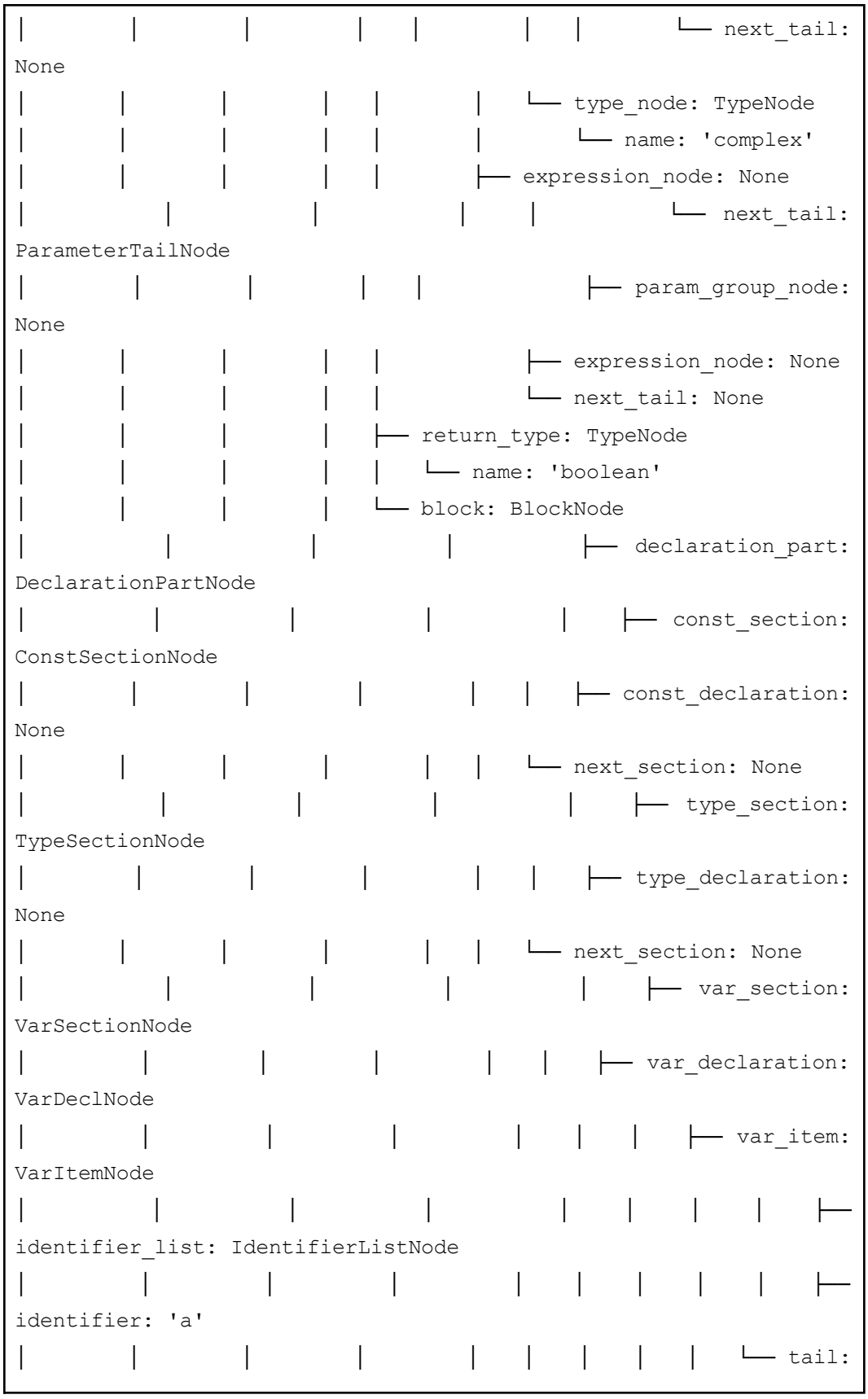

None
next_tail: None
range_operator: OperatorNode
lexeme: '..'
expression_2: SimpleExprNode
term: TermNode
factor: NumberNode
value: '5'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
type_node: TypeNode
name: 'char'
next_tail: VarTailNode
var_item: None
next_tail: None
next_section: VarSectionNode
var_declaration: None

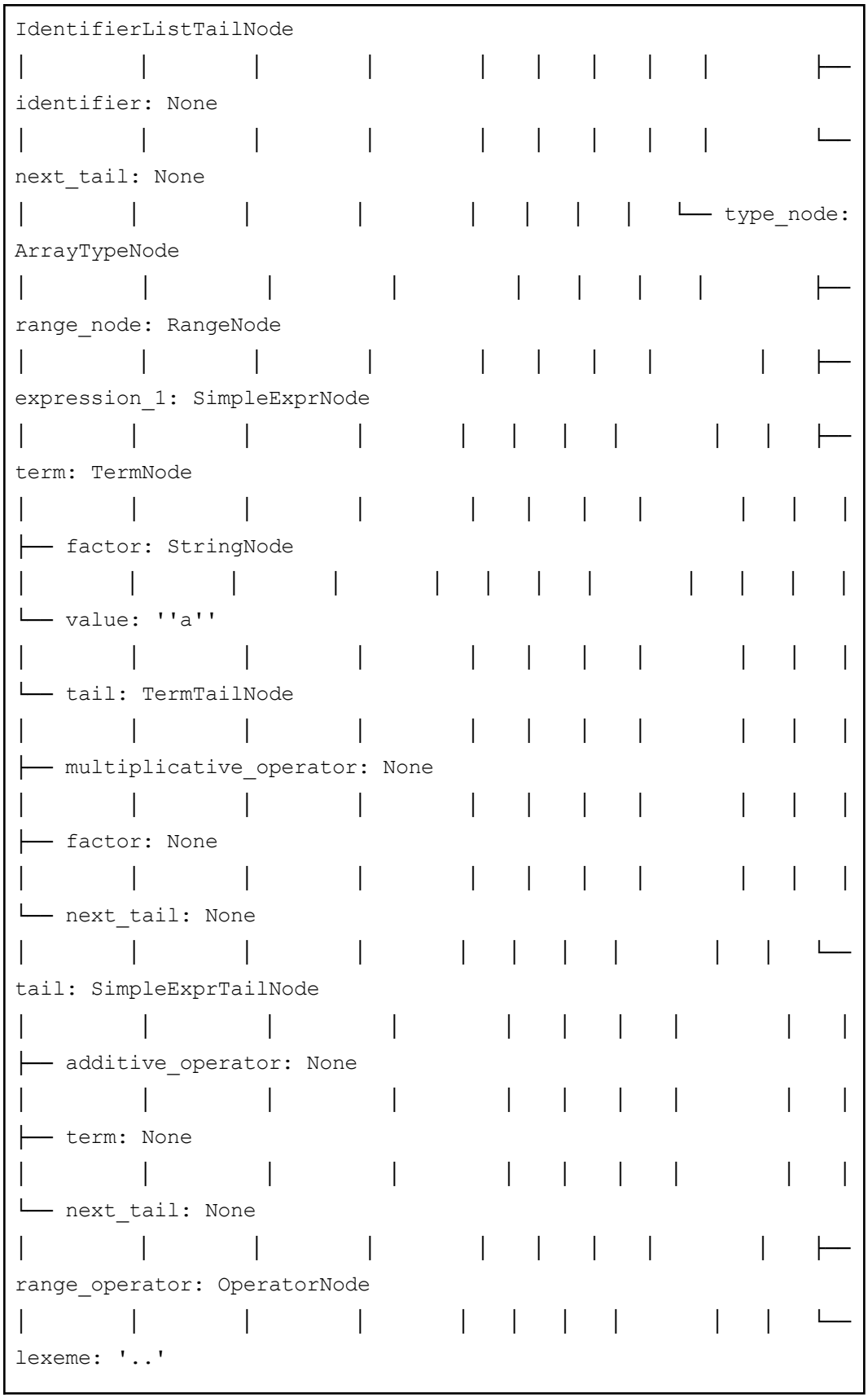
```
|   |       └── next_section: None
|   └── subprogram_section: SubprogramSectionNode
|       ├── subprogram_declaration: ProcedureNode
|       |   ├── identifier: 'dummy'
|       |   ├── formal_parameter_list: ParameterListNode
|       |   |   ├── param_group_node: ParameterGroupNode
|       |   |   |   ├── modifier: ParameterModifierNode
|       |   |   |   |   └── keyword: 'variabel'
|       |   |   |   ├── identifier_list: IdentifierListNode
|       |   |   |   |   ├── identifier: 'i'
|       |   |   |   |   └── tail: IdentifierListTailNode
|       |   |   |   |       ├── identifier: None
|       |   |   |   |       └── next_tail: None
|       |   |   |   └── type_node: TypeNode
|       |   |   |       └── name: 'integer'
|       |   |   ├── expression_node: None
|       |   |   └── next_tail: ParameterTailNode
|       |   |       ├── param_group_node: ParameterGroupNode
|       |   |       |   ├── modifier: ParameterModifierNode
|       |   |       |   |   └── keyword: 'variabel'
|       |   |       |   ├── identifier_list: IdentifierListNode
|       |   |       |   |   ├── identifier: 'z'
|       |   |       |   |   └── tail: IdentifierListTailNode
|       |   |       |   |       ├── identifier: None
|       |   |       |   |       └── next_tail: None
|       |   |       |   └── type_node: TypeNode
|       |   |       |       └── name: 'complex'
|       |   |       ├── expression_node: None
|       |   |       └── next_tail: ParameterTailNode
|       |   |           ├── param_group_node: None
|       |   |           ├── expression_node: None
|       |   |           └── next_tail: None
|       |   └── block: BlockNode
|       |       ├── declaration_part: DeclarationPartNode
|       |       |   ├── const_section: ConstSectionNode
|       |       |   |   ├── const_declaration: None
|       |       |   |   └── next_section: None
```

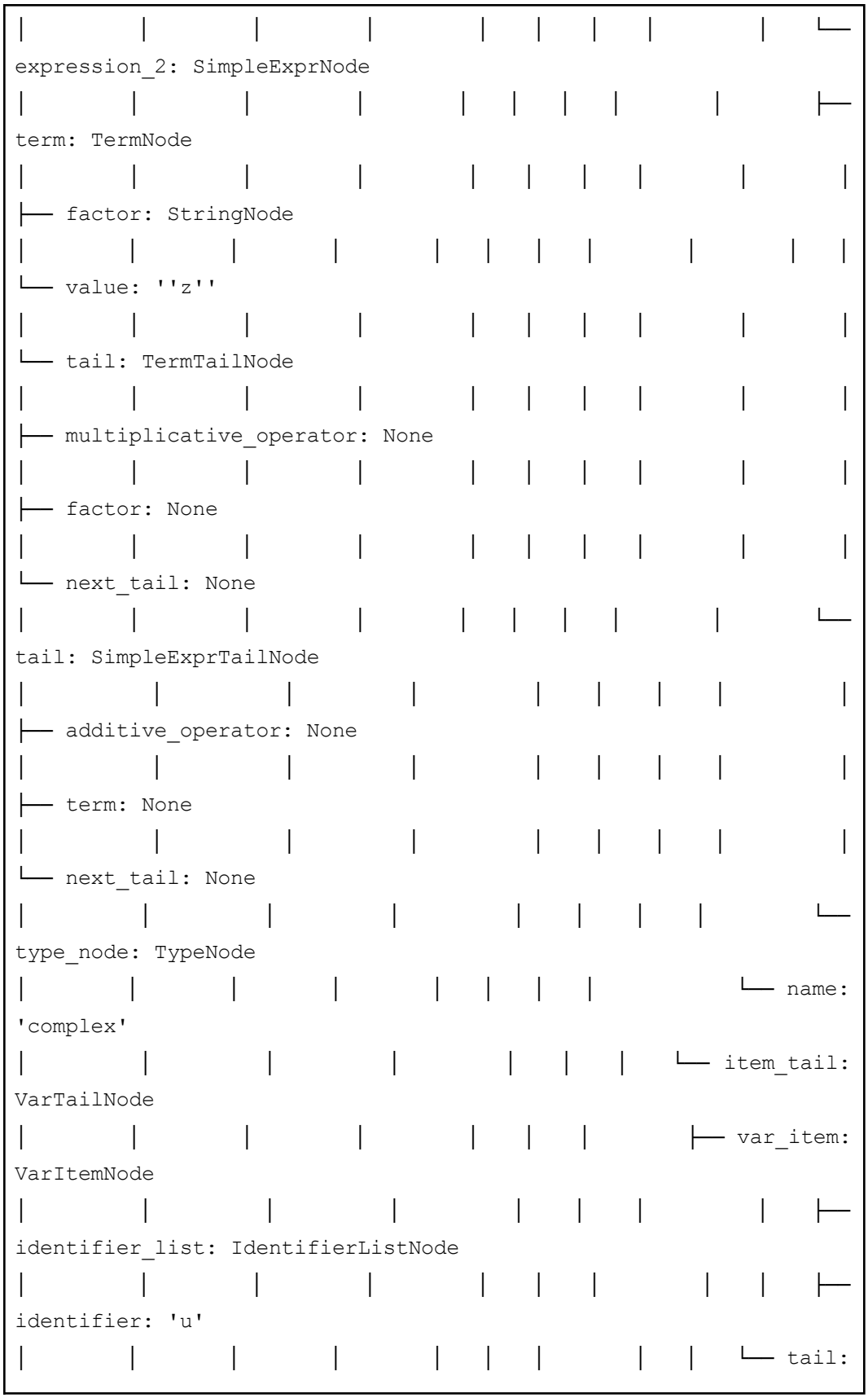

type_section: TypeSectionNode
type_declaration: None
next_section: None
var_section: VarSectionNode
var_declaration: VarDeclNode
var_item: VarItemNode
identifier_list: IdentifierListNode
identifier: 'u'
tail: IdentifierListTailNode
identifier: 'v'
next_tail: IdentifierListTailNode
identifier: None
next_tail: None
type_node: TypeNode
name: 'row'
item_tail: VarTailNode
var_item: VarItemNode
identifier_list: IdentifierListNode
identifier: 'h1'
tail: IdentifierListTailNode
identifier: 'h2'
next_tail: IdentifierListTailNode
identifier: None
next_tail: None
type_node: RecordTypeNode
field_list: FieldListNode
identifier_list: IdentifierListNode

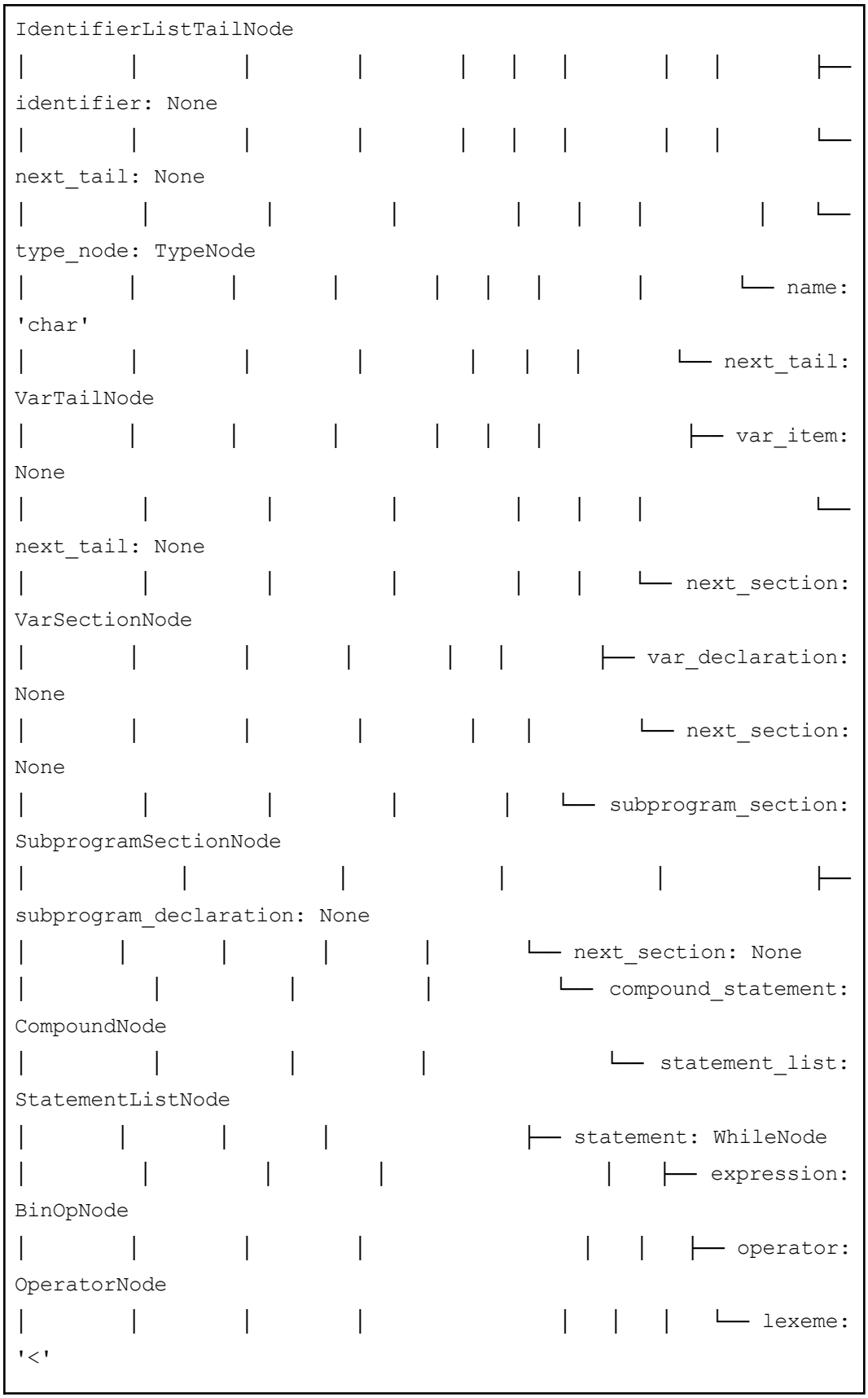

| | | | | | | ├── identifier:
'c'
| | | | | | | └── tail:
IdentifierListTailNode
| | | | | | | ├──
identifier: None
| | | | | | | └──
next_tail: None
| | | | | | ├── type_definition:
TypeNode
| | | | | | | └── name:
'complex'
| | | | | | └── field_list_tail:
FieldListNode
| | | | | | ├──
identifier_list: IdentifierListNode
| | | | | | | ├──
identifier: 'r'
| | | | | | └── tail:
IdentifierListTailNode
| | | | | | ├──
identifier: None
| | | | | | └──
next_tail: None
| | | | | ├──
type_definition: TypeNode
| | | | | └── name:
'row'
| | | | | └──
field_list_tail: FieldListTailNode
| | | | └── next_tail: VarTailNode
| | | | ├── var_item: None
| | | | └── next_tail: None
| | | └── next_section: VarSectionNode
| | | ├── var_declaration: None
| | | └── next_section: None
| | └── subprogram_section: SubprogramSectionNode

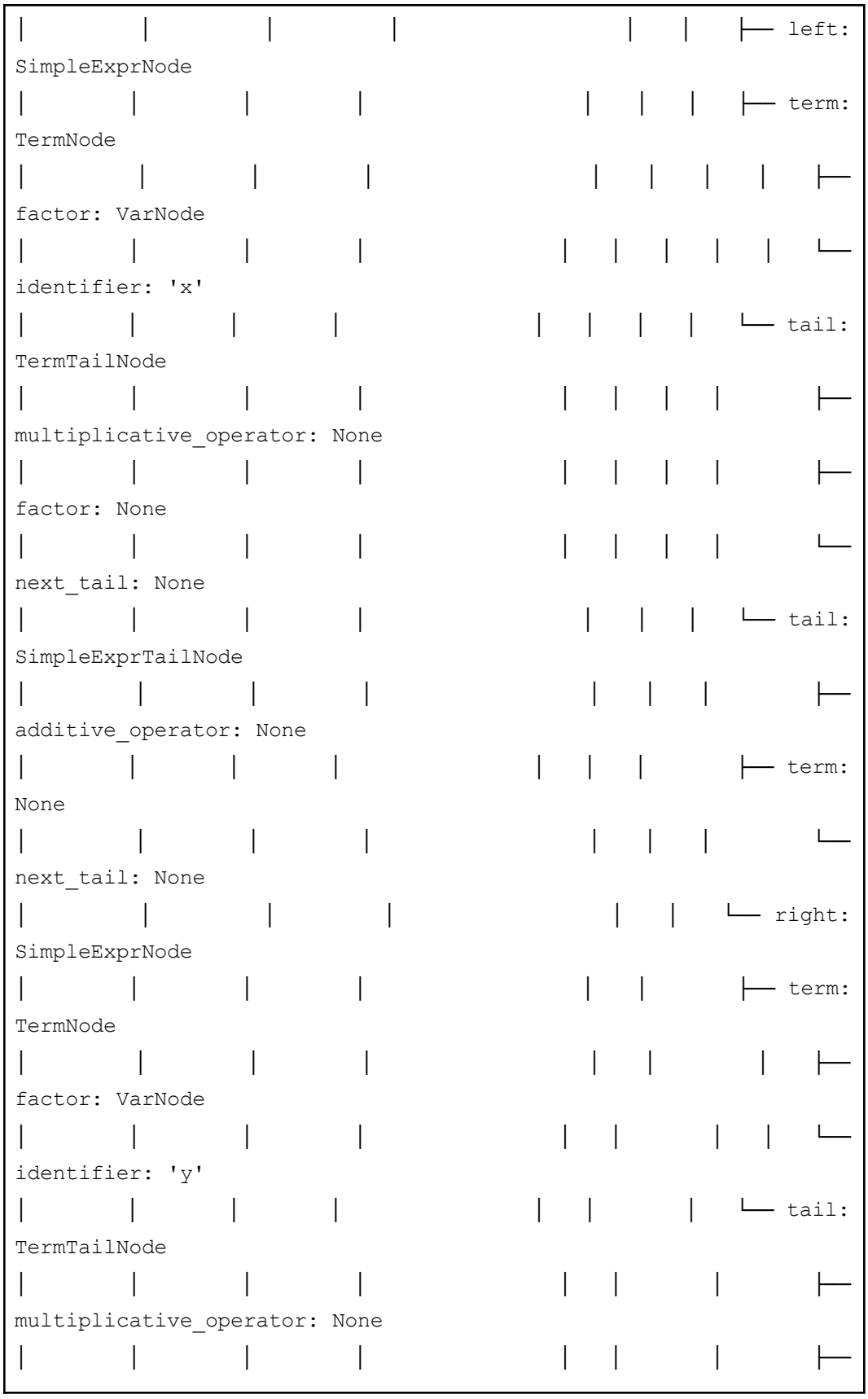

| | | ├── subprogram_declaration: FunctionNode
| | | | ├── identifier: 'null'
| | | | ├── formal_parameter_list: ParameterListNode
| | | | | ├── param_group_node: ParameterGroupNode
| | | | | | ├── modifier: ParameterModifierNode
| | | | | | | └── keyword: None
| | | | | | ├── identifier_list: IdentifierListNode
| | | | | | | ├── identifier: 'x'
| | | | | | | └── tail: IdentifierListTailNode
| | | | | | | ├── identifier: 'y'
| | | | | | | └── next_tail: IdentifierListTailNode
| | | | | | | ├── identifier: None
| | | | | | | └── next_tail: None
| | | | | | └── type_node: TypeNode
| | | | | | └── name: 'real'
| | | | | ├── expression_node: None
| | | | | └── next_tail: ParameterTailNode
| | | | | ├── param_group_node: ParameterGroupNode
| | | | | | ├── modifier: ParameterModifierNode
| | | | | | | └── keyword: None
| | | | | | ├── identifier_list: IdentifierListNode
| | | | | | | ├── identifier: 'z'
| | | | | | | └── tail: IdentifierListTailNode
| | | | | | | ├── identifier: None

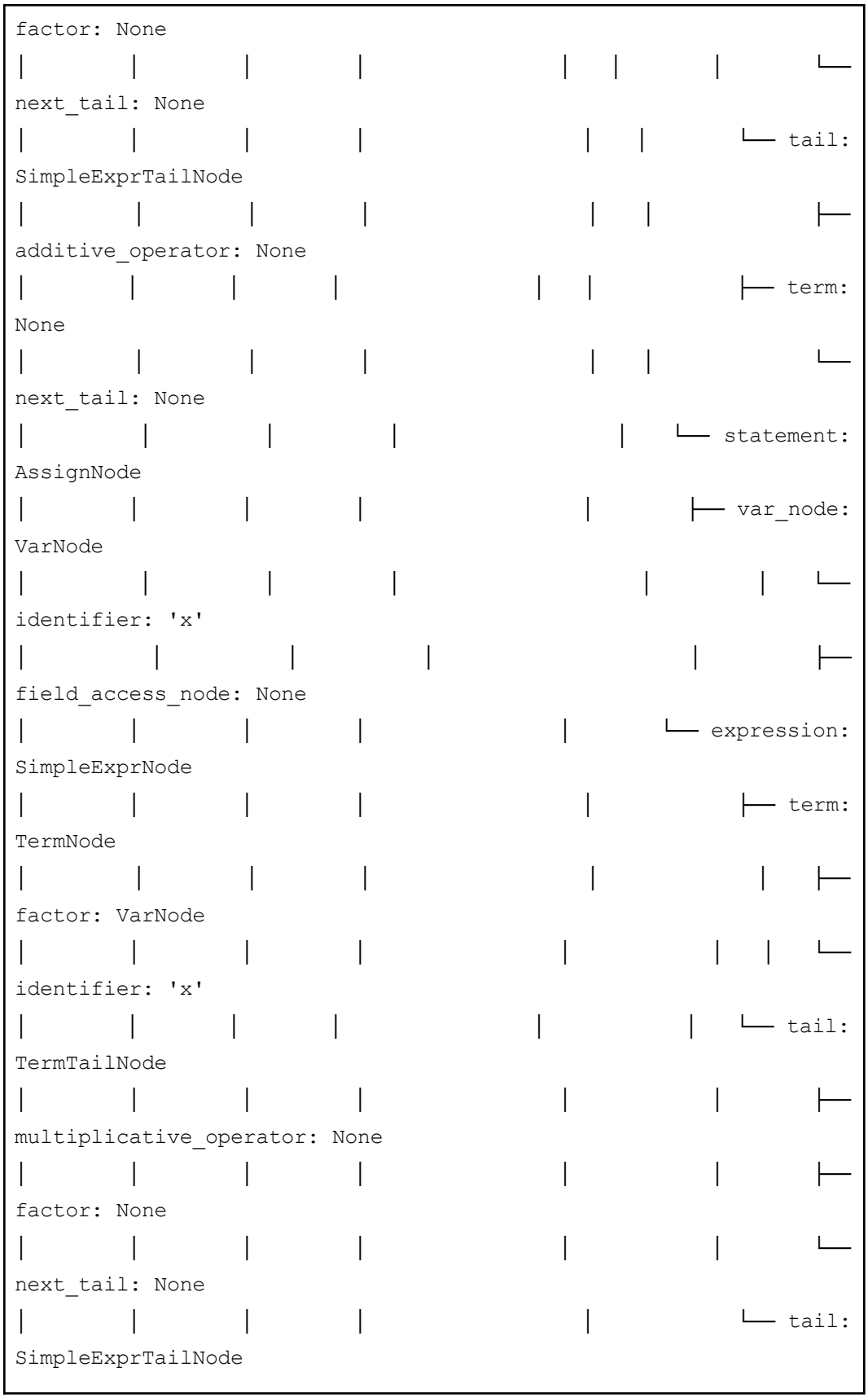

| | | | | | | └── next_tail:
None
| | | | | | └── type_node: TypeNode
| | | | | | └── name: 'complex'
| | | | | ├── expression_node: None
| | | | | └── next_tail:
ParameterTailNode
| | | | | ├── param_group_node:
None
| | | | | ├── expression_node: None
| | | | | └── next_tail: None
| | | | ├── return_type: TypeNode
| | | | | └── name: 'boolean'
| | | | └── block: BlockNode
| | | | ├── declaration_part:
DeclarationPartNode
| | | | | ├── const_section:
ConstSectionNode
| | | | | | ├── const_declaration:
None
| | | | | | └── next_section: None
| | | | | ├── type_section:
TypeSectionNode
| | | | | | ├── type_declaration:
None
| | | | | | └── next_section: None
| | | | | ├── var_section:
VarSectionNode
| | | | | | ├── var_declaration:
VarDeclNode
| | | | | | | ├── var_item:
VarItemNode
| | | | | | | | | ├──
identifier_list: IdentifierListNode
| | | | | | | | | ├──
identifier: 'a'
| | | | | | | | | └── tail:

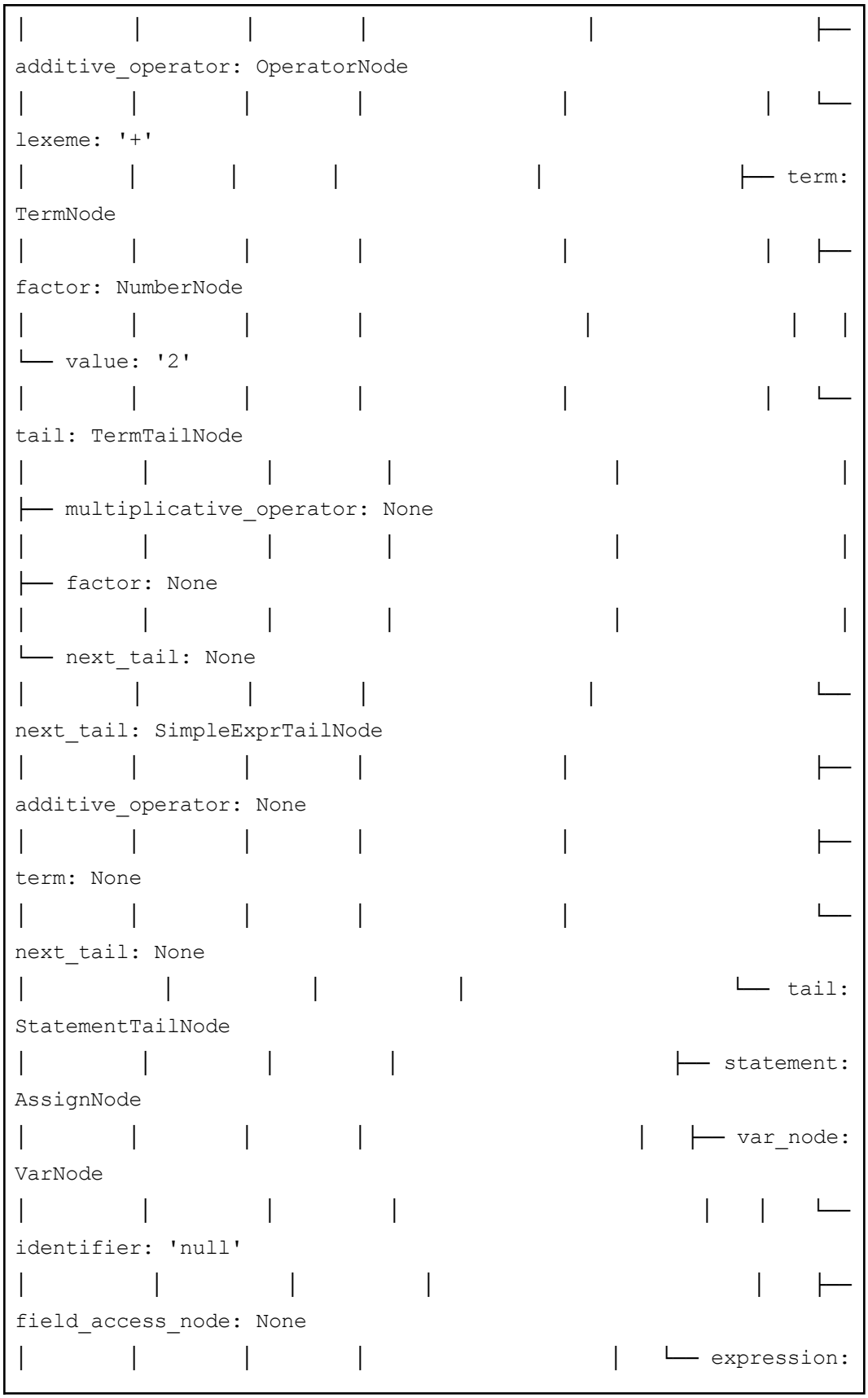

IdentifierListTailNode
identifier: None
next_tail: None
type_node:
ArrayTypeNode
range_node: RangeNode
expression_1: SimpleExprNode
term: TermNode
factor: StringNode
value: ''a''
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
range_operator: OperatorNode
lexeme: '..'

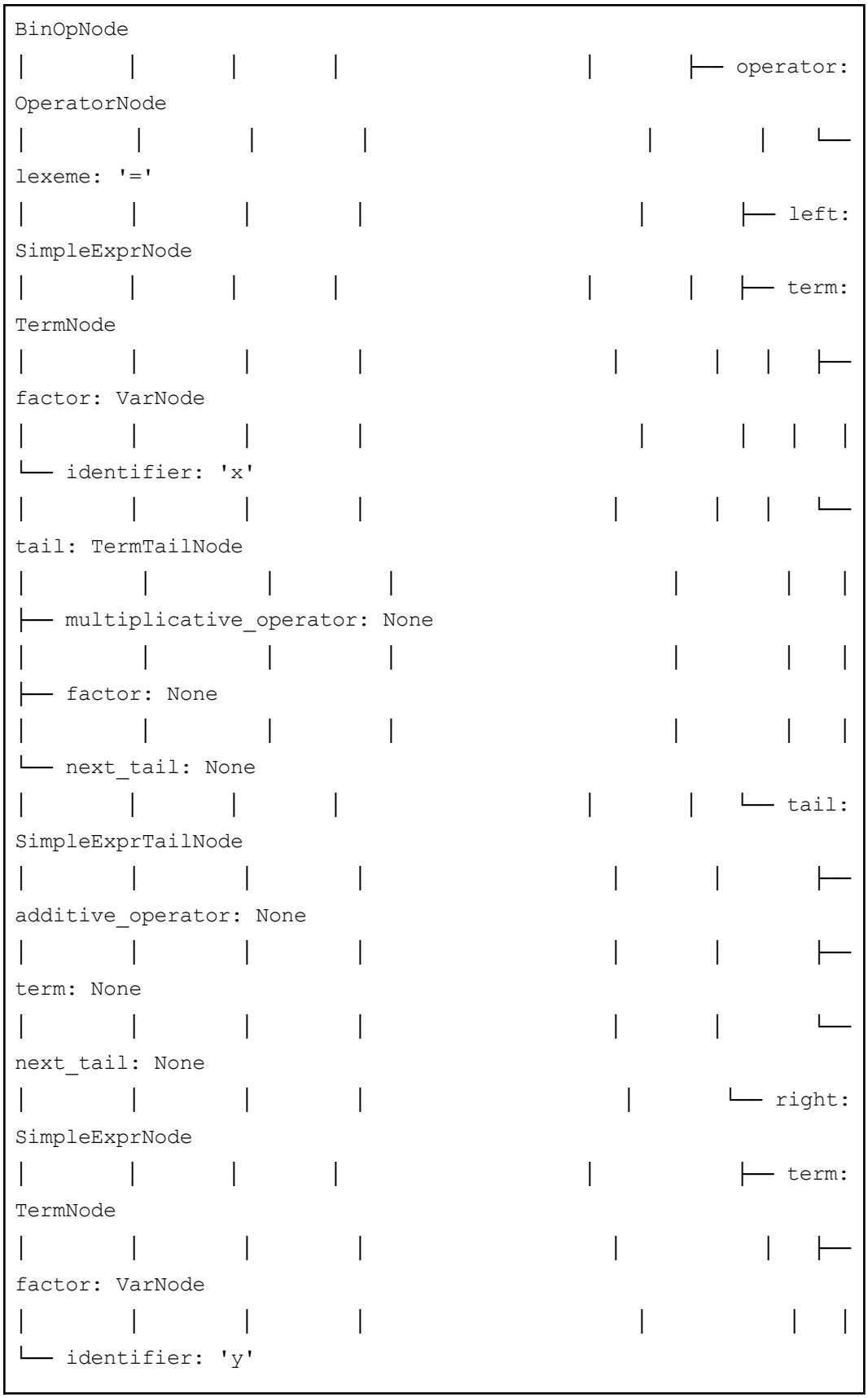

expression_2: SimpleExprNode
term: TermNode
├── factor: StringNode
└── value: ''z''
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
type_node: TypeNode
└── name: 'complex'
└── item_tail: VarTailNode
├── var_item: VarItemNode
├── identifier_list: IdentifierListNode
├── identifier: 'u'
└── tail:

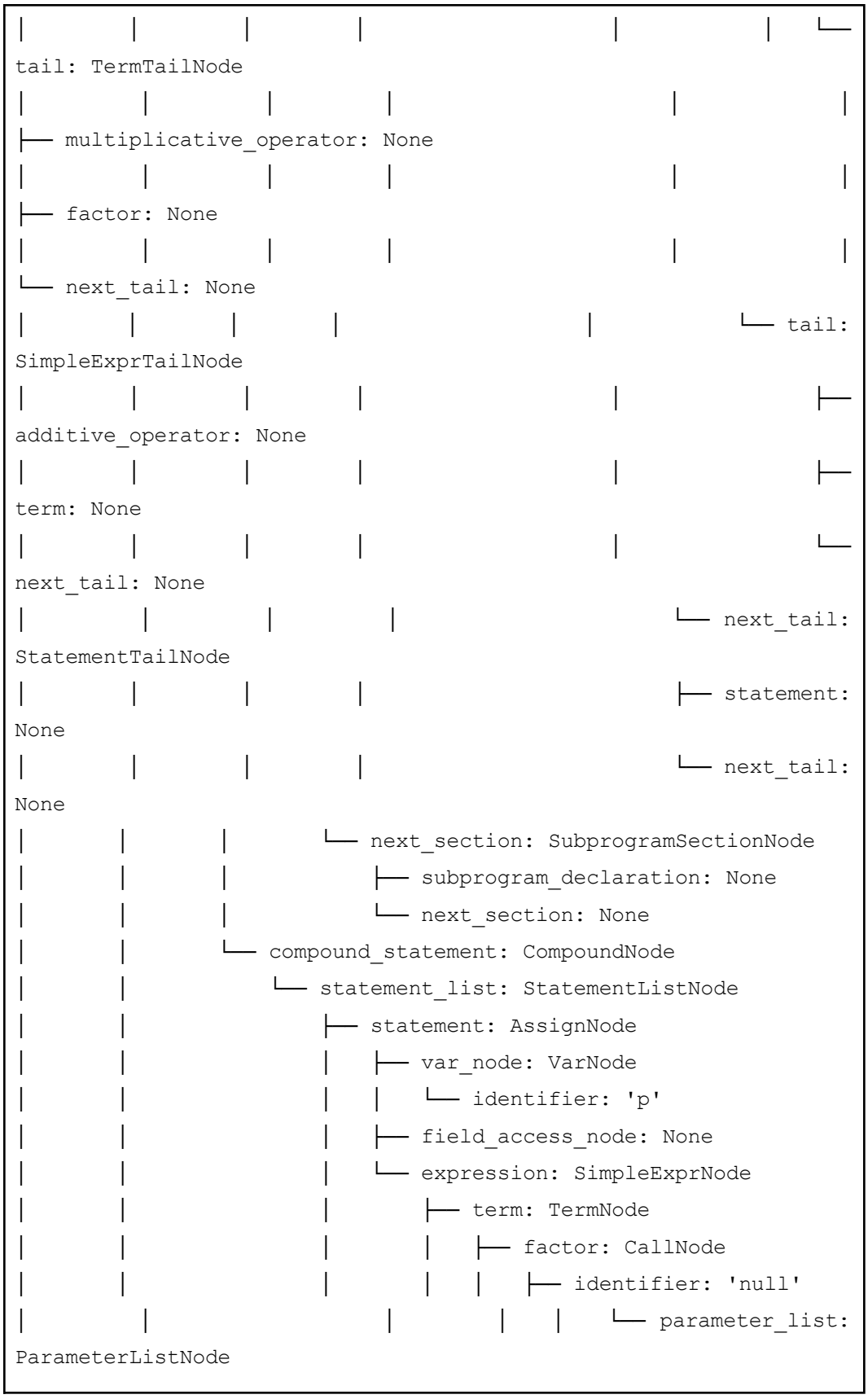

IdentifierListTailNode
identifier: None
next_tail: None
type_node: TypeNode
name:
'char'
next_tail:
VarTailNode
var_item:
None
next_tail: None
next_section:
VarSectionNode
var_declaration:
None
next_section:
None
subprogram_section:
SubprogramSectionNode
subprogram_declaration: None
next_section: None
compound_statement:
CompoundNode
statement_list:
StatementListNode
statement: WhileNode
expression:
BinOpNode
operator:
OperatorNode
lexeme:
'<'

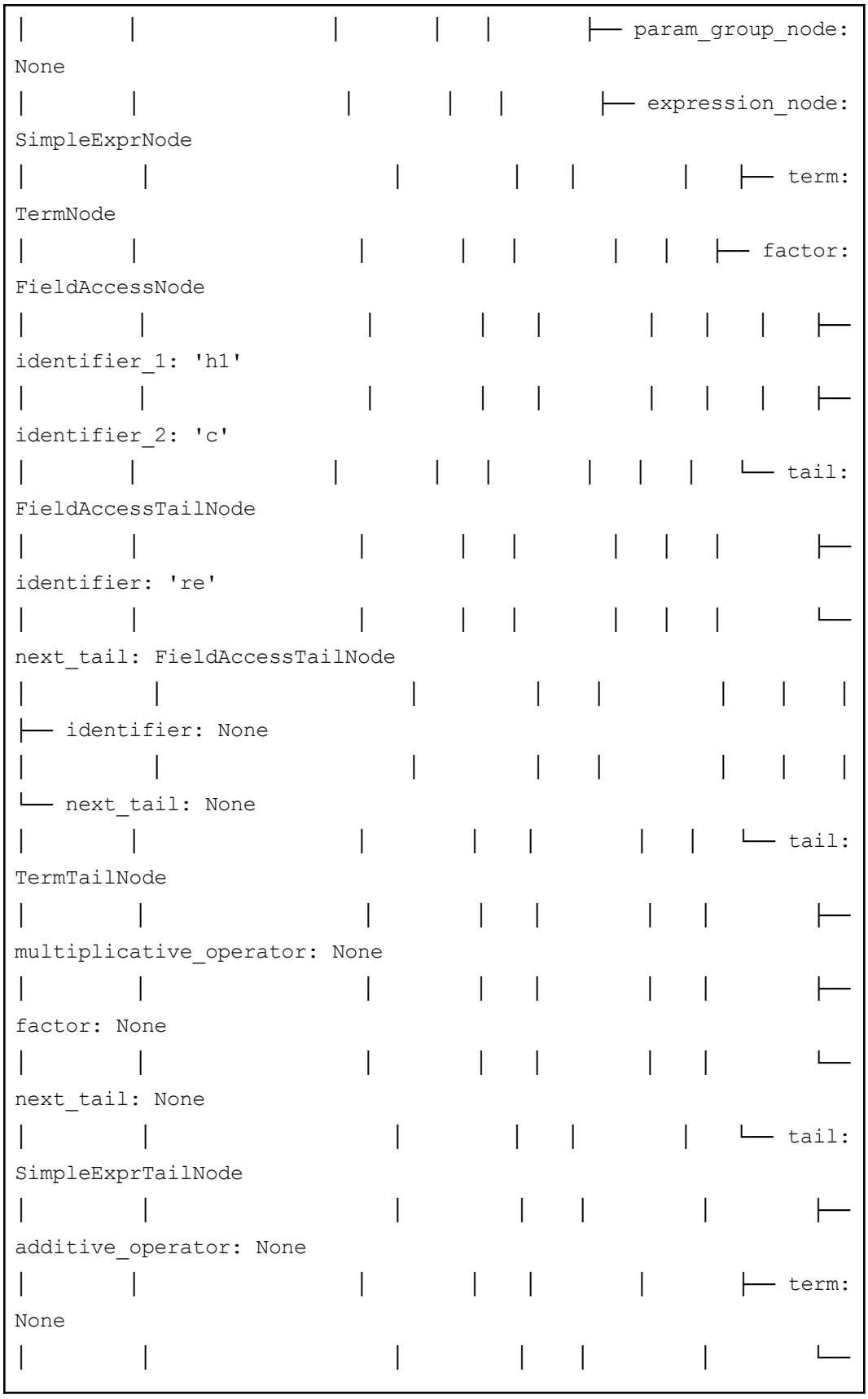

left:
SimpleExprNode
term:
TermNode
factor: VarNode
identifier: 'x'
tail:
TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail:
SimpleExprTailNode
additive_operator: None
term:
None
next_tail: None
right:
SimpleExprNode
term:
TermNode
factor: VarNode
identifier: 'y'
tail:
TermTailNode
multiplicative_operator: None

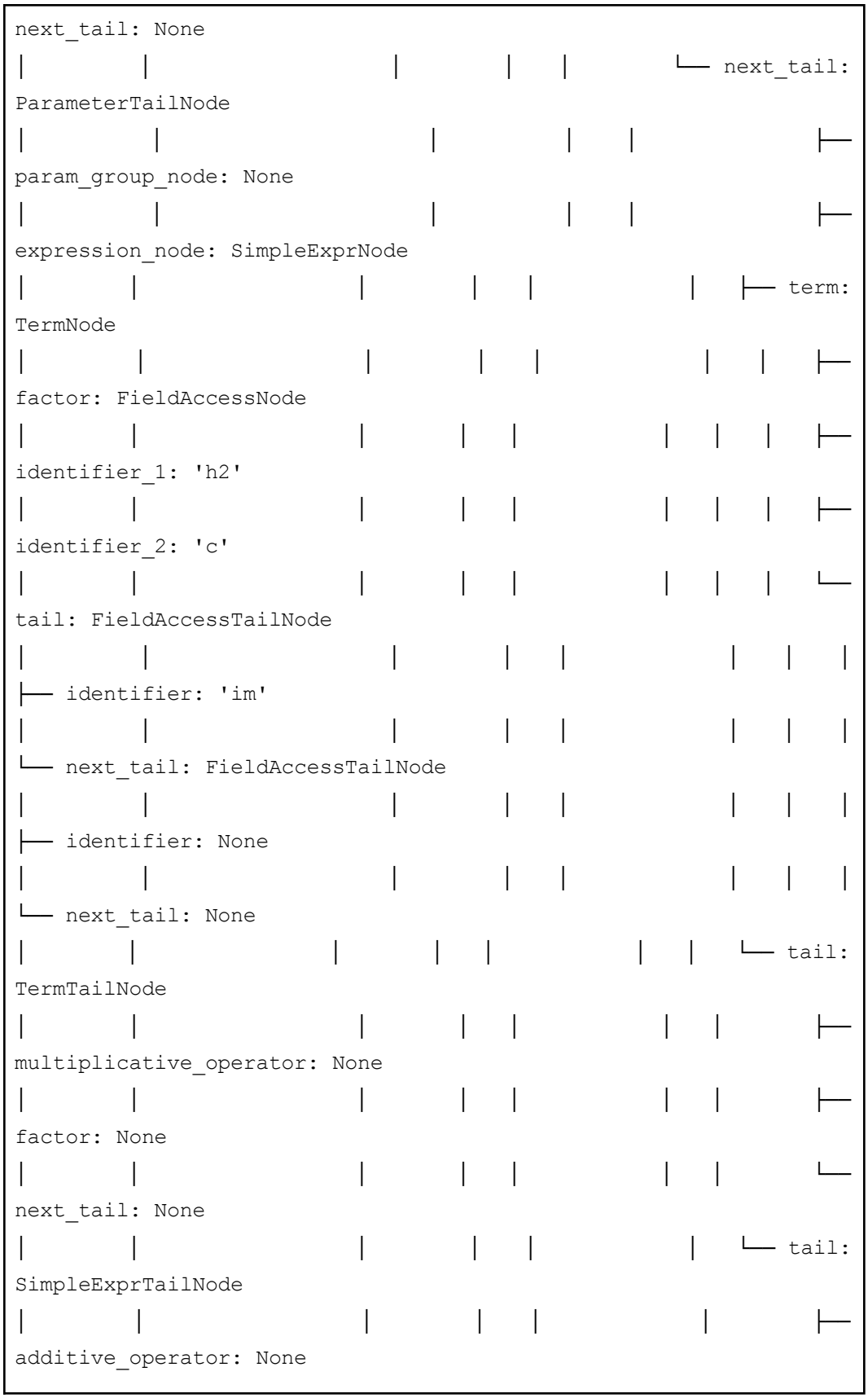

factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
statement: AssignNode
var_node: VarNode
identifier: 'x'
field_access_node: None
expression: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'x'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode

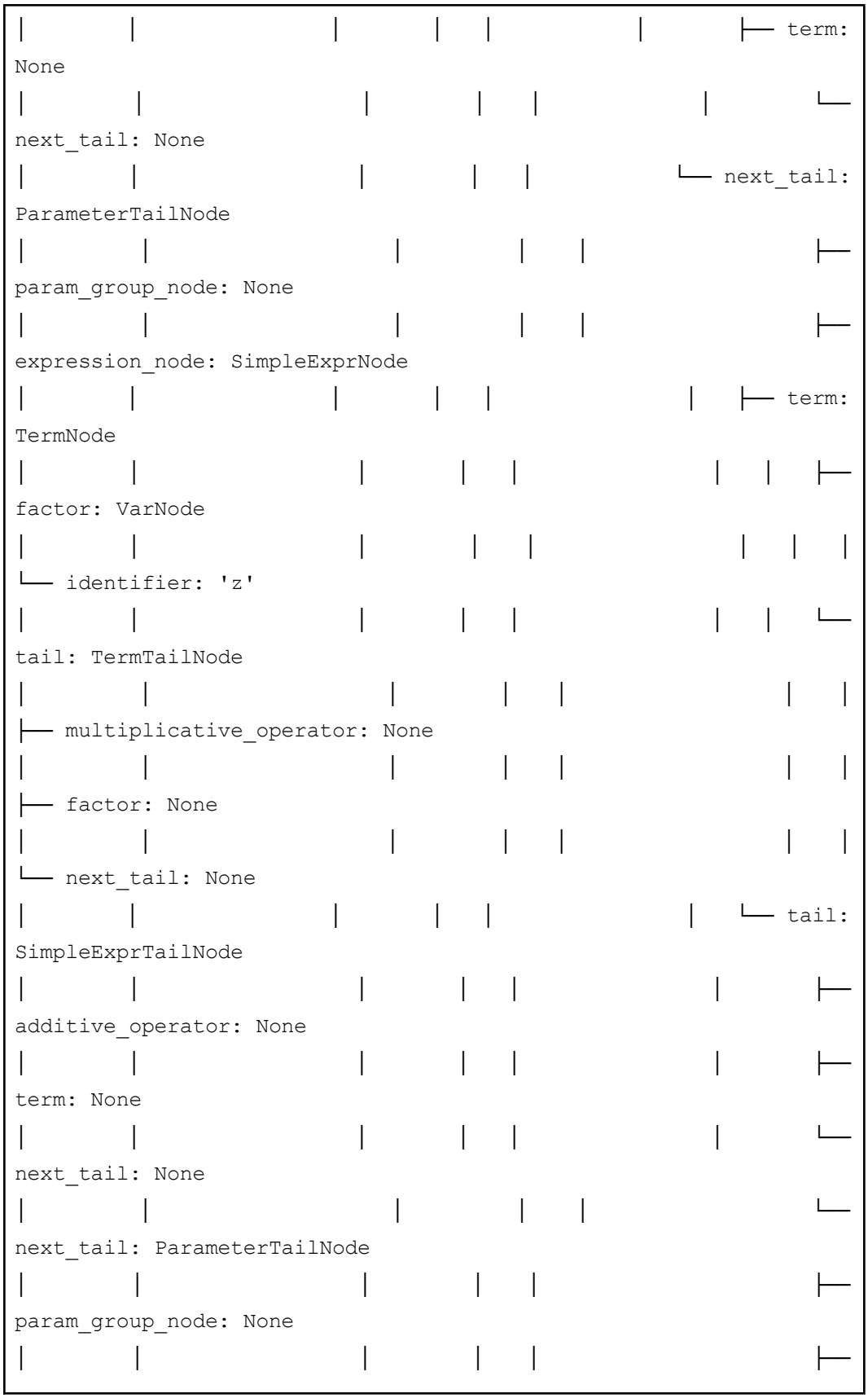

additive_operator: OperatorNode
lexeme: '+'
term: TermNode
factor: NumberNode
value: '2'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
next_tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
tail: StatementTailNode
statement: AssignNode
var_node: VarNode
identifier: 'null'
field_access_node: None
expression:

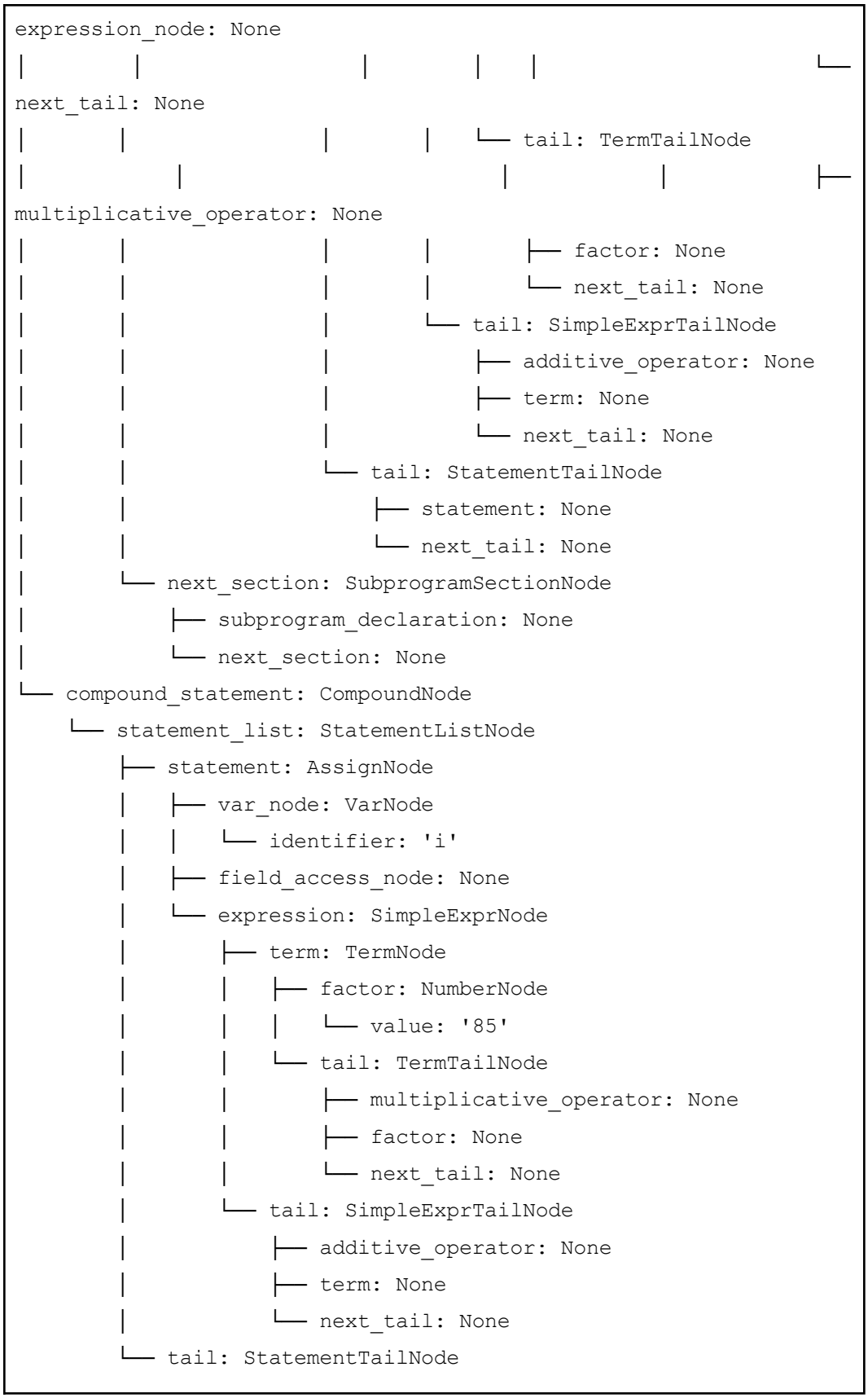

BinOpNode
├── operator: OperatorNode
│   └── lexeme: '='
├── left: SimpleExprNode
│   ├── term: TermNode
│   │   ├── factor: VarNode
│   │   │   └── identifier: 'x'
│   │   └── tail: TermTailNode
│   │   ├── multiplicative_operator: None
│   │   ├── factor: None
│   │   └── next_tail: None
│   └── tail: SimpleExprTailNode
│   ├── additive_operator: None
│   ├── term: None
│   └── next_tail: None
└── right: SimpleExprNode
    ├── term: TermNode
    │   ├── factor: VarNode
    │   │   └── identifier: 'y'

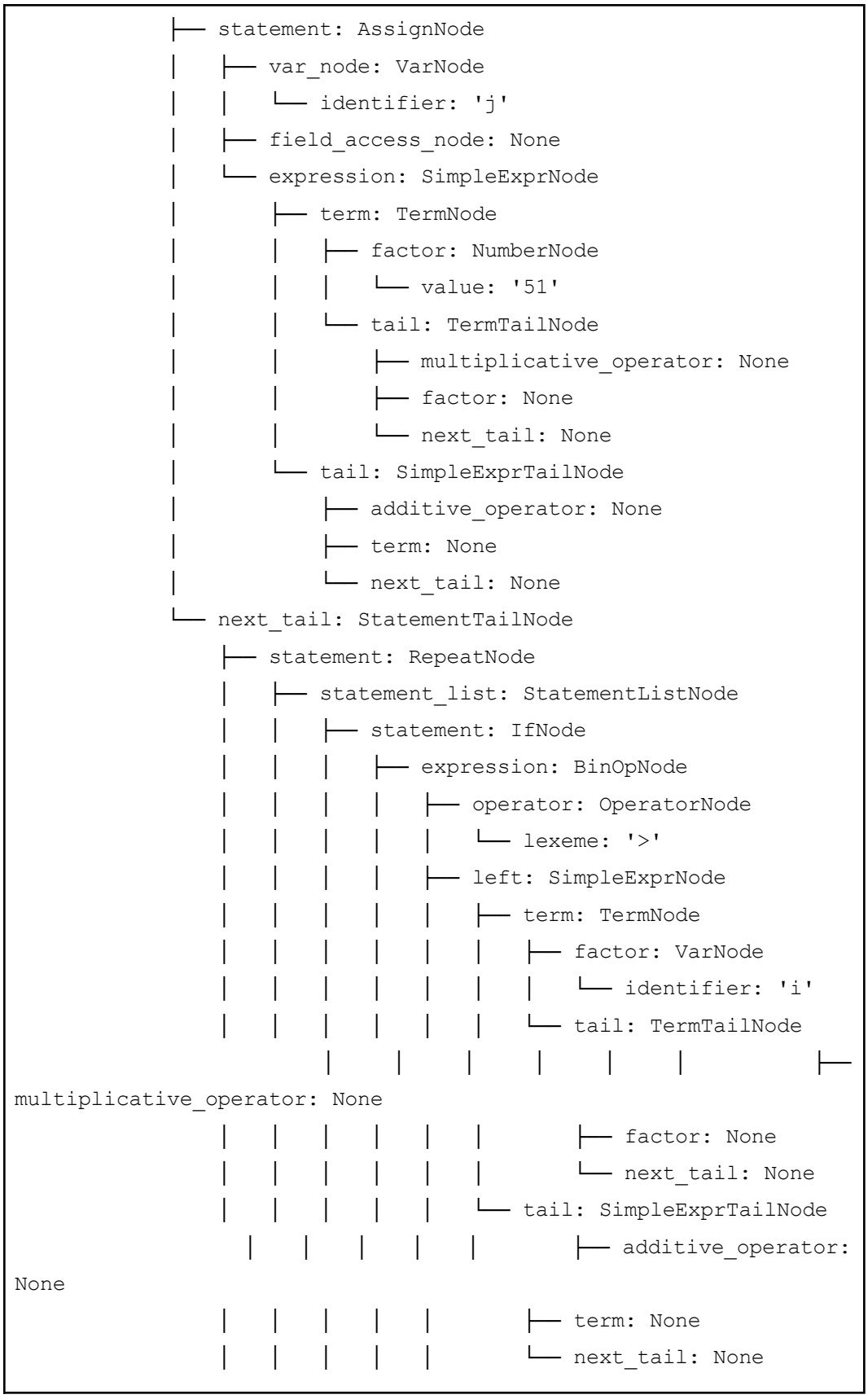

tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
next_tail: StatementTailNode
statement: None
next_tail: None
next_section: SubprogramSectionNode
subprogram_declaration: None
next_section: None
compound_statement: CompoundNode
statement_list: StatementListNode
statement: AssignNode
var_node: VarNode
identifier: 'p'
field_access_node: None
expression: SimpleExprNode
term: TermNode
factor: CallNode
identifier: 'null'
parameter_list: ParameterListNode

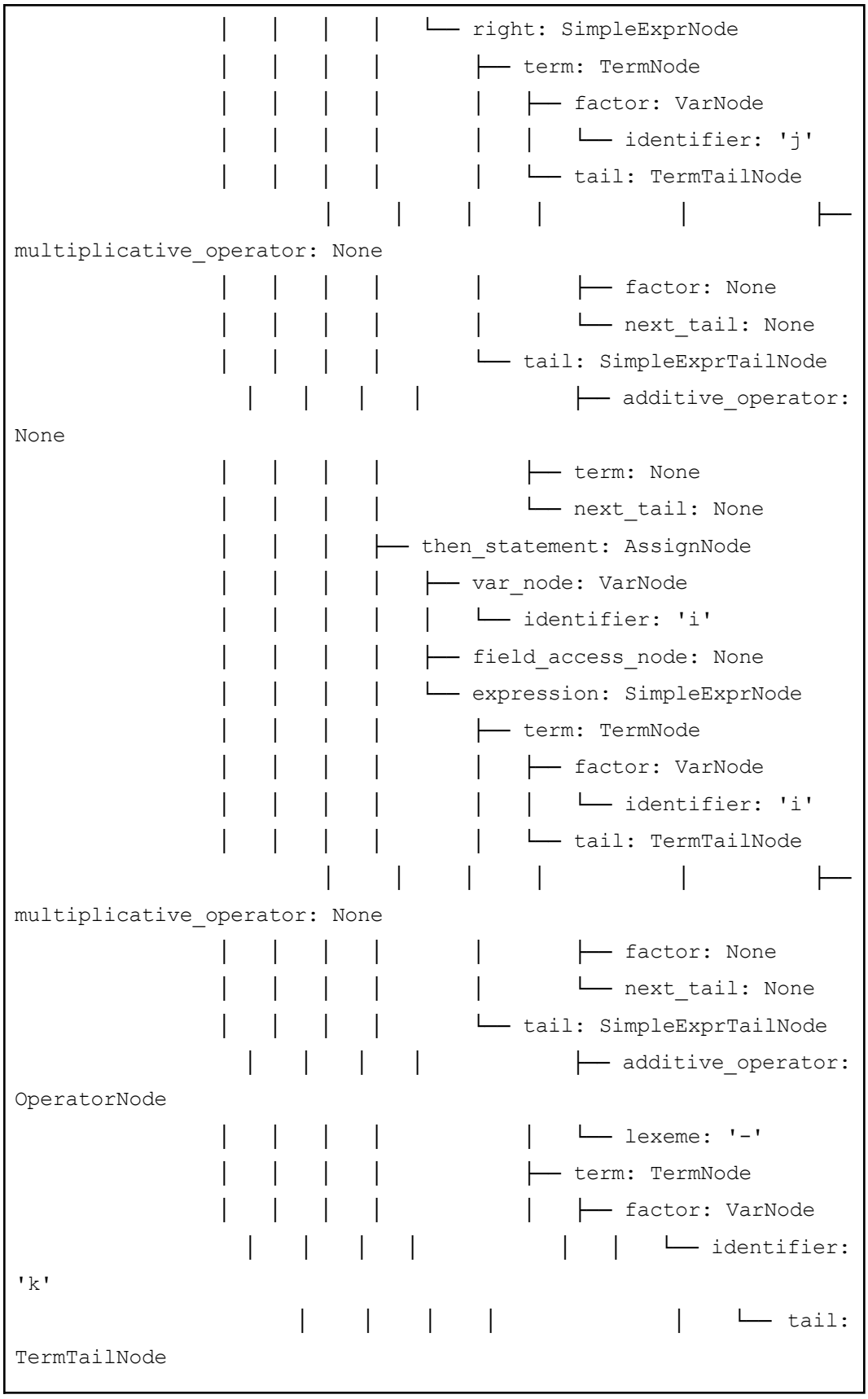

| | | | | ├── param_group_node: None
| | | | | ├── expression_node: SimpleExprNode
| | | | | | ├── term: TermNode
| | | | | | | ├── factor: FieldAccessNode
| | | | | | | | ├── identifier_1: 'h1'
| | | | | | | | ├── identifier_2: 'c'
| | | | | | | | └── tail: FieldAccessTailNode
| | | | | | | | ├── identifier: 're'
| | | | | | | | └── next_tail: FieldAccessTailNode
| | | | | | | | ├── identifier: None
| | | | | | | | └── next_tail: None
| | | | | | | └── tail: TermTailNode
| | | | | | | ├── multiplicative_operator: None
| | | | | | | ├── factor: None
| | | | | | | └── next_tail: None
| | | | | | └── tail: SimpleExprTailNode
| | | | | | ├── additive_operator: None
| | | | | | ├── term: None
| | | | | | └──

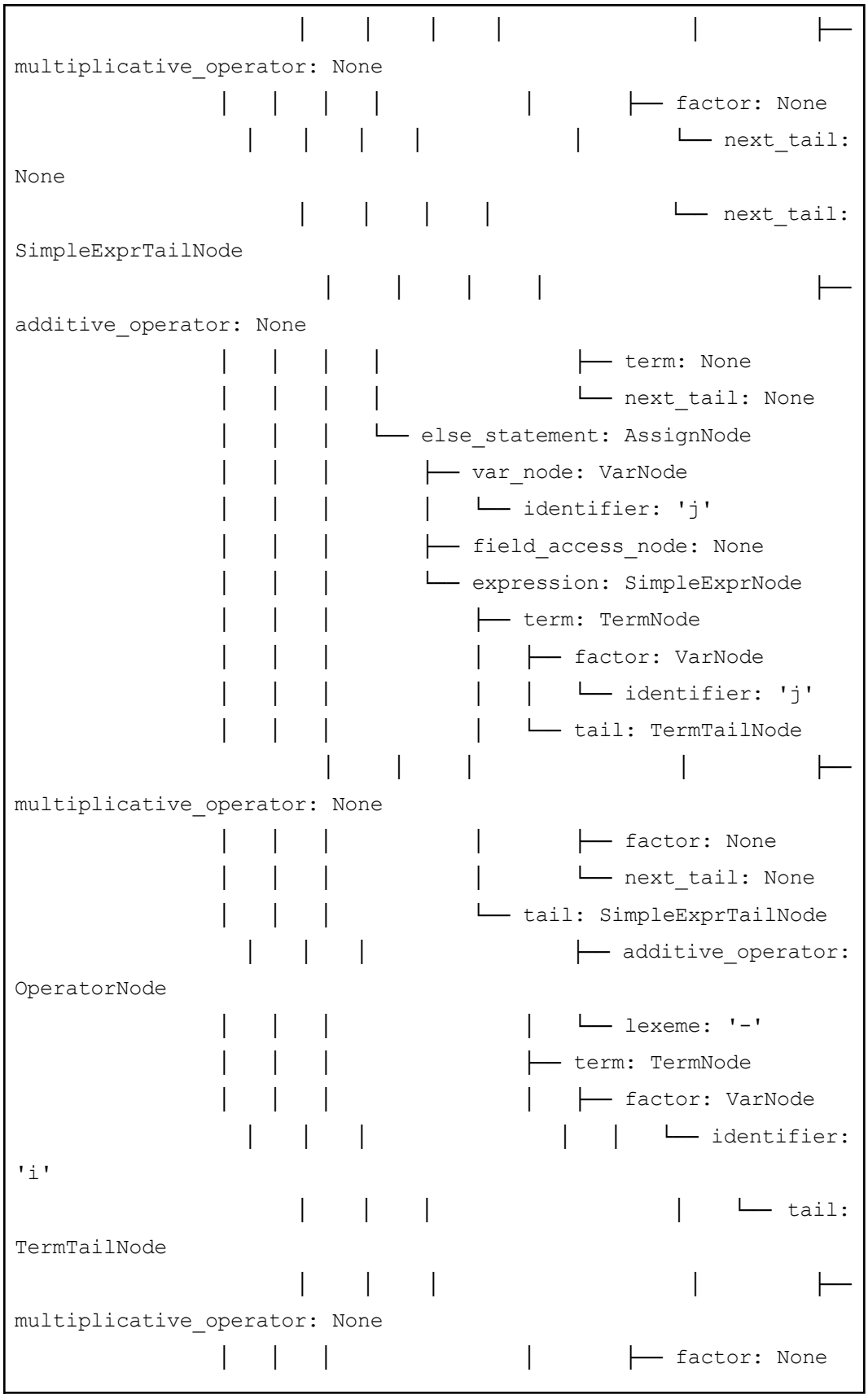

next_tail: None
└── next_tail: ParameterTailNode
param_group_node: None
expression_node: SimpleExprNode
├── term: TermNode
factor: FieldAccessNode
identifier_1: 'h2'
identifier_2: 'c'
└── tail: FieldAccessTailNode
├── identifier: 'im'
└── next_tail: FieldAccessTailNode
├── identifier: None
└── next_tail: None
└── tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
└── tail: SimpleExprTailNode
additive_operator: None

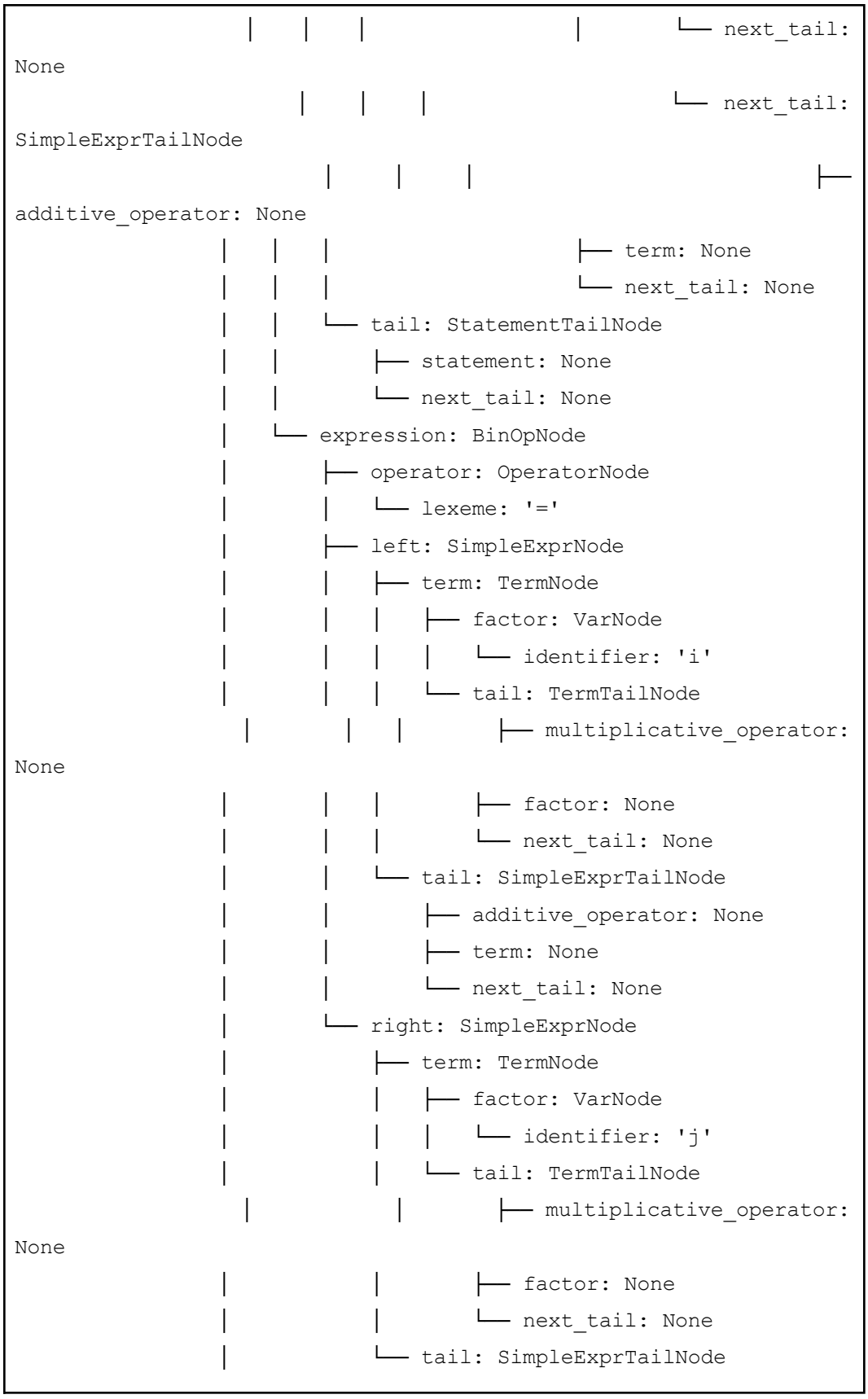

term: None
next_tail: None
next_tail: ParameterTailNode
param_group_node: None
expression_node: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'z'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
next_tail: ParameterTailNode
param_group_node: None

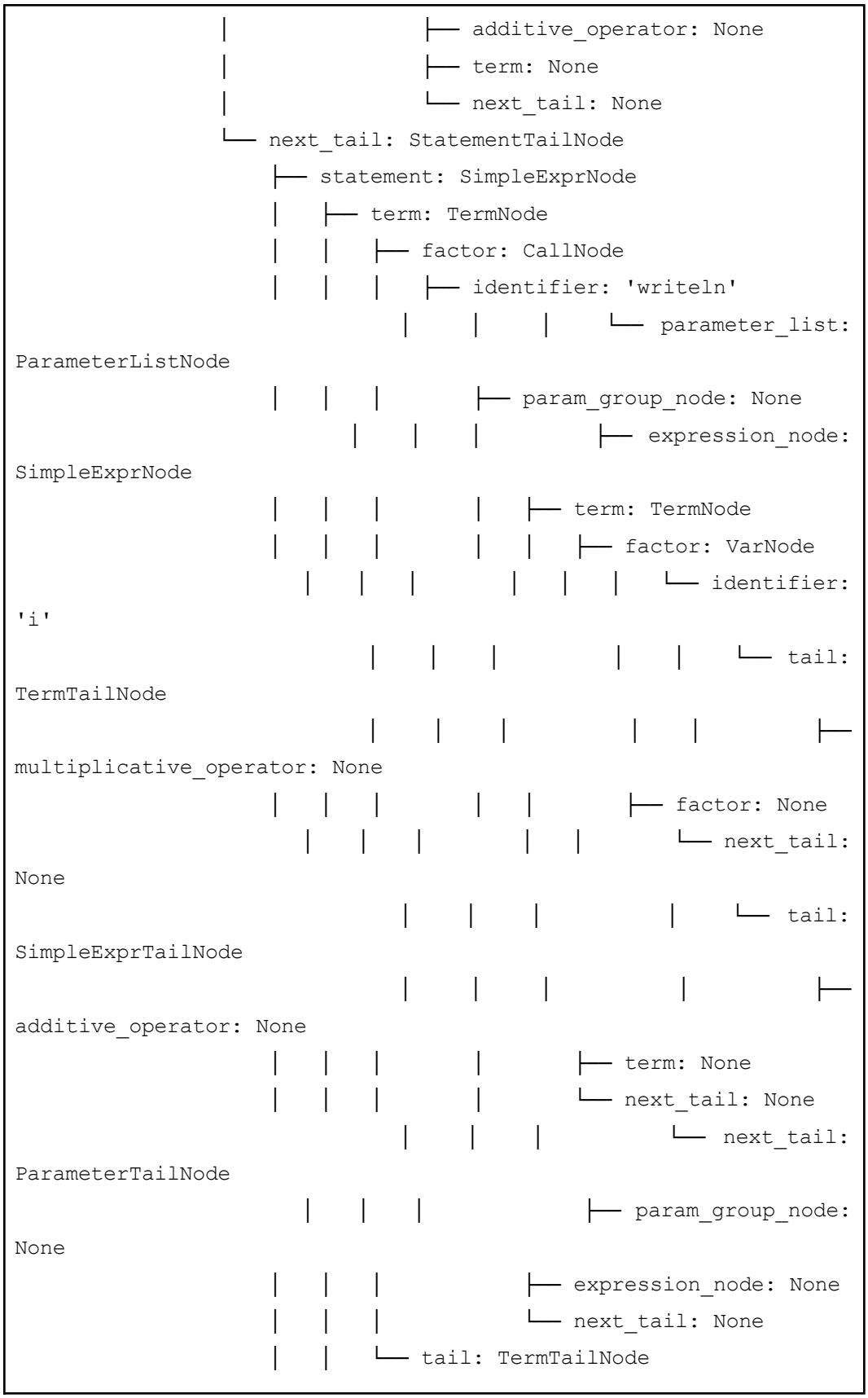

expression_node: None
next_tail: None
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
tail: StatementTailNode
statement: None
next_tail: None
next_section: SubprogramSectionNode
subprogram_declaration: None
next_section: None
compound_statement: CompoundNode
statement_list: StatementListNode
statement: AssignNode
var_node: VarNode
identifier: 'i'
field_access_node: None
expression: SimpleExprNode
term: TermNode
factor: NumberNode
value: '85'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
tail: StatementTailNode


├── statement: AssignNode
│   ├── var_node: VarNode
│   │   └── identifier: 'j'
│   ├── field_access_node: None
│   └── expression: SimpleExprNode
│   ├── term: TermNode
│   │   ├── factor: NumberNode
│   │   │   └── value: '51'
│   │   └── tail: TermTailNode
│   │   ├── multiplicative_operator: None
│   │   ├── factor: None
│   │   └── next_tail: None
│   └── tail: SimpleExprTailNode
│   ├── additive_operator: None
│   ├── term: None
│   └── next_tail: None
└── next_tail: StatementTailNode
    ├── statement: RepeatNode
    │   ├── statement_list: StatementListNode
    │   │   ├── statement: IfNode
    │   │   │   ├── expression: BinOpNode
    │   │   │   │   ├── operator: OperatorNode
    │   │   │   │   │   └── lexeme: '>'
    │   │   │   │   ├── left: SimpleExprNode
    │   │   │   │   │   ├── term: TermNode
    │   │   │   │   │   │   ├── factor: VarNode
    │   │   │   │   │   │   │   └── identifier: 'i'
    │   │   │   │   │   │   └── tail: TermTailNode
    │   │   │   │   │   │   ├── multiplicative_operator: None
    │   │   │   │   │   │   ├── factor: None
    │   │   │   │   │   │   └── next_tail: None
    │   │   │   │   │   └── tail: SimpleExprTailNode
    │   │   │   │   │   ├── additive_operator: None
    │   │   │   │   │   ├── term: None
    │   │   │   │   │   └── next_tail: None


└── right: SimpleExprNode
├── term: TermNode
├── factor: VarNode
└── identifier: 'j'
└── tail: TermTailNode
multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
├── then_statement: AssignNode
├── var_node: VarNode
└── identifier: 'i'
├── field_access_node: None
└── expression: SimpleExprNode
├── term: TermNode
├── factor: VarNode
└── identifier: 'i'
└── tail: TermTailNode
multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: OperatorNode
└── lexeme: '-'
├── term: TermNode
├── factor: VarNode
└── identifier: 'k'
└── tail: TermTailNode


multiplicative_operator: None
factor: None
next_tail: None
next_tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
else_statement: AssignNode
var_node: VarNode
identifier: 'j'
field_access_node: None
expression: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'j'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: OperatorNode
lexeme: '-'
term: TermNode
factor: VarNode
identifier: 'i'
tail: TermTailNode
multiplicative_operator: None
factor: None


| | | | └── next_tail:
None
| | | └── next_tail:
SimpleExprTailNode
| | | ├──
additive_operator: None
| | | ├── term: None
| | | └── next_tail: None
| | └── tail: StatementTailNode
| | ├── statement: None
| | └── next_tail: None
| └── expression: BinOpNode
| ├── operator: OperatorNode
| | └── lexeme: '='
| ├── left: SimpleExprNode
| | ├── term: TermNode
| | | ├── factor: VarNode
| | | | └── identifier: 'i'
| | | └── tail: TermTailNode
| | | | ├── multiplicative_operator:
None
| | | ├── factor: None
| | | └── next_tail: None
| | └── tail: SimpleExprTailNode
| | ├── additive_operator: None
| | ├── term: None
| | └── next_tail: None
| └── right: SimpleExprNode
| ├── term: TermNode
| | ├── factor: VarNode
| | | └── identifier: 'j'
| | └── tail: TermTailNode
| | | ├── multiplicative_operator:
None
| | ├── factor: None
| | └── next_tail: None
| └── tail: SimpleExprTailNode


additive_operator: None
term: None
next_tail: None
next_tail: StatementTailNode
statement: SimpleExprNode
term: TermNode
factor: CallNode
identifier: 'writeln'
parameter_list: ParameterListNode
param_group_node: None
expression_node: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'i'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
next_tail: ParameterTailNode
param_group_node: None
expression_node: None
next_tail: None
tail: TermTailNode

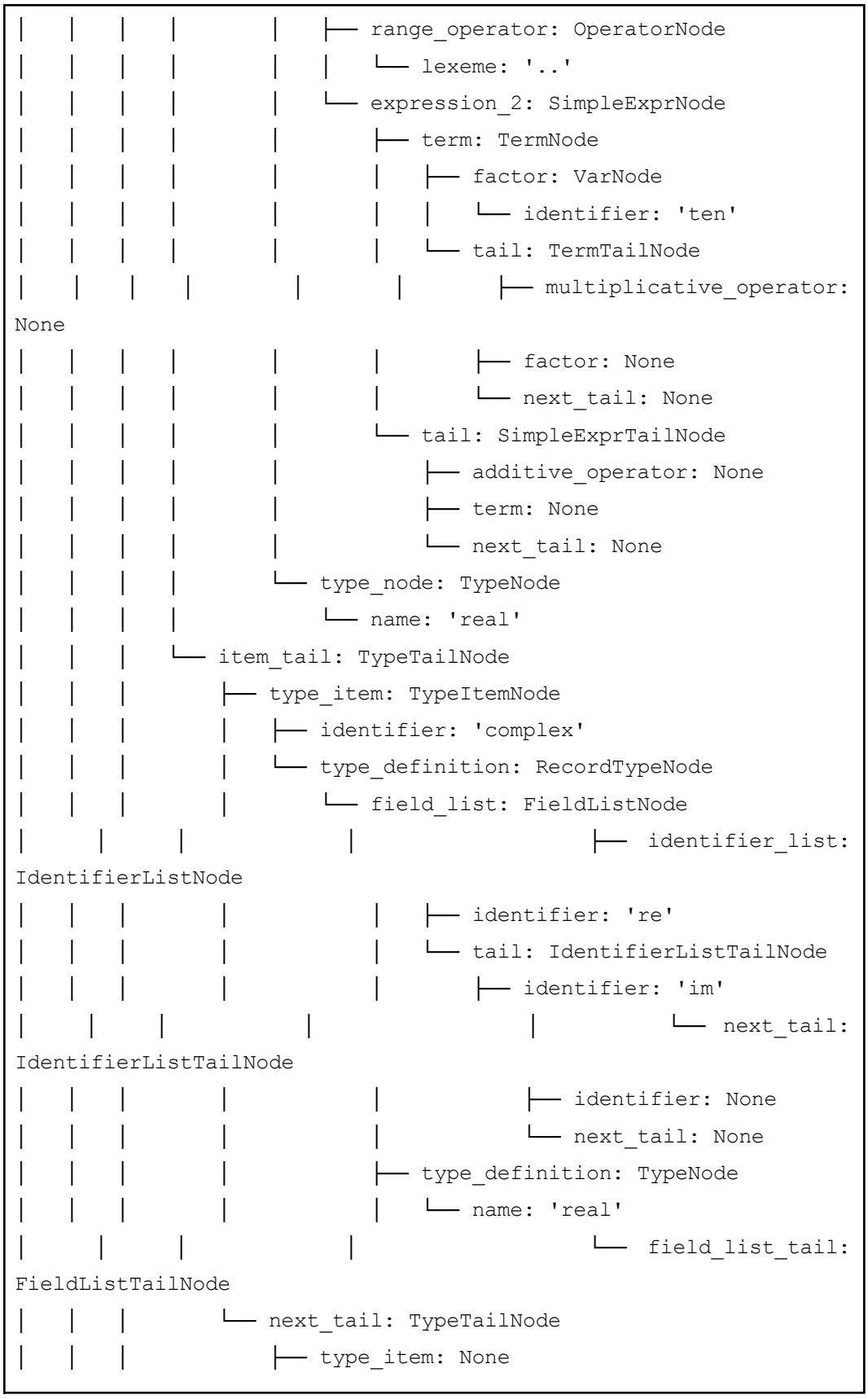
```
                    │   │       ├── multiplicative_operator: None
                    │   │       ├── factor: None
                    │   │       └── next_tail: None
                    │   └── tail: SimpleExprTailNode
                    │       ├── additive_operator: None
                    │       ├── term: None
                    │       └── next_tail: None
                    └── next_tail: StatementTailNode
                        ├── statement: None
                        └── next_tail: None

==================================================
Tahap: Semantic Analysis (Symbol Tables)
==================================================
Semantic Error: Undeclared variable 'k'
```

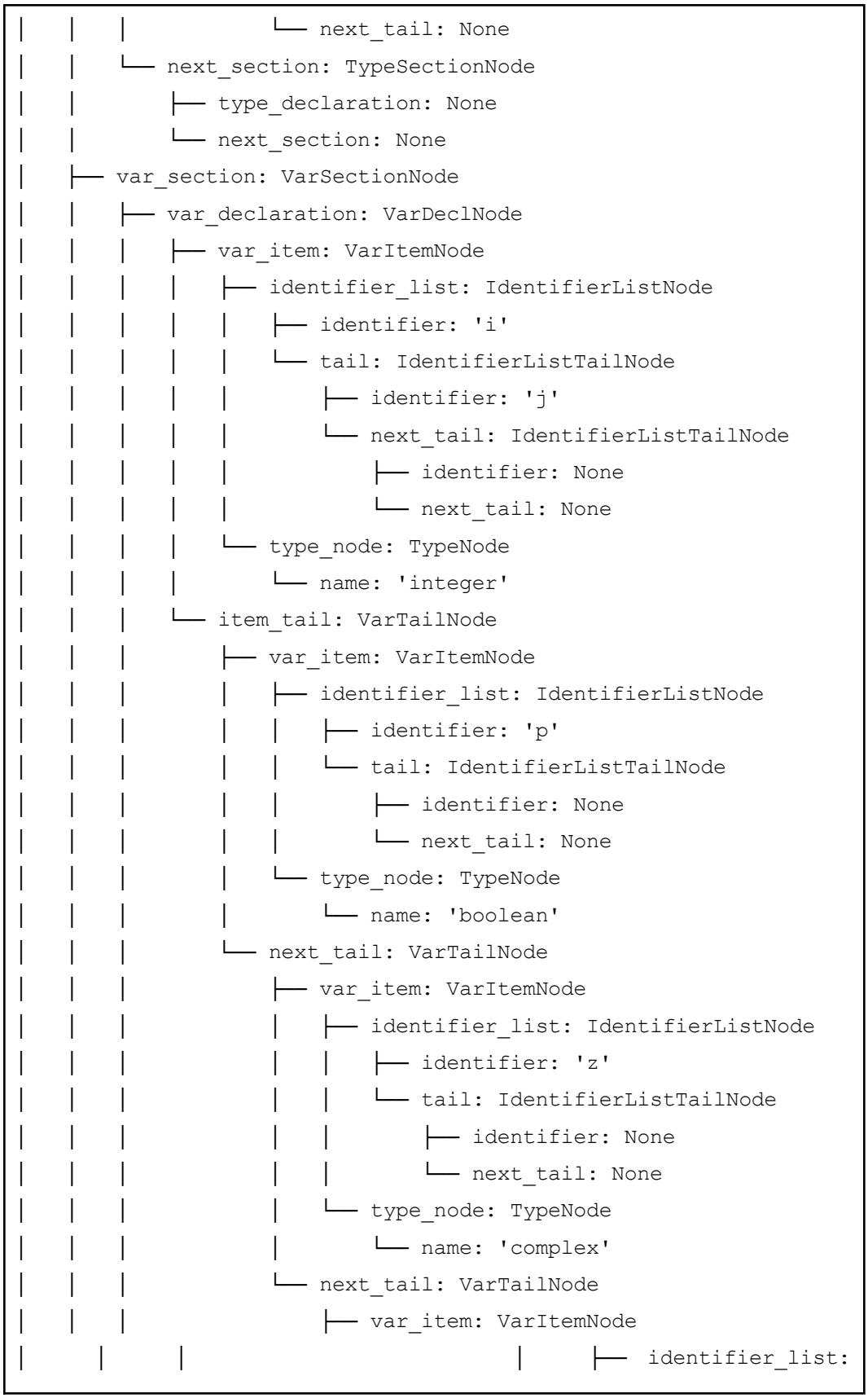
Pembahasan: Pada pengujian ini, Semantic Analyzer berhasil mendeteksi penggunaan variabel yang belum dideklarasikan. AST berhasil dibentuk untuk struktur program, namun proses analisis semantik menghentikan kompilasi saat menemukan variabel 'k' digunakan dalam operasi pengurangan `i := i - k`. Pesan error "Undeclared variable 'k'" memberitahu bahwa Symbol Table berfungsi dengan baik dalam memvalidasi keberadaan identifier sebelum digunakan. Karena 'k' tidak ditemukan dalam scope manapun (baik lokal maupun global), analyzer dengan tepat menolak program tersebut, memastikan integritas referensi variabel dalam kode.

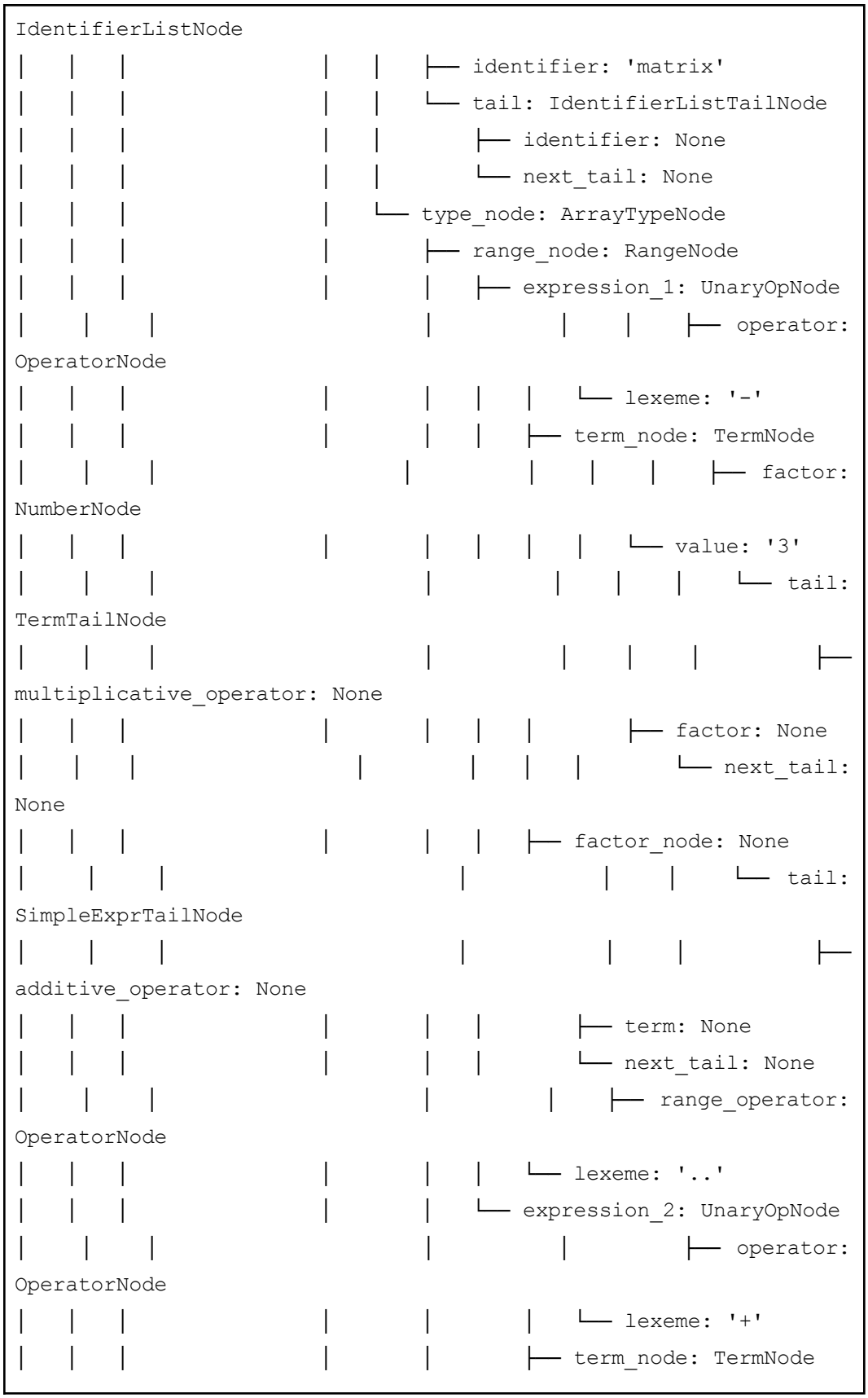
3.2.4 Hasil Uji 4

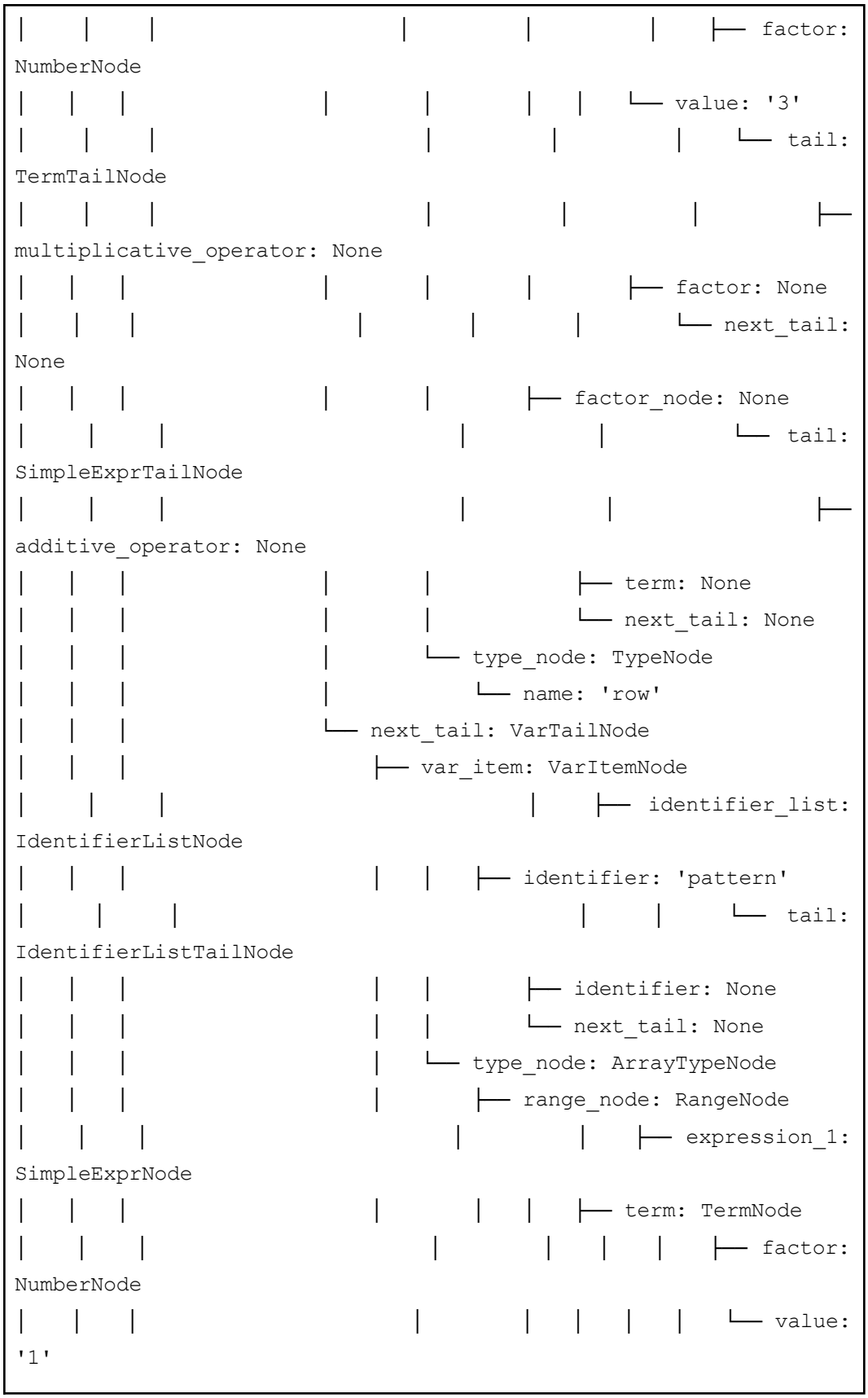
```
==================================================
Tahap: AST Generation (Abstract Syntax Tree)
==================================================
AST Structure:
------------------------------
ProgramNode
├── program_header: ProgramHeaderNode
│   └── identifier: 'test0'
├── declaration_part: DeclarationPartNode
│   ├── const_section: ConstSectionNode
```

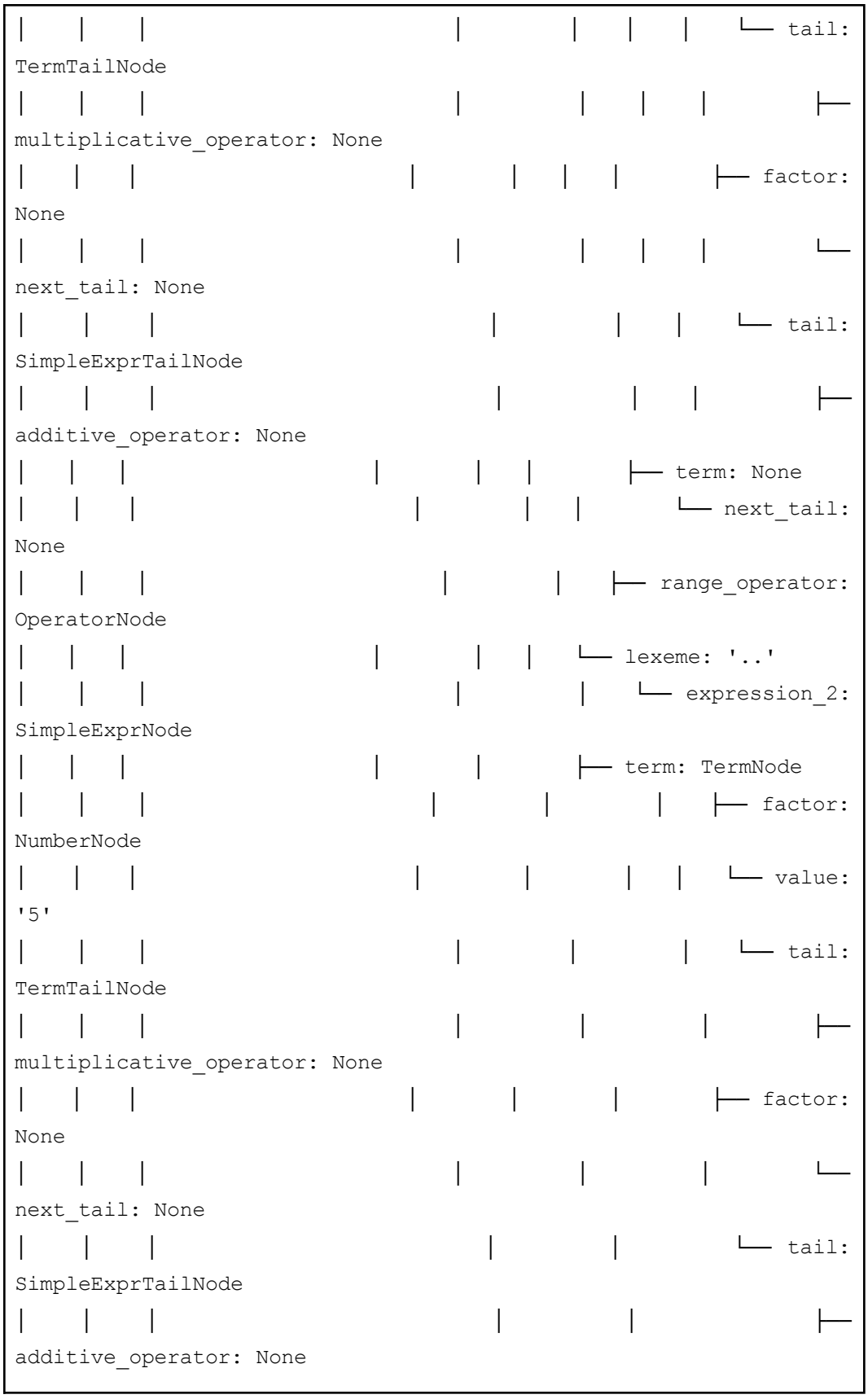
```
|   |   ├── const_declaration: ConstDeclNode
|   |   |   ├── const_item: ConstItemNode
|   |   |   |   ├── identifier: 'ten'
|   |   |   |   ├── assign_operator: '='
|   |   |   |   └── value_node: NumberNode
|   |   |   |       └── value: '10'
|   |   |   └── item_tail: ConstTailNode
|   |   |       ├── const_item: ConstItemNode
|   |   |       |   ├── identifier: 'plus'
|   |   |       |   ├── assign_operator: '='
|   |   |       |   └── value_node: StringNode
|   |   |       |       └── value: ''+''
|   |   |       └── next_tail: ConstTailNode
|   |   |           ├── const_item: None
|   |   |           └── next_tail: None
|   |   └── next_section: ConstSectionNode
|   |       ├── const_declaration: None
|   |       └── next_section: None
|   ├── type_section: TypeSectionNode
|   |   ├── type_declaration: TypeDeclNode
|   |   |   ├── type_item: TypeItemNode
|   |   |   |   ├── identifier: 'row'
|   |   |   |   └── type_definition: ArrayTypeNode
|   |   |   |       ├── range_node: RangeNode
|   |   |   |       |   ├── expression_1: SimpleExprNode
|   |   |   |       |   |   ├── term: TermNode
|   |   |   |       |   |   |   ├── factor: NumberNode
|   |   |   |       |   |   |   |   └── value: '1'
|   |   |   |       |   |   |   └── tail: TermTailNode
|   |   |   |       |   |   |       ├── multiplicative_operator:
None
|   |   |   |       |   |   |       ├── factor: None
|   |   |   |       |   |   |       └── next_tail: None
|   |   |   |       |   |   └── tail: SimpleExprTailNode
|   |   |   |       |   |       ├── additive_operator: None
|   |   |   |       |   |       ├── term: None
|   |   |   |       |   |       └── next_tail: None
```

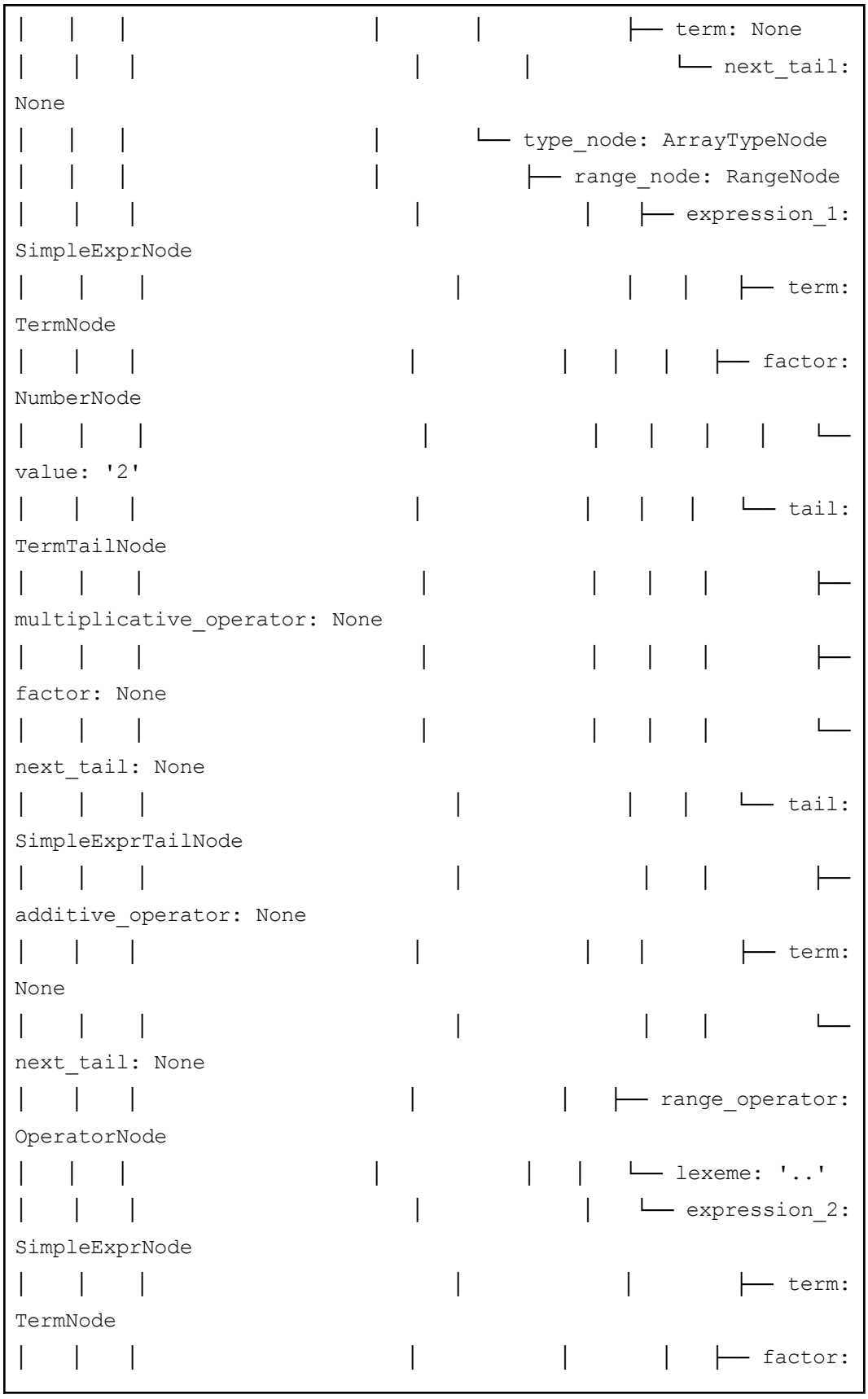

range_operator: OperatorNode
lexeme: '..'
expression_2: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'ten'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
type_node: TypeNode
name: 'real'
item_tail: TypeTailNode
type_item: TypeItemNode
identifier: 'complex'
type_definition: RecordTypeNode
field_list: FieldListNode
identifier_list: IdentifierListNode
identifier: 're'
tail: IdentifierListTailNode
identifier: 'im'
next_tail: IdentifierListTailNode
identifier: None
next_tail: None
type_definition: TypeNode
name: 'real'
field_list_tail: FieldListTailNode
next_tail: TypeTailNode
type_item: None

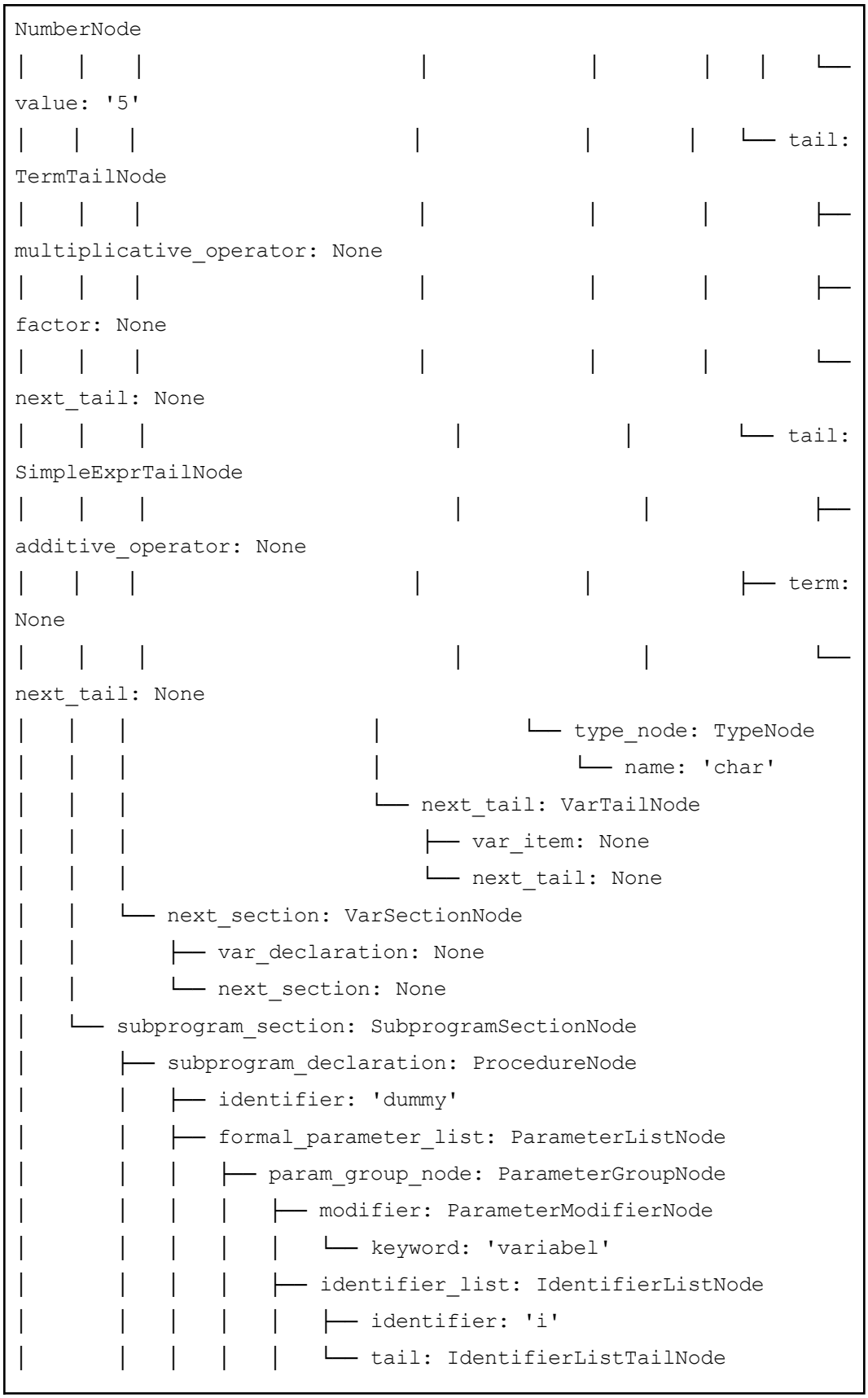

└── next_tail: None
└── next_section: TypeSectionNode
├── type_declaration: None
└── next_section: None
├── var_section: VarSectionNode
├── var_declaration: VarDeclNode
├── var_item: VarItemNode
├── identifier_list: IdentifierListNode
├── identifier: 'i'
└── tail: IdentifierListTailNode
├── identifier: 'j'
└── next_tail: IdentifierListTailNode
├── identifier: None
└── next_tail: None
└── type_node: TypeNode
└── name: 'integer'
└── item_tail: VarTailNode
├── var_item: VarItemNode
├── identifier_list: IdentifierListNode
├── identifier: 'p'
└── tail: IdentifierListTailNode
├── identifier: None
└── next_tail: None
└── type_node: TypeNode
└── name: 'boolean'
└── next_tail: VarTailNode
├── var_item: VarItemNode
├── identifier_list: IdentifierListNode
├── identifier: 'z'
└── tail: IdentifierListTailNode
├── identifier: None
└── next_tail: None
└── type_node: TypeNode
└── name: 'complex'
└── next_tail: VarTailNode
├── var_item: VarItemNode
├── identifier_list:

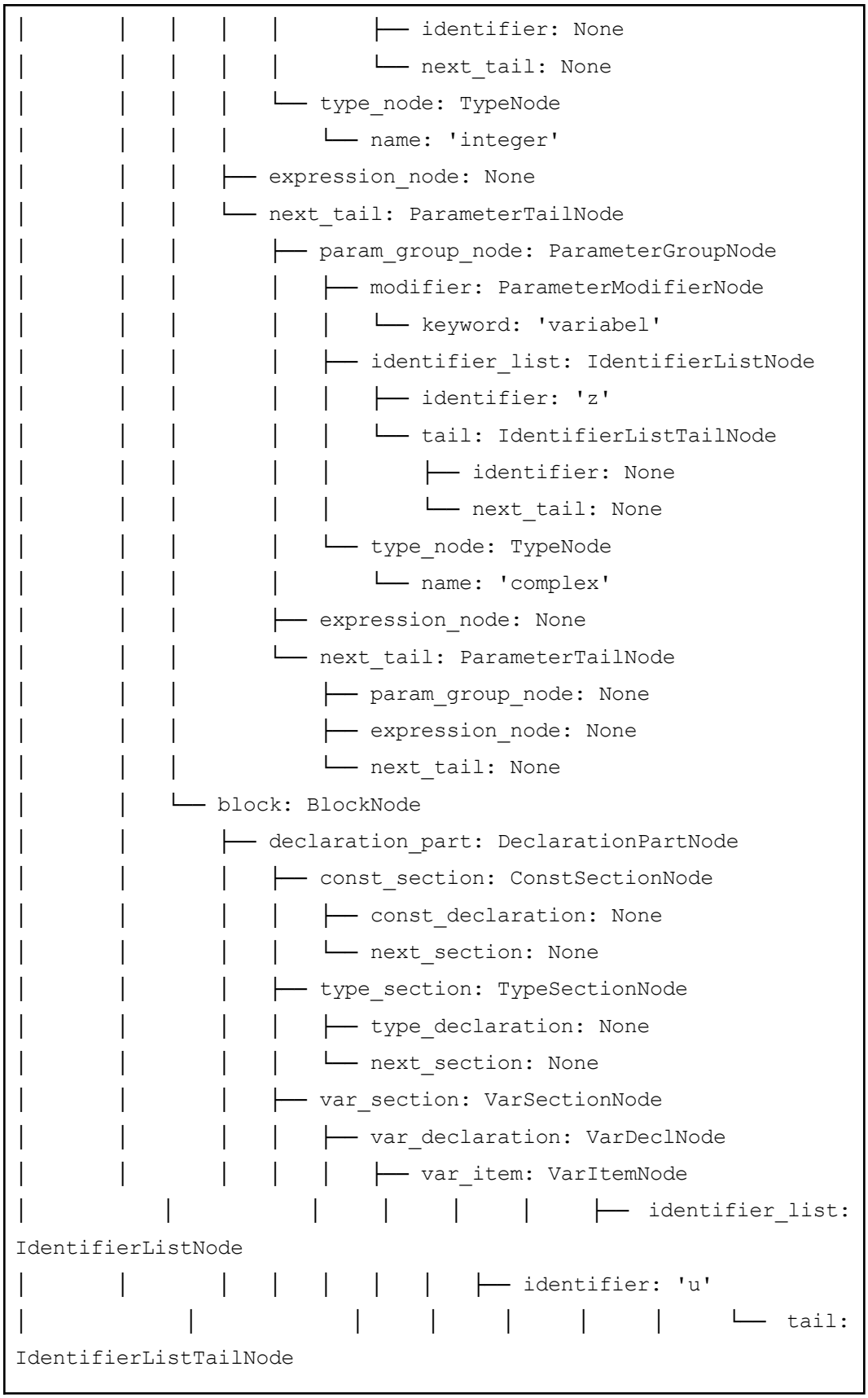

IdentifierListNode
identifier: 'matrix'
tail: IdentifierListTailNode
identifier: None
next_tail: None
type_node: ArrayTypeNode
range_node: RangeNode
expression_1: UnaryOpNode
operator: OperatorNode
lexeme: '-'
term_node: TermNode
factor: NumberNode
value: '3'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
factor_node: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
range_operator: OperatorNode
lexeme: '..'
expression_2: UnaryOpNode
operator: OperatorNode
lexeme: '+'
term_node: TermNode

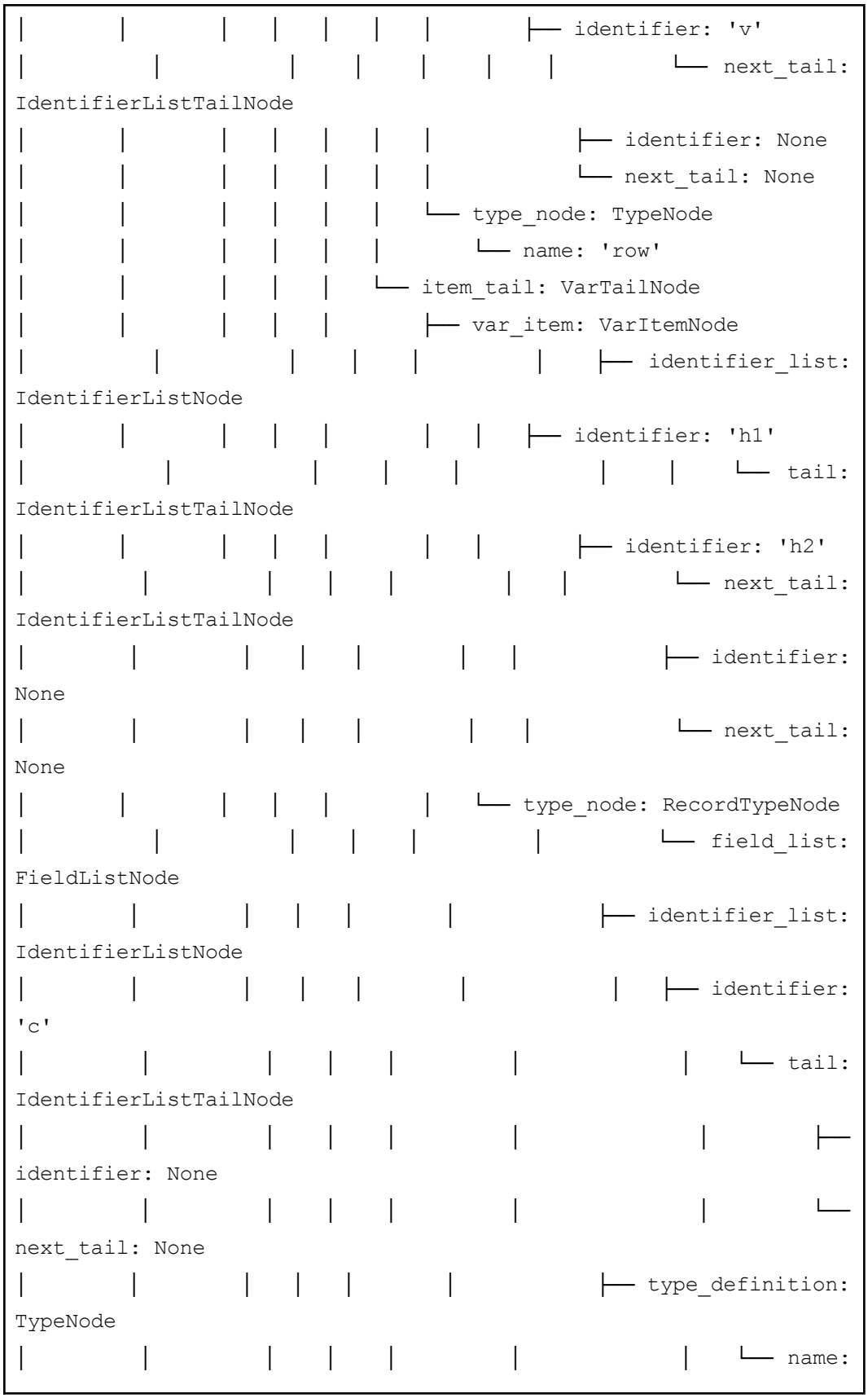

| | | | | | | ├── factor:
NumberNode
| | | | | | | | └── value: '3'
| | | | | | | └── tail:
TermTailNode
| | | | | | | ├──
multiplicative_operator: None
| | | | | | ├── factor: None
| | | | | | └── next_tail:
None
| | | | | ├── factor_node: None
| | | | | | └── tail:
SimpleExprTailNode
| | | | | | ├──
additive_operator: None
| | | | | ├── term: None
| | | | | └── next_tail: None
| | | | └── type_node: TypeNode
| | | | └── name: 'row'
| | | └── next_tail: VarTailNode
| | | ├── var_item: VarItemNode
| | | | ├── identifier_list:
IdentifierListNode
| | | | | ├── identifier: 'pattern'
| | | | | └── tail:
IdentifierListTailNode
| | | | | ├── identifier: None
| | | | | └── next_tail: None
| | | | └── type_node: ArrayTypeNode
| | | | ├── range_node: RangeNode
| | | | | ├── expression_1:
SimpleExprNode
| | | | | ├── term: TermNode
| | | | | | ├── factor:
NumberNode
| | | | | | | └── value:
'1'

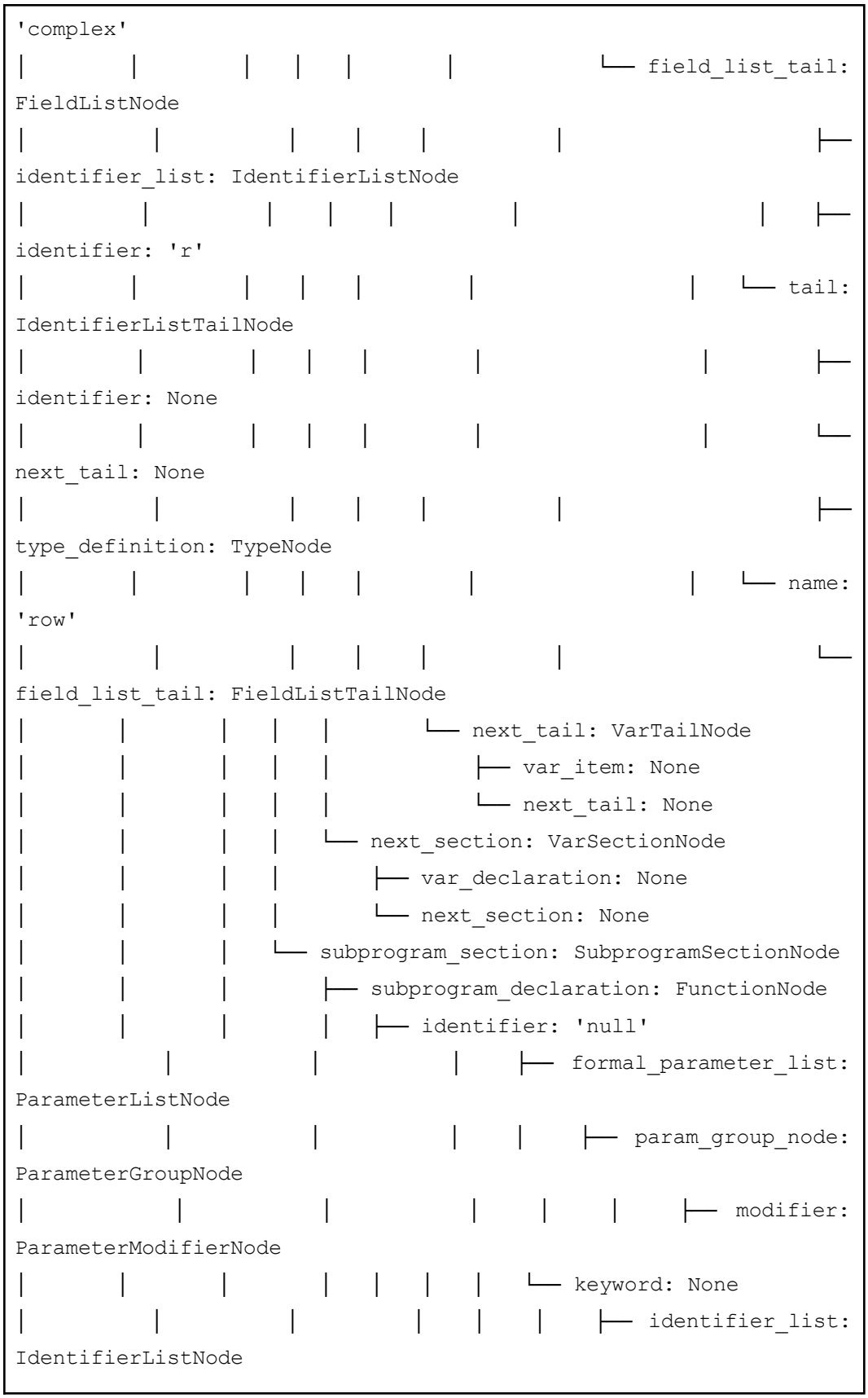

| | | | | | | └── tail:
TermTailNode
| | | | | | | ├──
multiplicative_operator: None
| | | | | | ├── factor:
None
| | | | | | └──
next_tail: None
| | | | | └── tail:
SimpleExprTailNode
| | | | | ├──
additive_operator: None
| | | | ├── term: None
| | | | └── next_tail:
None
| | | | ├── range_operator:
OperatorNode
| | | | └── lexeme: '..'
| | | | └── expression_2:
SimpleExprNode
| | | | ├── term: TermNode
| | | | ├── factor:
NumberNode
| | | | └── value:
'5'
| | | | └── tail:
TermTailNode
| | | | ├──
multiplicative_operator: None
| | | | ├── factor:
None
| | | | └──
next_tail: None
| | | └── tail:
SimpleExprTailNode
| | | ├──
additive_operator: None

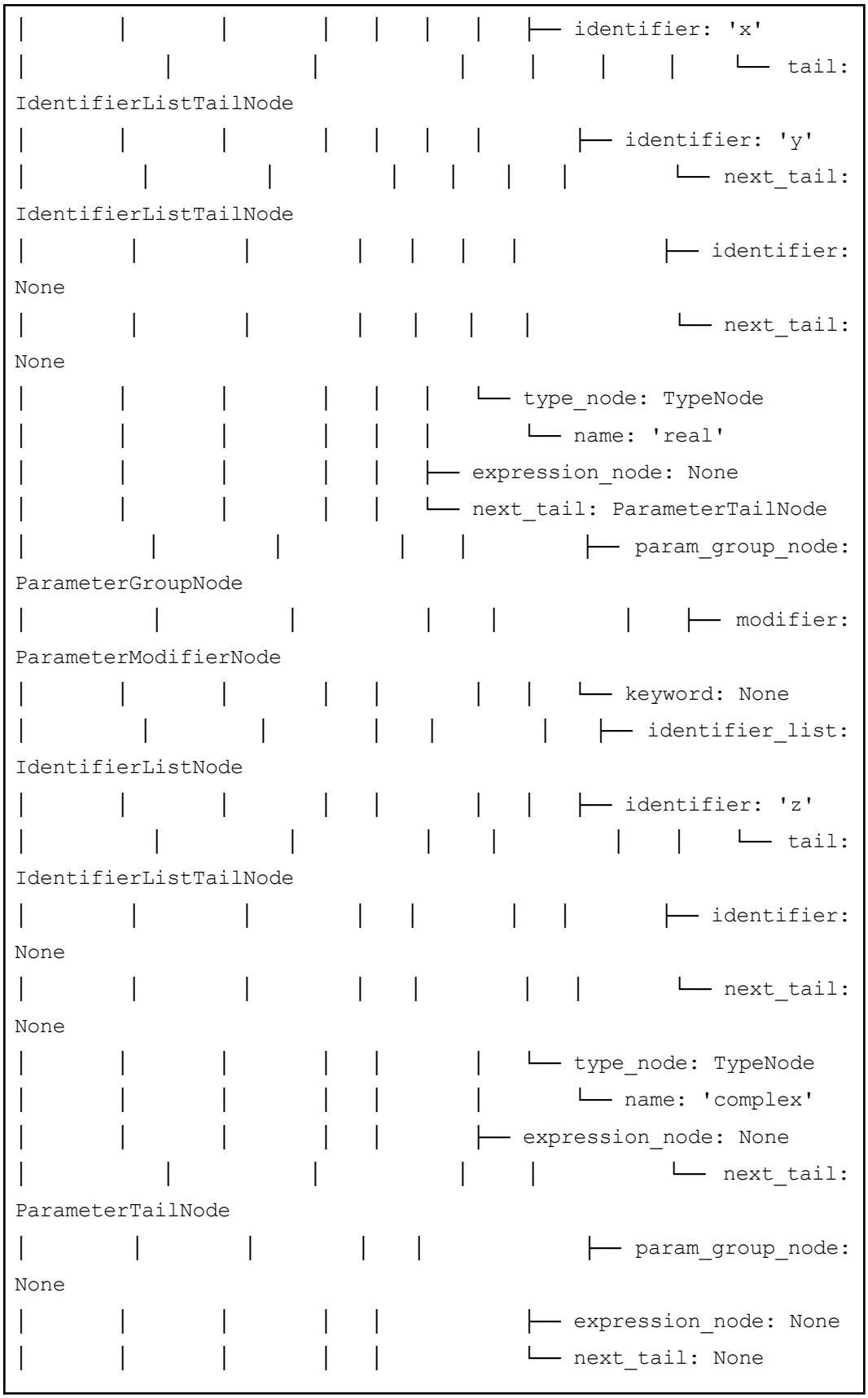

| | | | | | ├── term: None
| | | | | | └── next_tail:
None
| | | | | └── type_node: ArrayTypeNode
| | | | | ├── range_node: RangeNode
| | | | | | ├── expression_1:
SimpleExprNode
| | | | | | | ├── term:
TermNode
| | | | | | | | ├── factor:
NumberNode
| | | | | | | | | └──
value: '2'
| | | | | | | | └── tail:
TermTailNode
| | | | | | | | ├──
multiplicative_operator: None
| | | | | | | | ├──
factor: None
| | | | | | | | └──
next_tail: None
| | | | | | | └── tail:
SimpleExprTailNode
| | | | | | | ├──
additive_operator: None
| | | | | | | ├── term:
None
| | | | | | | └──
next_tail: None
| | | | | | ├── range_operator:
OperatorNode
| | | | | | | └── lexeme: '..'
| | | | | | └── expression_2:
SimpleExprNode
| | | | | | ├── term:
TermNode
| | | | | | | ├── factor:

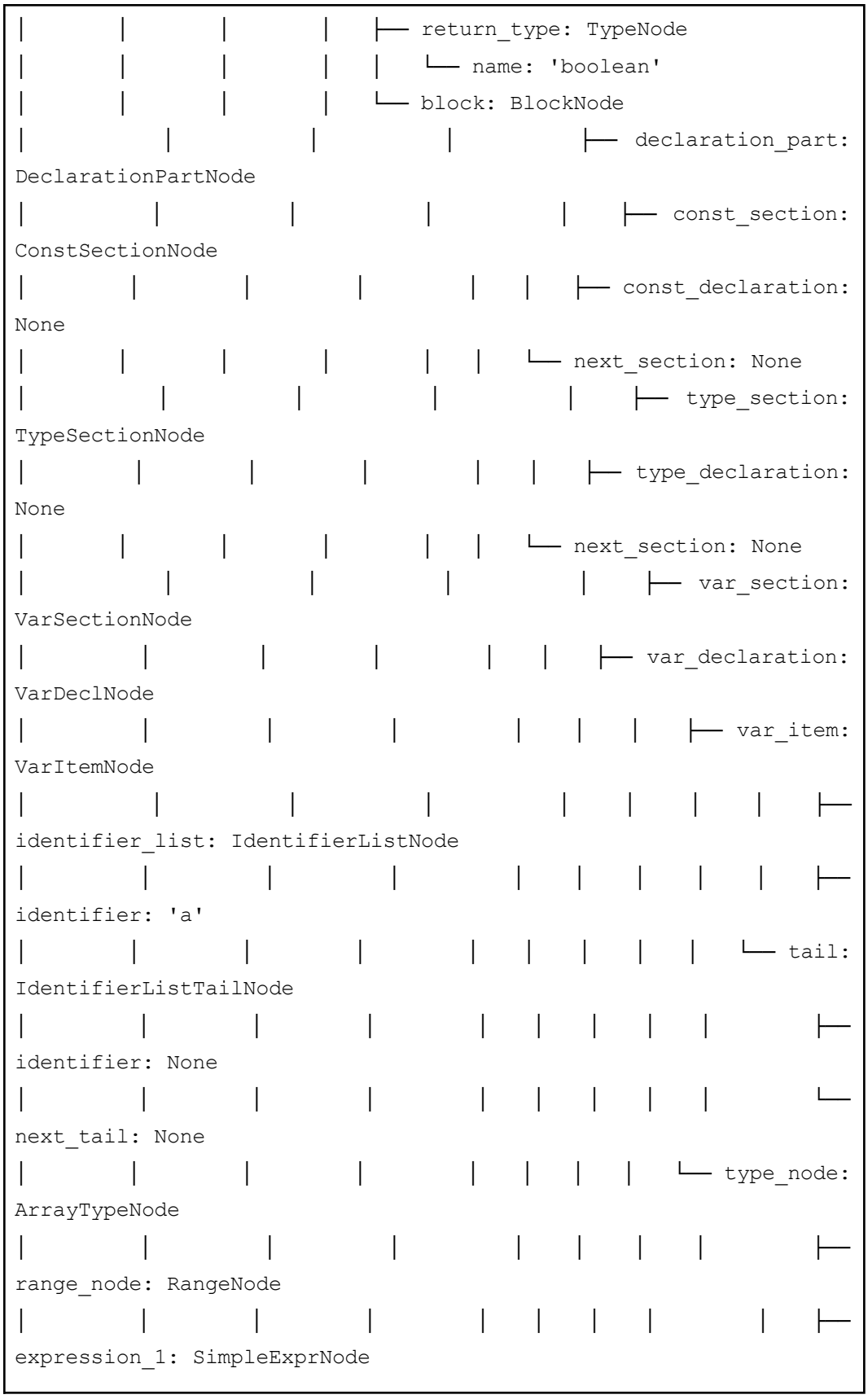

NumberNode
value: '5'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
type_node: TypeNode
name: 'char'
next_tail: VarTailNode
var_item: None
next_tail: None
next_section: VarSectionNode
var_declaration: None
next_section: None
subprogram_section: SubprogramSectionNode
subprogram_declaration: ProcedureNode
identifier: 'dummy'
formal_parameter_list: ParameterListNode
param_group_node: ParameterGroupNode
modifier: ParameterModifierNode
keyword: 'variabel'
identifier_list: IdentifierListNode
identifier: 'i'
tail: IdentifierListTailNode

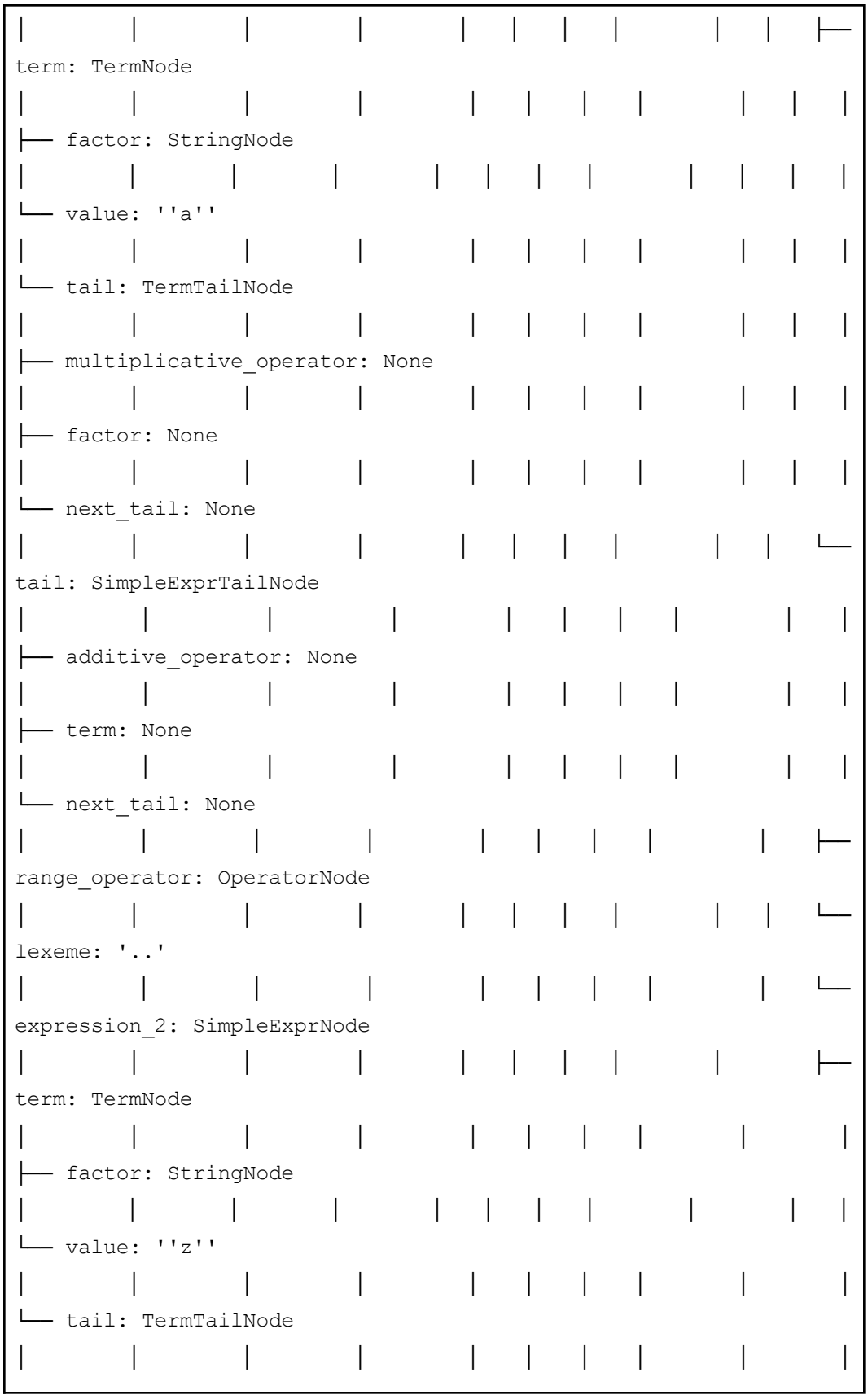

identifier: None
next_tail: None
type_node: TypeNode
name: 'integer'
expression_node: None
next_tail: ParameterTailNode
param_group_node: ParameterGroupNode
modifier: ParameterModifierNode
keyword: 'variabel'
identifier_list: IdentifierListNode
identifier: 'z'
tail: IdentifierListTailNode
identifier: None
next_tail: None
type_node: TypeNode
name: 'complex'
expression_node: None
next_tail: ParameterTailNode
param_group_node: None
expression_node: None
next_tail: None
block: BlockNode
declaration_part: DeclarationPartNode
const_section: ConstSectionNode
const_declaration: None
next_section: None
type_section: TypeSectionNode
type_declaration: None
next_section: None
var_section: VarSectionNode
var_declaration: VarDeclNode
var_item: VarItemNode
identifier_list: IdentifierListNode
identifier: 'u'
tail: IdentifierListTailNode

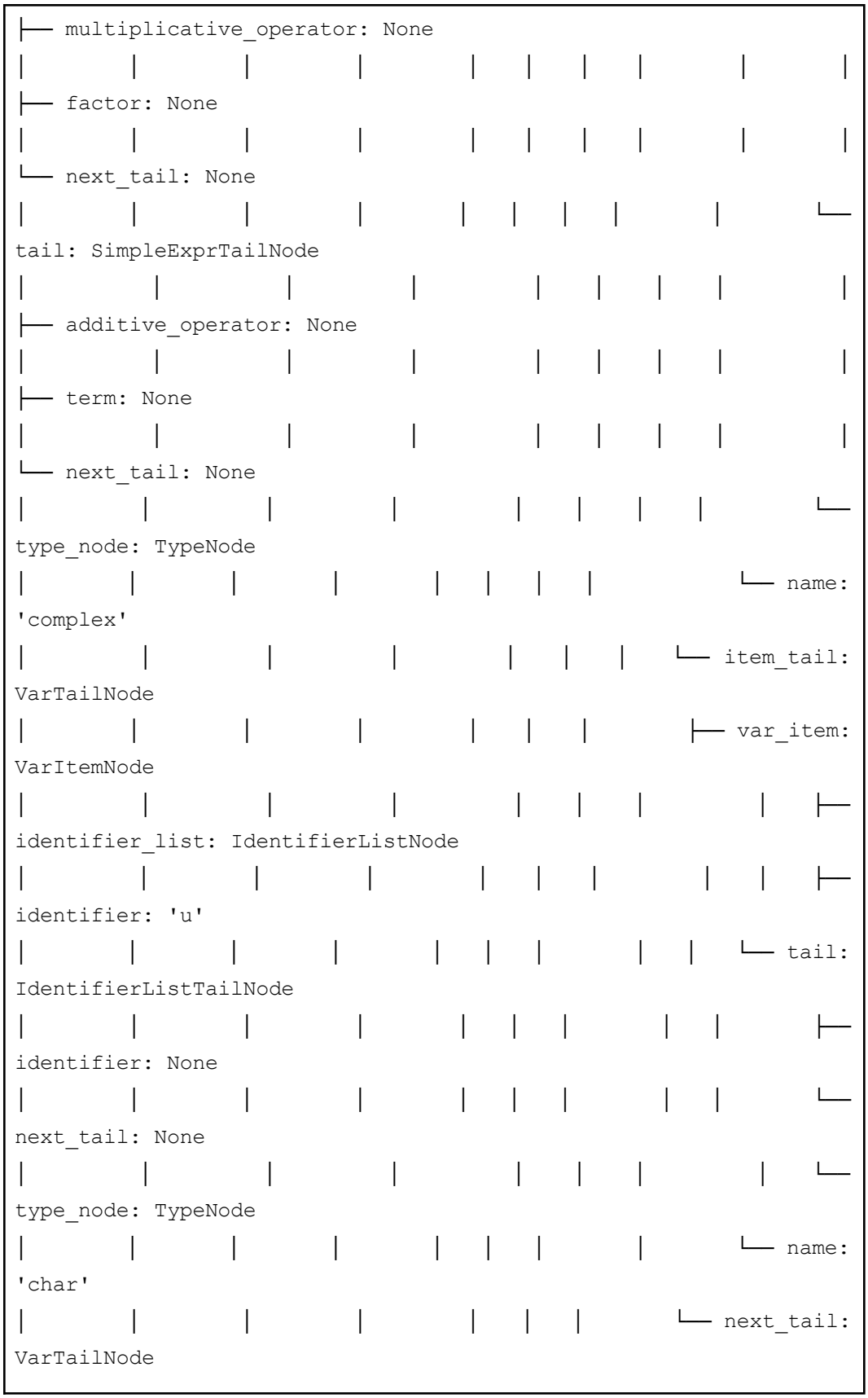

identifier: 'v'
next_tail:
IdentifierListTailNode
identifier: None
next_tail: None
type_node: TypeNode
name: 'row'
item_tail: VarTailNode
var_item: VarItemNode
identifier_list:
IdentifierListNode
identifier: 'h1'
tail:
IdentifierListTailNode
identifier: 'h2'
next_tail:
IdentifierListTailNode
identifier:
None
next_tail:
None
type_node: RecordTypeNode
field_list:
FieldListNode
identifier_list:
IdentifierListNode
identifier:
'c'
tail:
IdentifierListTailNode
identifier: None
next_tail: None
type_definition:
TypeNode
name:

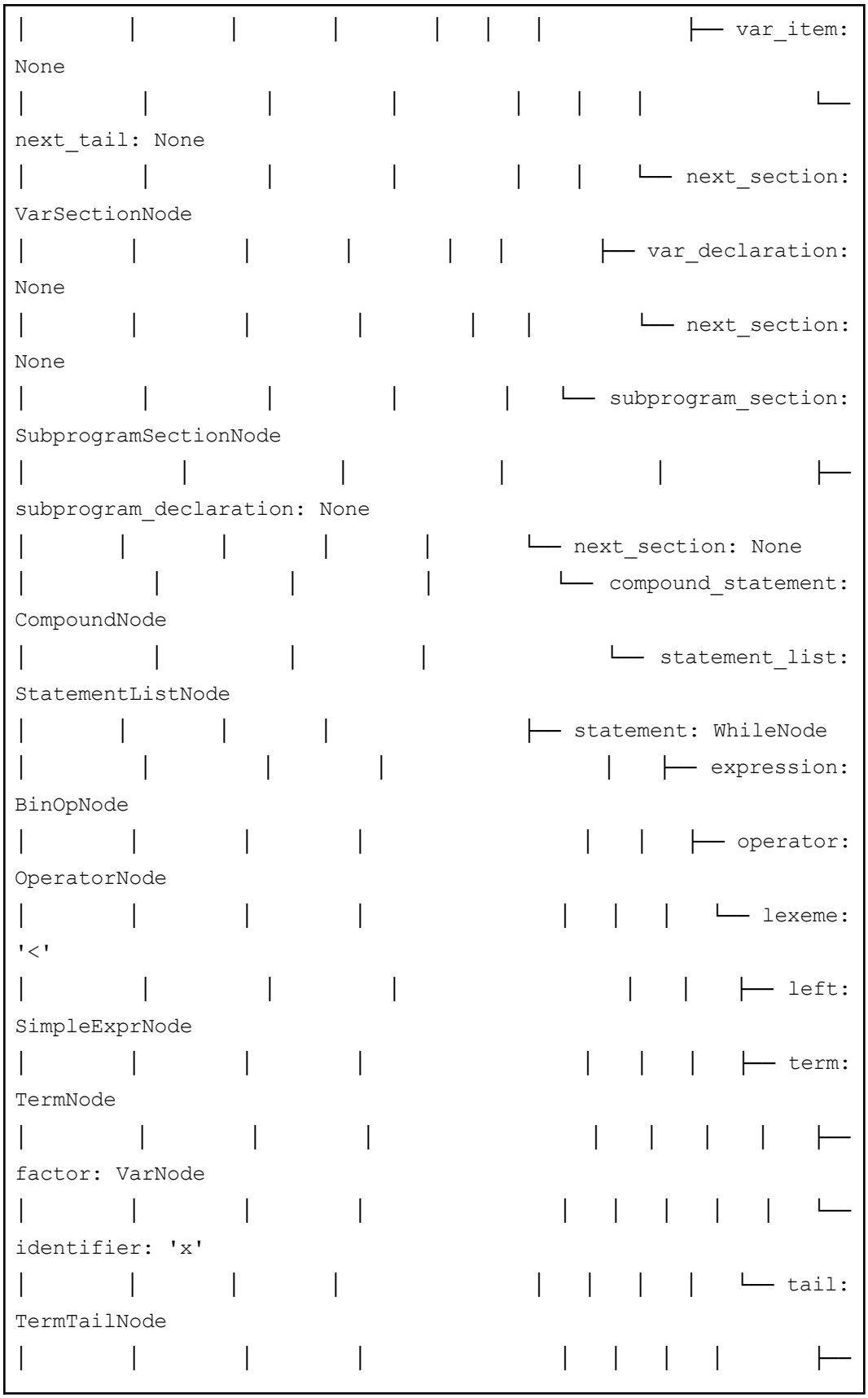

'complex'
└── field_list_tail: FieldListNode
├── identifier_list: IdentifierListNode
│ ├── identifier: 'r'
│ └── tail: IdentifierListTailNode
│ ├── identifier: None
│ └── next_tail: None
├── type_definition: TypeNode
│ └── name: 'row'
└── field_list_tail: FieldListTailNode
└── next_tail: VarTailNode
├── var_item: None
└── next_tail: None
└── next_section: VarSectionNode
├── var_declaration: None
└── next_section: None
└── subprogram_section: SubprogramSectionNode
├── subprogram_declaration: FunctionNode
│ ├── identifier: 'null'
│ ├── formal_parameter_list: ParameterListNode
│ ├── param_group_node: ParameterGroupNode
│ ├── modifier: ParameterModifierNode
│ └── keyword: None
├── identifier_list: IdentifierListNode

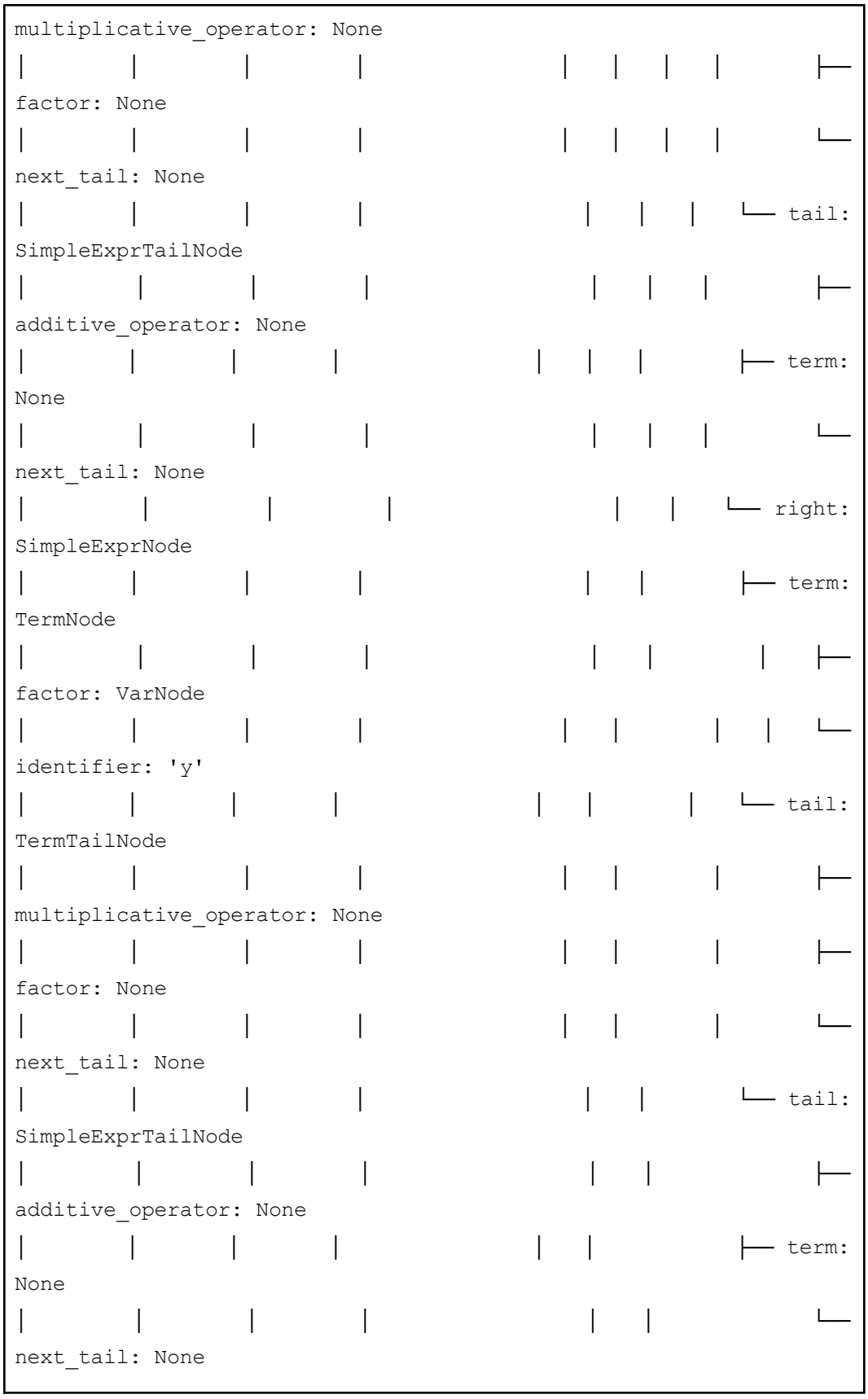

| | | | | | | ├── identifier: 'x'
| | | | | | | └── tail: IdentifierListTailNode
| | | | | | | ├── identifier: 'y'
| | | | | | | └── next_tail: IdentifierListTailNode
| | | | | | | ├── identifier: None
| | | | | | | └── next_tail: None
| | | | | | └── type_node: TypeNode
| | | | | | └── name: 'real'
| | | | | ├── expression_node: None
| | | | | └── next_tail: ParameterTailNode
| | | | | ├── param_group_node: ParameterGroupNode
| | | | | | ├── modifier: ParameterModifierNode
| | | | | | └── keyword: None
| | | | | | ├── identifier_list: IdentifierListNode
| | | | | | ├── identifier: 'z'
| | | | | | | └── tail: IdentifierListTailNode
| | | | | | ├── identifier: None
| | | | | | └── next_tail: None
| | | | | └── type_node: TypeNode
| | | | | └── name: 'complex'
| | | | ├── expression_node: None
| | | | └── next_tail: ParameterTailNode
| | | | ├── param_group_node: None
| | | | ├── expression_node: None
| | | | └── next_tail: None

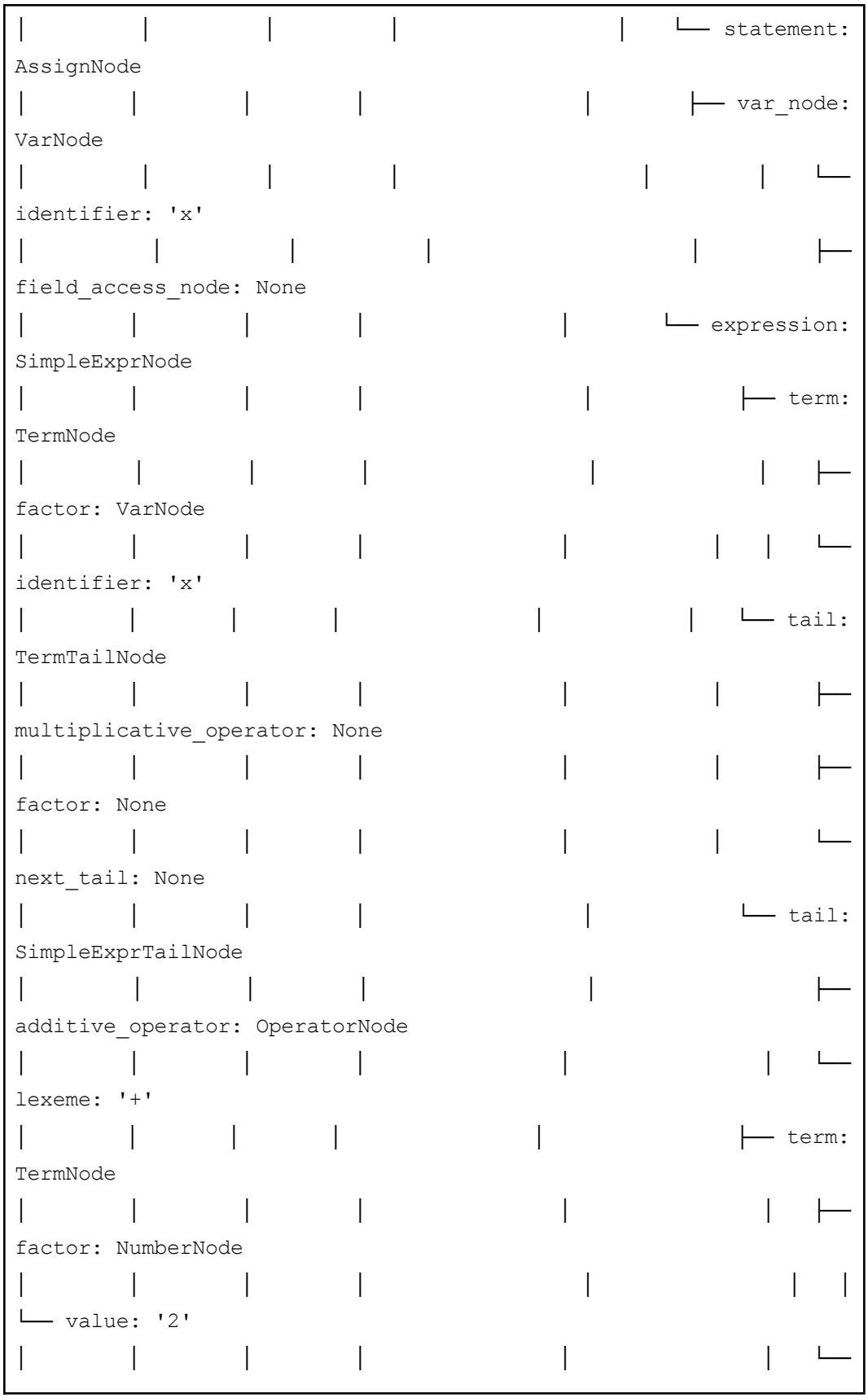

| | | | ├── return_type: TypeNode
| | | | │ └── name: 'boolean'
| | | | └── block: BlockNode
| | | | ├── declaration_part: DeclarationPartNode
| | | | | ├── const_section: ConstSectionNode
| | | | | | ├── const_declaration: None
| | | | | | └── next_section: None
| | | | | ├── type_section: TypeSectionNode
| | | | | | ├── type_declaration: None
| | | | | | └── next_section: None
| | | | | ├── var_section: VarSectionNode
| | | | | | ├── var_declaration: VarDeclNode
| | | | | | | ├── var_item: VarItemNode
| | | | | | | | ├── identifier_list: IdentifierListNode
| | | | | | | | | ├── identifier: 'a'
| | | | | | | | | └── tail: IdentifierListTailNode
| | | | | | | | | ├── identifier: None
| | | | | | | | | └── next_tail: None
| | | | | | | | └── type_node: ArrayTypeNode
| | | | | | | | ├── range_node: RangeNode
| | | | | | | | | ├── expression_1: SimpleExprNode

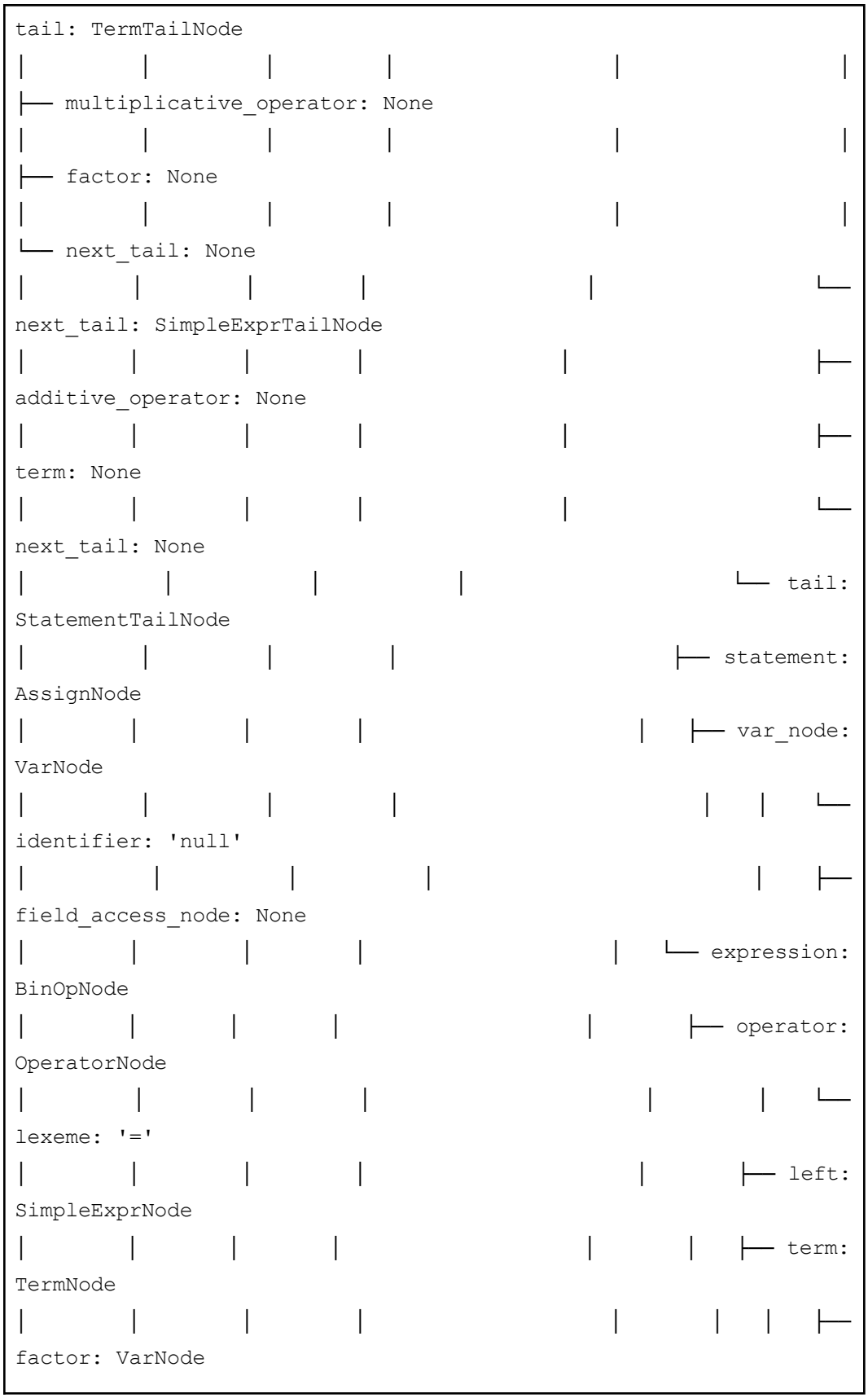

term: TermNode
├── factor: StringNode
└── value: ''a''
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
range_operator: OperatorNode
└── lexeme: '..'
expression_2: SimpleExprNode
term: TermNode
├── factor: StringNode
└── value: ''z''
└── tail: TermTailNode

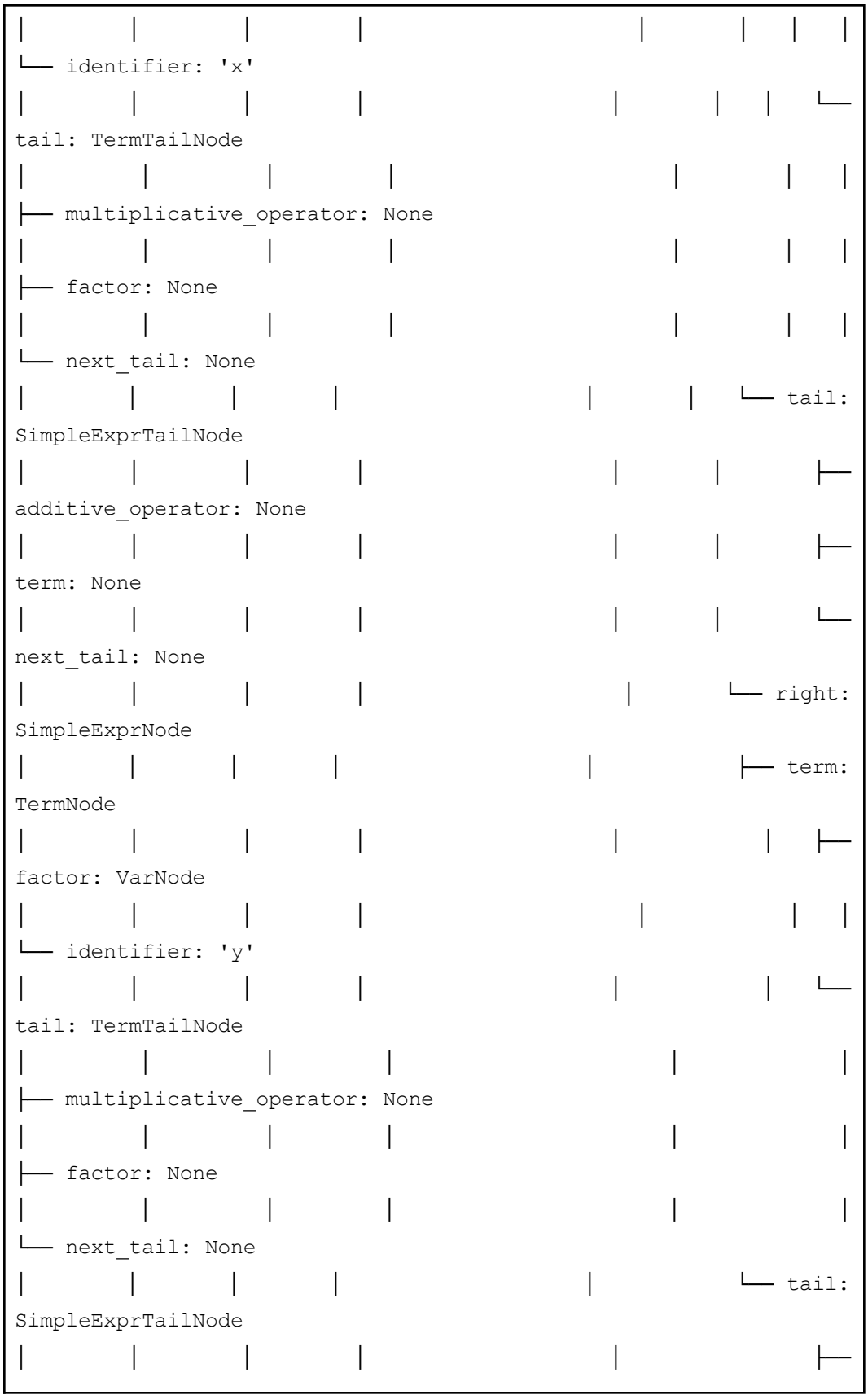

├── multiplicative_operator: None
│ │ │ │ │ │ │ │ │
├── factor: None
│ │ │ │ │ │ │ │ │
└── next_tail: None
│ │ │ │ │ │ │ │ └──
tail: SimpleExprTailNode
│ │ │ │ │ │ │ │
├── additive_operator: None
│ │ │ │ │ │ │ │
├── term: None
│ │ │ │ │ │ │ │
└── next_tail: None
│ │ │ │ │ │ │ └──
type_node: TypeNode
│ │ │ │ │ │ │ └── name:
'complex'
│ │ │ │ │ │ └── item_tail:
VarTailNode
│ │ │ │ │ │ ├── var_item:
VarItemNode
│ │ │ │ │ │ │ ├──
identifier_list: IdentifierListNode
│ │ │ │ │ │ │ │ ├──
identifier: 'u'
│ │ │ │ │ │ │ │ └── tail:
IdentifierListTailNode
│ │ │ │ │ │ │ │ ├──
identifier: None
│ │ │ │ │ │ │ │ └──
next_tail: None
│ │ │ │ │ │ │ └──
type_node: TypeNode
│ │ │ │ │ │ │ └── name:
'char'
│ │ │ │ │ │ └── next_tail:
VarTailNode

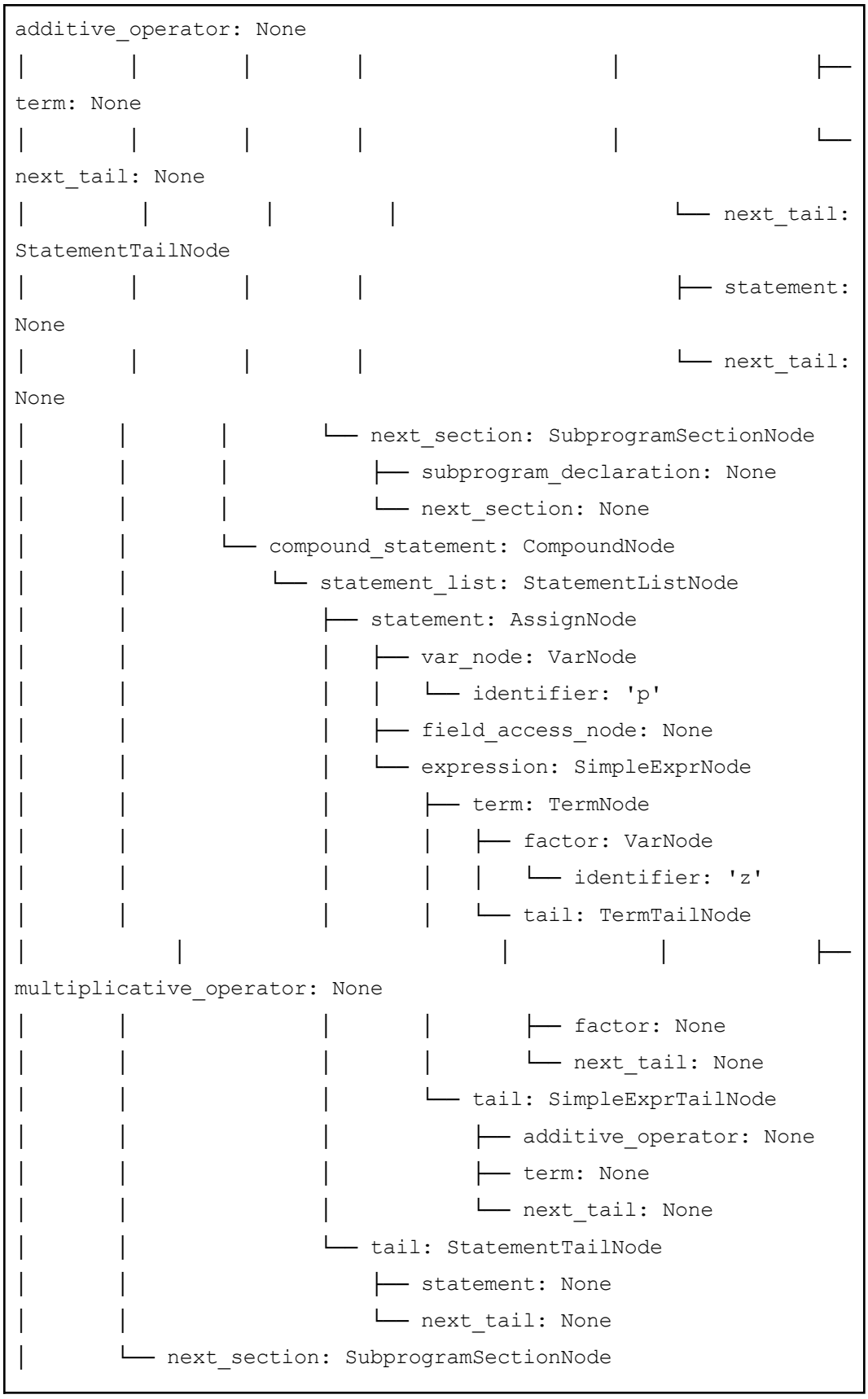

| | | | | | | ├── var_item: None
| | | | | | | └── next_tail: None
| | | | | | └── next_section: VarSectionNode
| | | | | | ├── var_declaration: None
| | | | | | └── next_section: None
| | | | | └── subprogram_section: SubprogramSectionNode
| | | | | | ├── subprogram_declaration: None
| | | | | └── next_section: None
| | | | └── compound_statement: CompoundNode
| | | | └── statement_list: StatementListNode
| | | | ├── statement: WhileNode
| | | | | ├── expression: BinOpNode
| | | | | | ├── operator: OperatorNode
| | | | | | | └── lexeme: '<'
| | | | | | ├── left: SimpleExprNode
| | | | | | | ├── term: TermNode
| | | | | | | | ├── factor: VarNode
| | | | | | | | | └── identifier: 'x'
| | | | | | | | └── tail: TermTailNode
| | | | | | | | ├──

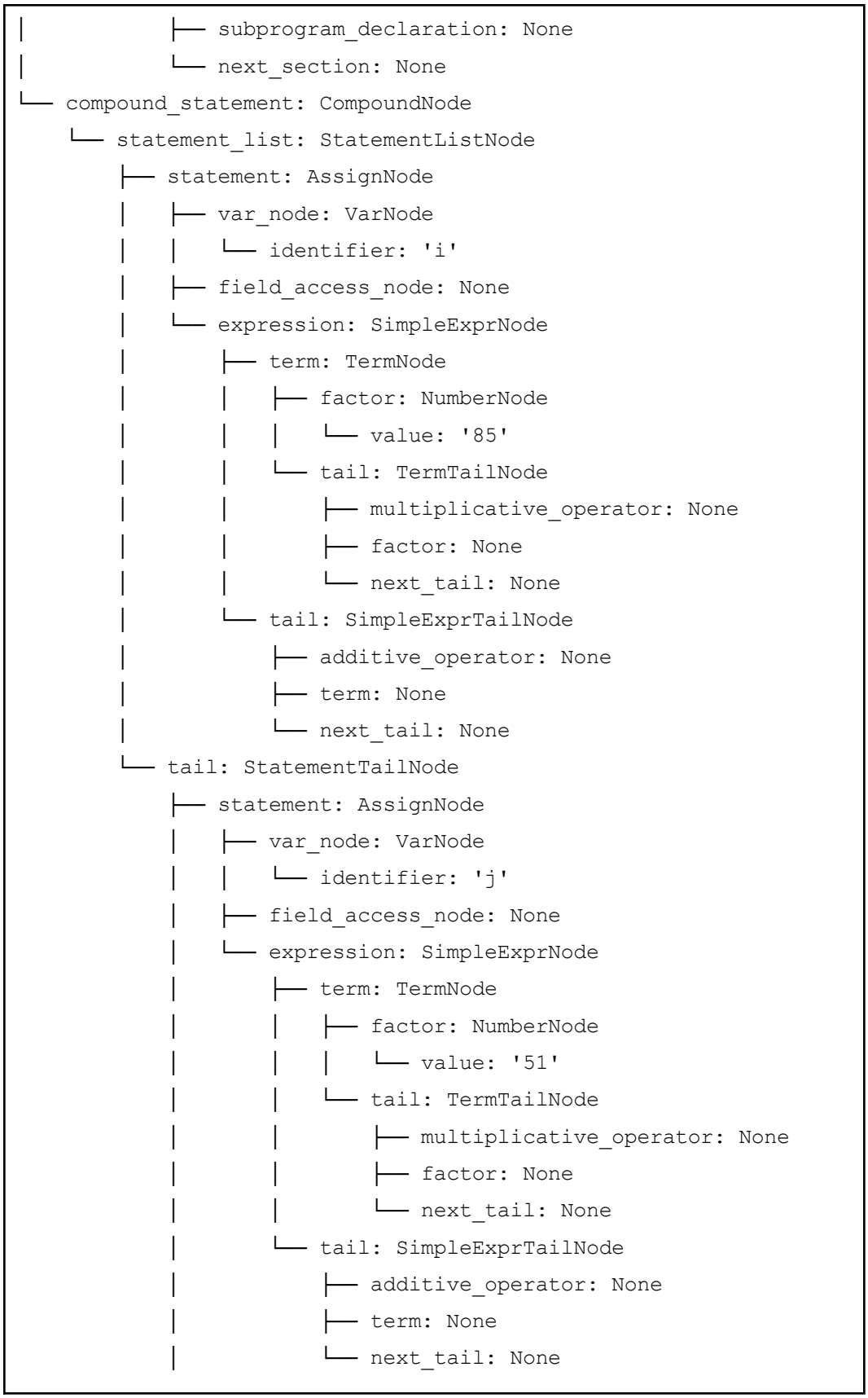

multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
right: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'y'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None

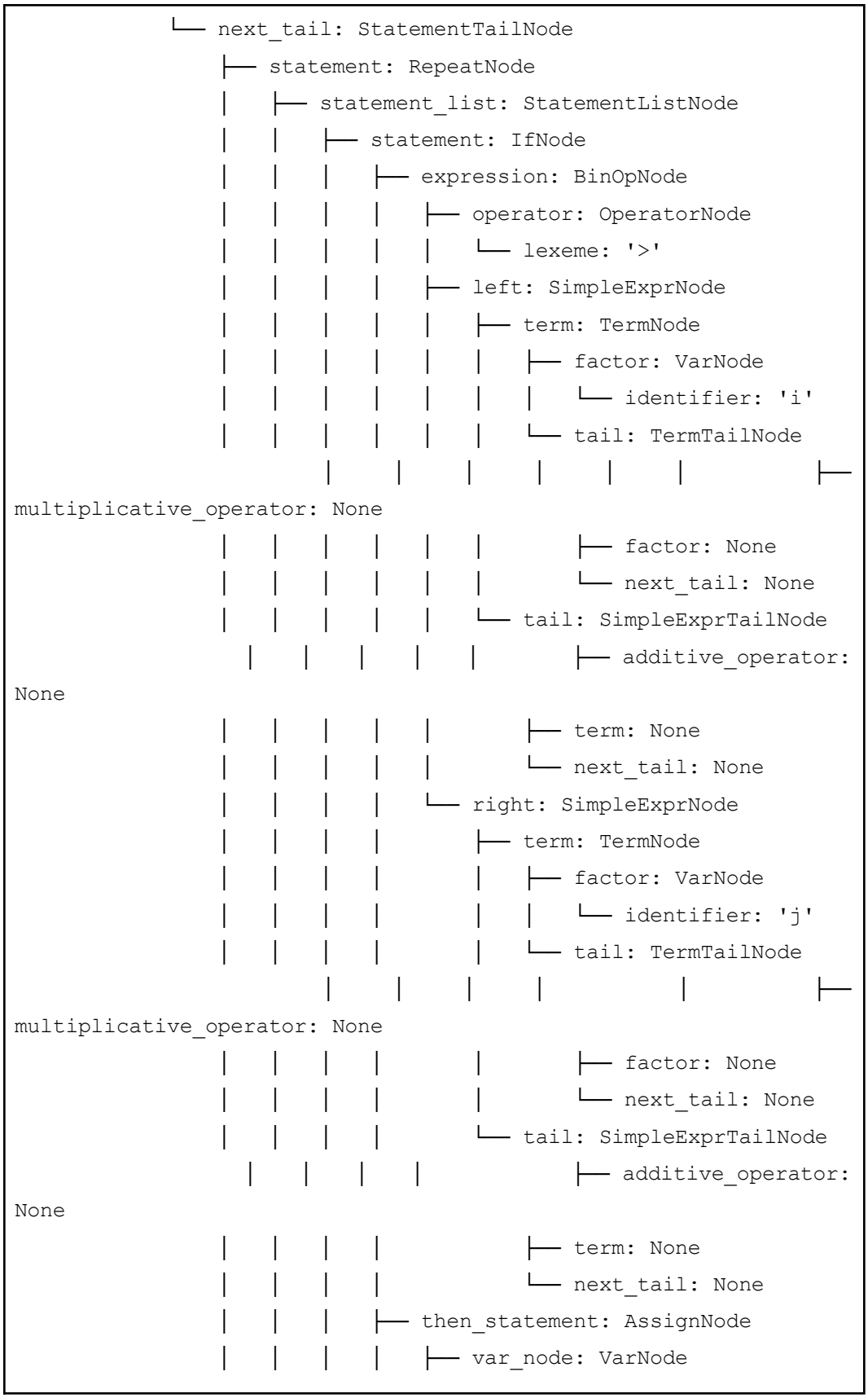

└── statement: AssignNode
├── var_node: VarNode
└── identifier: 'x'
├── field_access_node: None
└── expression: SimpleExprNode
├── term: TermNode
├── factor: VarNode
└── identifier: 'x'
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: OperatorNode
└── lexeme: '+'
├── term: TermNode
├── factor: NumberNode
└── value: '2'
└──

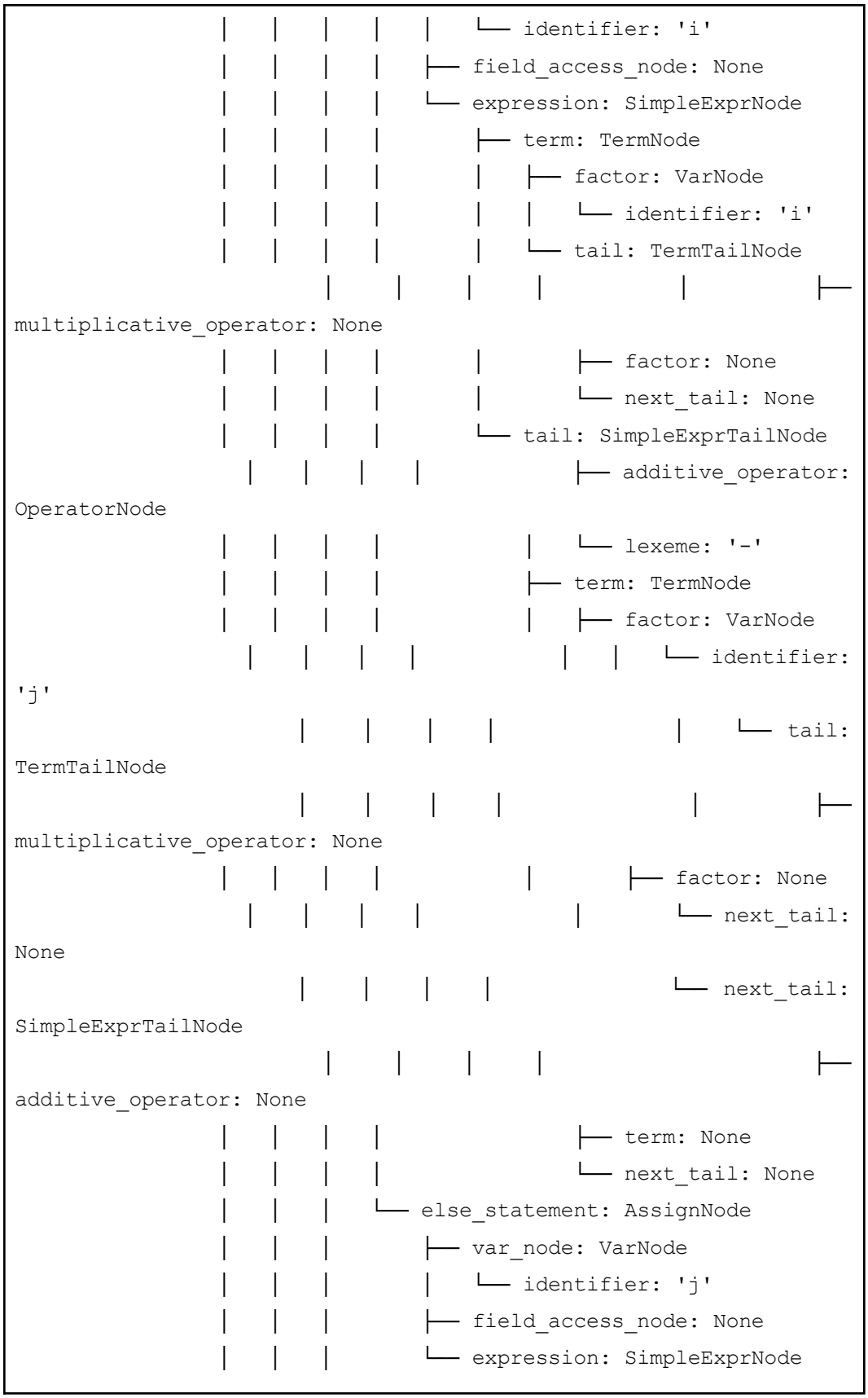

tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
next_tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
tail: StatementTailNode
statement: AssignNode
var_node: VarNode
identifier: 'null'
field_access_node: None
expression: BinOpNode
operator: OperatorNode
lexeme: '='
left: SimpleExprNode
term: TermNode
factor: VarNode

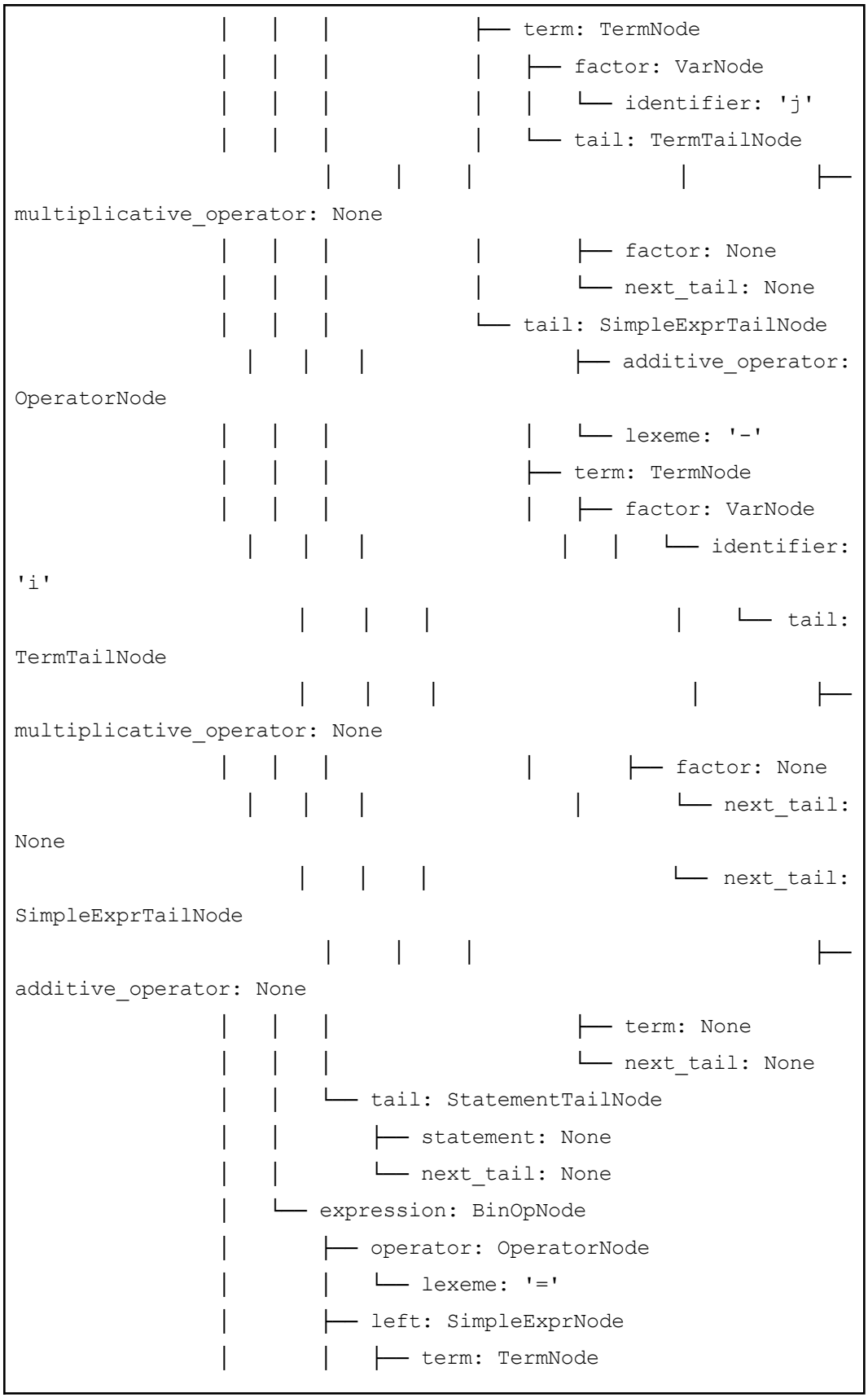

└── identifier: 'x'
tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail:
SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
└── right:
SimpleExprNode
├── term:
TermNode
factor: VarNode
└── identifier: 'y'
tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail:
SimpleExprTailNode
├──

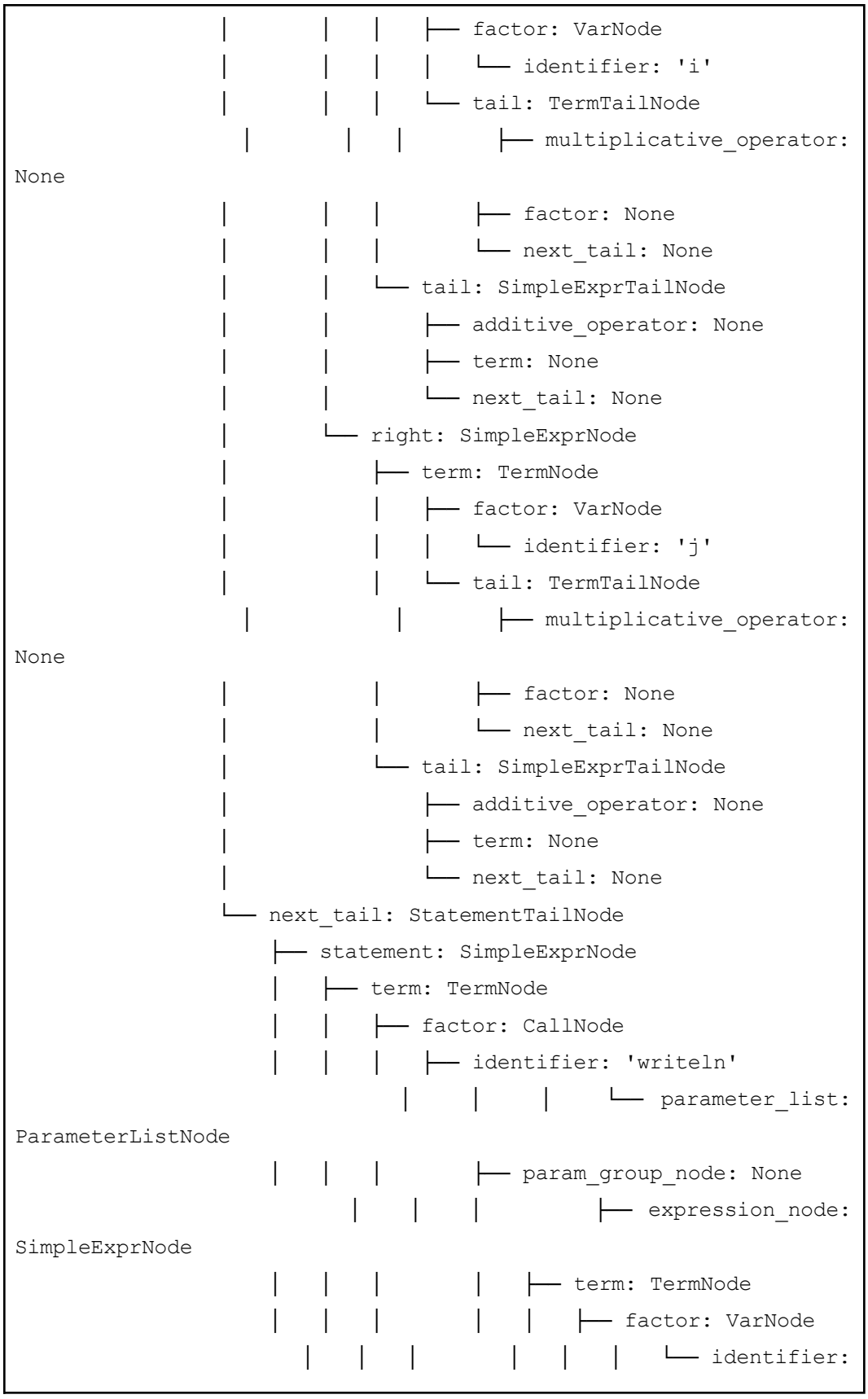

additive_operator: None
term: None
next_tail: None
next_tail: StatementTailNode
statement: None
next_tail: None
next_section: SubprogramSectionNode
subprogram_declaration: None
next_section: None
compound_statement: CompoundNode
statement_list: StatementListNode
statement: AssignNode
var_node: VarNode
identifier: 'p'
field_access_node: None
expression: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'z'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
tail: StatementTailNode
statement: None
next_tail: None
next_section: SubprogramSectionNode


├── subprogram_declaration: None
└── next_section: None
└── compound_statement: CompoundNode
└── statement_list: StatementListNode
├── statement: AssignNode
│ ├── var_node: VarNode
│ │ └── identifier: 'i'
│ ├── field_access_node: None
│ └── expression: SimpleExprNode
│ ├── term: TermNode
│ │ ├── factor: NumberNode
│ │ │ └── value: '85'
│ │ └── tail: TermTailNode
│ │ ├── multiplicative_operator: None
│ │ ├── factor: None
│ │ └── next_tail: None
│ └── tail: SimpleExprTailNode
│ ├── additive_operator: None
│ ├── term: None
│ └── next_tail: None
└── tail: StatementTailNode
├── statement: AssignNode
│ ├── var_node: VarNode
│ │ └── identifier: 'j'
│ ├── field_access_node: None
│ └── expression: SimpleExprNode
│ ├── term: TermNode
│ │ ├── factor: NumberNode
│ │ │ └── value: '51'
│ │ └── tail: TermTailNode
│ │ ├── multiplicative_operator: None
│ │ ├── factor: None
│ │ └── next_tail: None
│ └── tail: SimpleExprTailNode
│ ├── additive_operator: None
│ ├── term: None
│ └── next_tail: None


└── next_tail: StatementTailNode
├── statement: RepeatNode
│ ├── statement_list: StatementListNode
│ │ ├── statement: IfNode
│ │ │ ├── expression: BinOpNode
│ │ │ │ ├── operator: OperatorNode
│ │ │ │ │ └── lexeme: '>'
│ │ │ │ ├── left: SimpleExprNode
│ │ │ │ │ ├── term: TermNode
│ │ │ │ │ │ ├── factor: VarNode
│ │ │ │ │ │ │ └── identifier: 'i'
│ │ │ │ │ │ └── tail: TermTailNode
multiplicative_operator: None
│ │ │ │ │ │ ├── factor: None
│ │ │ │ │ │ └── next_tail: None
│ │ │ │ │ └── tail: SimpleExprTailNode
None
│ │ │ │ │ ├── additive_operator:
│ │ │ │ │ ├── term: None
│ │ │ │ │ └── next_tail: None
│ │ │ │ └── right: SimpleExprNode
│ │ │ │ ├── term: TermNode
│ │ │ │ │ ├── factor: VarNode
│ │ │ │ │ │ └── identifier: 'j'
│ │ │ │ │ └── tail: TermTailNode
multiplicative_operator: None
│ │ │ │ │ ├── factor: None
│ │ │ │ │ └── next_tail: None
│ │ │ │ └── tail: SimpleExprTailNode
None
│ │ │ │ ├── additive_operator:
│ │ │ │ ├── term: None
│ │ │ │ └── next_tail: None
│ │ │ ├── then_statement: AssignNode
│ │ │ ├── var_node: VarNode


└── identifier: 'i'
├── field_access_node: None
└── expression: SimpleExprNode
├── term: TermNode
├── factor: VarNode
└── identifier: 'i'
└── tail: TermTailNode
multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: OperatorNode
└── lexeme: '-'
├── term: TermNode
├── factor: VarNode
└── identifier: 'j'
└── tail: TermTailNode
multiplicative_operator: None
├── factor: None
└── next_tail: None
└── next_tail: SimpleExprTailNode
additive_operator: None
├── term: None
└── next_tail: None
└── else_statement: AssignNode
├── var_node: VarNode
└── identifier: 'j'
├── field_access_node: None
└── expression: SimpleExprNode


│ │ │ ├── term: TermNode
│ │ │ │ ├── factor: VarNode
│ │ │ │ │ └── identifier: 'j'
│ │ │ │ └── tail: TermTailNode
│ │ │ │ ├── multiplicative_operator: None
│ │ │ │ ├── factor: None
│ │ │ │ └── next_tail: None
│ │ │ └── tail: SimpleExprTailNode
│ │ │ ├── additive_operator: OperatorNode
│ │ │ └── lexeme: '-'
│ │ │ ├── term: TermNode
│ │ │ ├── factor: VarNode
│ │ │ └── identifier: 'i'
│ │ │ └── tail: TermTailNode
│ │ │ ├── multiplicative_operator: None
│ │ │ ├── factor: None
│ │ │ └── next_tail: None
│ │ └── next_tail: SimpleExprTailNode
│ │ ├── additive_operator: None
│ │ ├── term: None
│ │ └── next_tail: None
│ └── tail: StatementTailNode
│ ├── statement: None
│ └── next_tail: None
└── expression: BinOpNode
├── operator: OperatorNode
│ └── lexeme: '='
├── left: SimpleExprNode
│ ├── term: TermNode

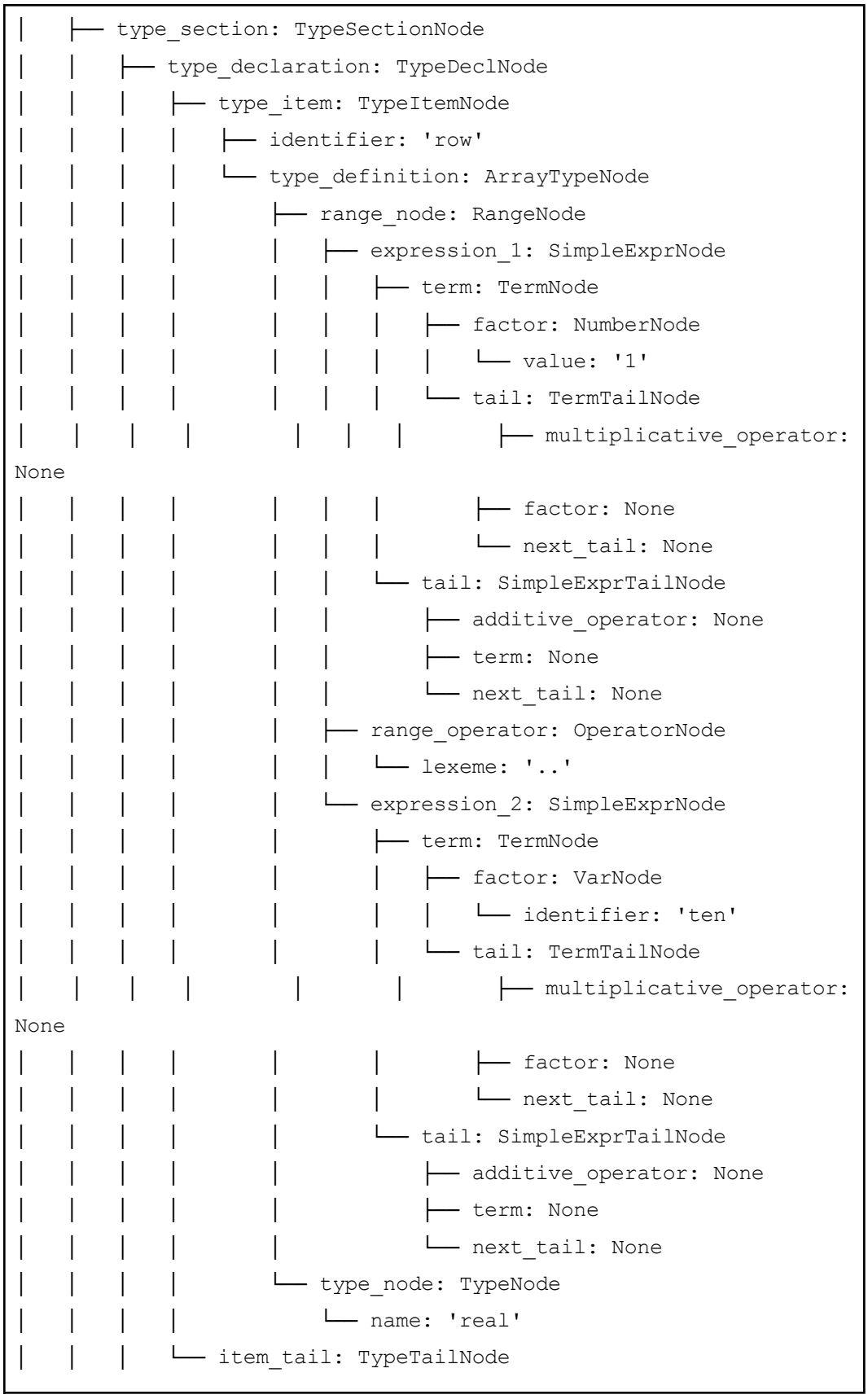

│ │ │ ├── factor: VarNode
│ │ │ │ └── identifier: 'i'
│ │ │ └── tail: TermTailNode
│ │ │ ├── multiplicative_operator:
None
│ │ │ ├── factor: None
│ │ │ └── next_tail: None
│ │ └── tail: SimpleExprTailNode
│ │ ├── additive_operator: None
│ │ ├── term: None
│ │ └── next_tail: None
│ └── right: SimpleExprNode
│ ├── term: TermNode
│ │ ├── factor: VarNode
│ │ │ └── identifier: 'j'
│ │ └── tail: TermTailNode
│ │ ├── multiplicative_operator:
None
│ │ ├── factor: None
│ │ └── next_tail: None
│ └── tail: SimpleExprTailNode
│ ├── additive_operator: None
│ ├── term: None
│ └── next_tail: None
└── next_tail: StatementTailNode
├── statement: SimpleExprNode
│ ├── term: TermNode
│ │ ├── factor: CallNode
│ │ │ ├── identifier: 'writeln'
│ │ │ └── parameter_list:
ParameterListNode
│ │ ├── param_group_node: None
│ │ ├── expression_node:
SimpleExprNode
│ │ ├── term: TermNode
│ │ │ ├── factor: VarNode
│ │ │ └── identifier:

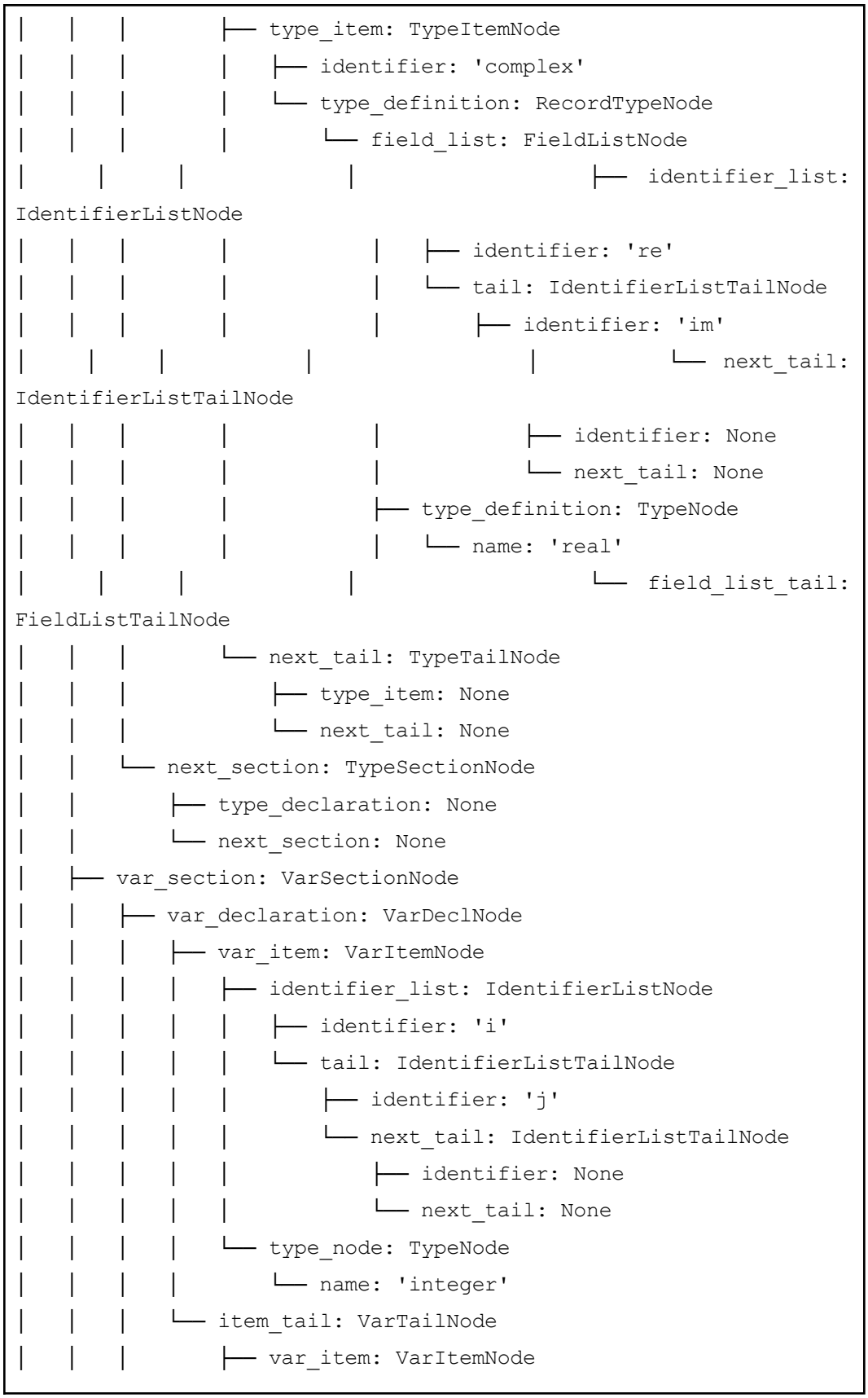
```
'i'
                          |   |   |       |   |   └── tail:
TermTailNode
                          |   |   |       |   |       ├──
multiplicative_operator: None
                    |   |   |       |   |       ├── factor: None
                      |   |   |       |   |       └── next_tail:
None
                            |   |   |       |   └── tail:
SimpleExprTailNode
                            |   |   |       |       ├──
additive_operator: None
                    |   |   |       |       ├── term: None
                    |   |   |       |       └── next_tail: None
                            |   |   |           └── next_tail:
ParameterTailNode
                      |   |   |           ├── param_group_node:
None
                    |   |   |           ├── expression_node: None
                    |   |   |           └── next_tail: None
                    |   |   └── tail: TermTailNode
                    |   |       ├── multiplicative_operator: None
                    |   |       ├── factor: None
                    |   |       └── next_tail: None
                    |   └── tail: SimpleExprTailNode
                    |       ├── additive_operator: None
                    |       ├── term: None
                    |       └── next_tail: None
                    └── next_tail: StatementTailNode
                        ├── statement: None
                        └── next_tail: None

================================================
Tahap: Semantic Analysis (Symbol Tables)
================================================
Semantic Error: Type mismatch: cannot assign complex to boolean
```

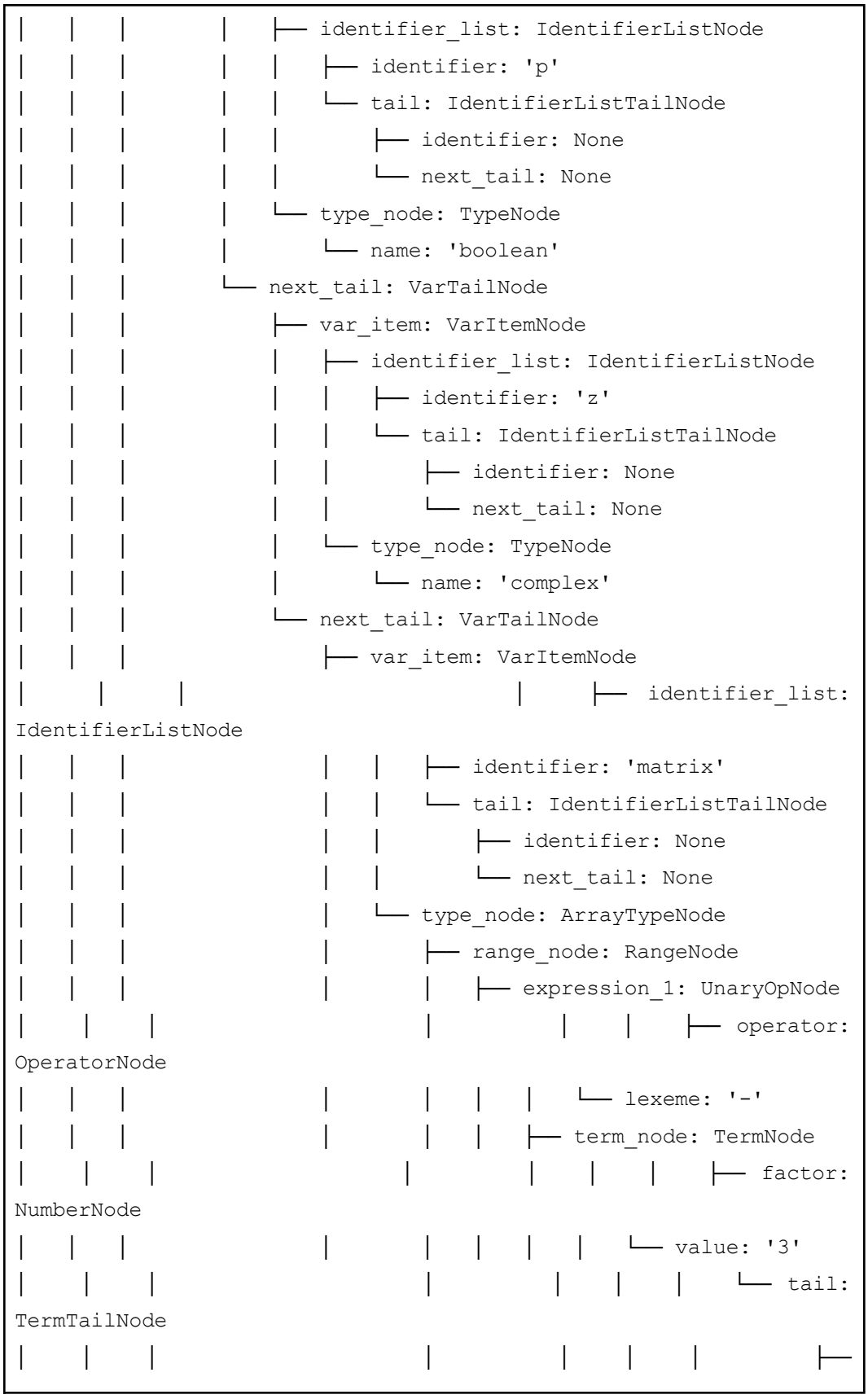
Pembahasan: Pada pengujian ini, Semantic Analyzer berhasil mendeteksi ketidaksesuaian tipe data pada operasi assignment. AST yang terbentuk menunjukkan struktur program yang valid secara sintaksis, namun pada tahap analisis semantik, ditemukan upaya pengisian nilai (assignment) variabel bertipe boolean dengan nilai bertipe complex. Hal ini melanggar aturan tipe data (strong typing) yang diterapkan dalam bahasa Pascal.

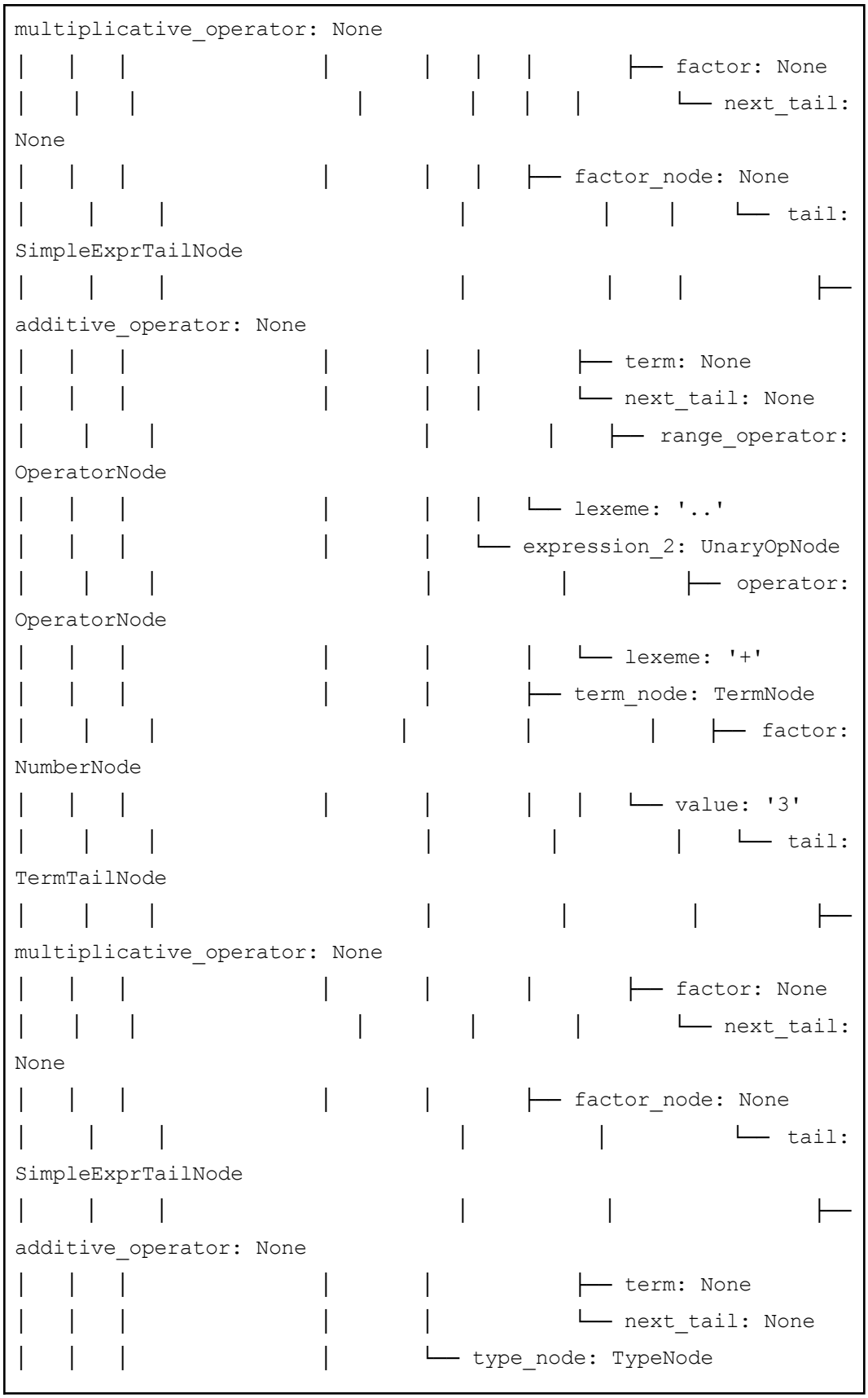
3.2.5 Hasil Uji 5

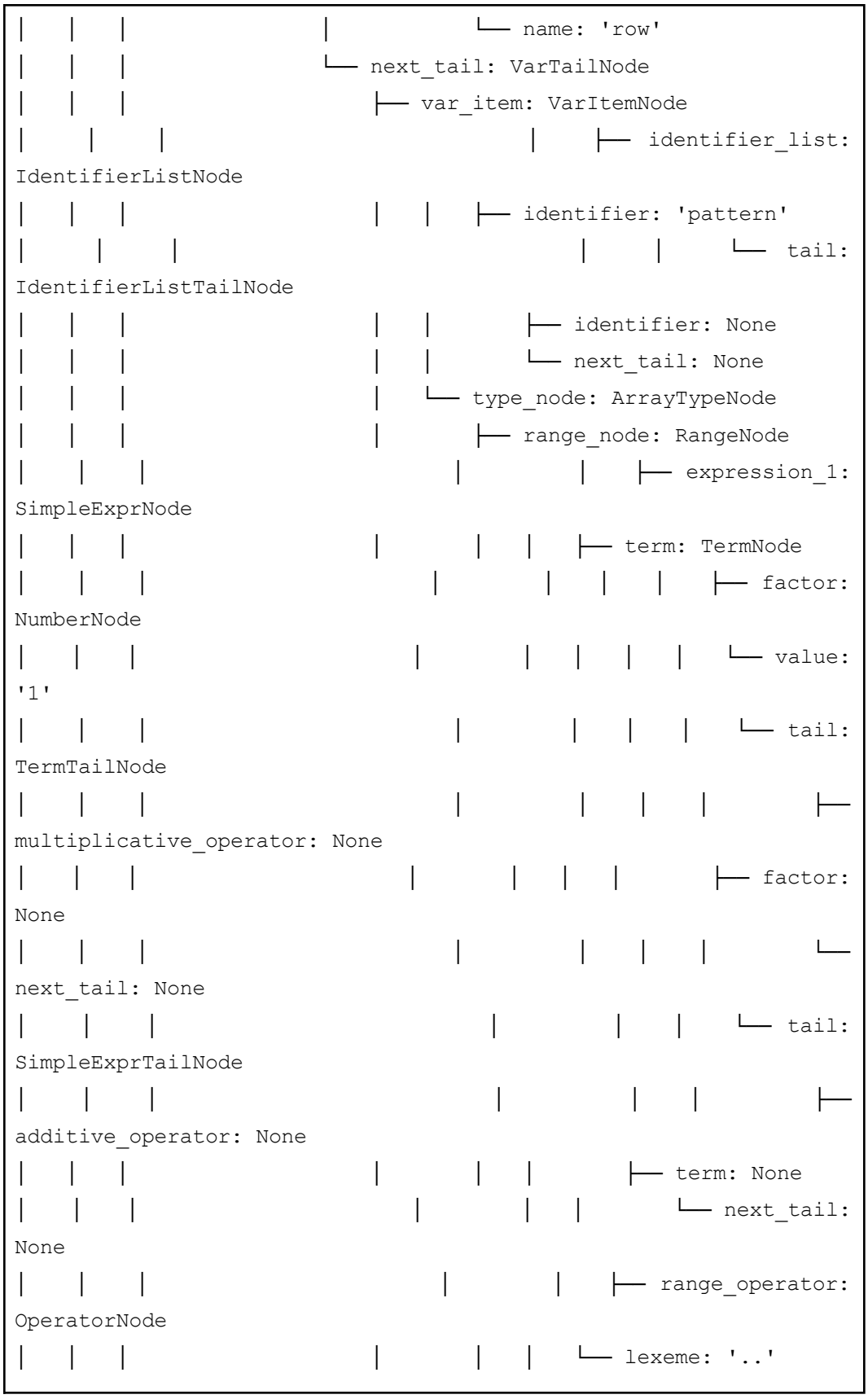
```
==================================================
Tahap: AST Generation (Abstract Syntax Tree)
==================================================
AST Structure:
------------------------------
ProgramNode
├── program_header: ProgramHeaderNode
│   └── identifier: 'test0'
├── declaration_part: DeclarationPartNode
│   ├── const_section: ConstSectionNode
│   │   ├── const_declaration: ConstDeclNode
│   │   │   ├── const_item: ConstItemNode
│   │   │   │   ├── identifier: 'ten'
│   │   │   │   ├── assign_operator: '='
│   │   │   │   └── value_node: NumberNode
│   │   │   │       └── value: '10'
│   │   │   └── item_tail: ConstTailNode
│   │   │       ├── const_item: ConstItemNode
│   │   │       │   ├── identifier: 'plus'
│   │   │       │   ├── assign_operator: '='
│   │   │       │   └── value_node: StringNode
│   │   │       │       └── value: ''+''
│   │   │       └── next_tail: ConstTailNode
│   │   │           ├── const_item: None
│   │   │           └── next_tail: None
│   │   └── next_section: ConstSectionNode
│   │       ├── const_declaration: None
│   │       └── next_section: None
```


type_section: TypeSectionNode
type_declaration: TypeDeclNode
type_item: TypeItemNode
identifier: 'row'
type_definition: ArrayTypeNode
range_node: RangeNode
expression_1: SimpleExprNode
term: TermNode
factor: NumberNode
value: '1'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
range_operator: OperatorNode
lexeme: '..'
expression_2: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'ten'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
type_node: TypeNode
name: 'real'
item_tail: TypeTailNode

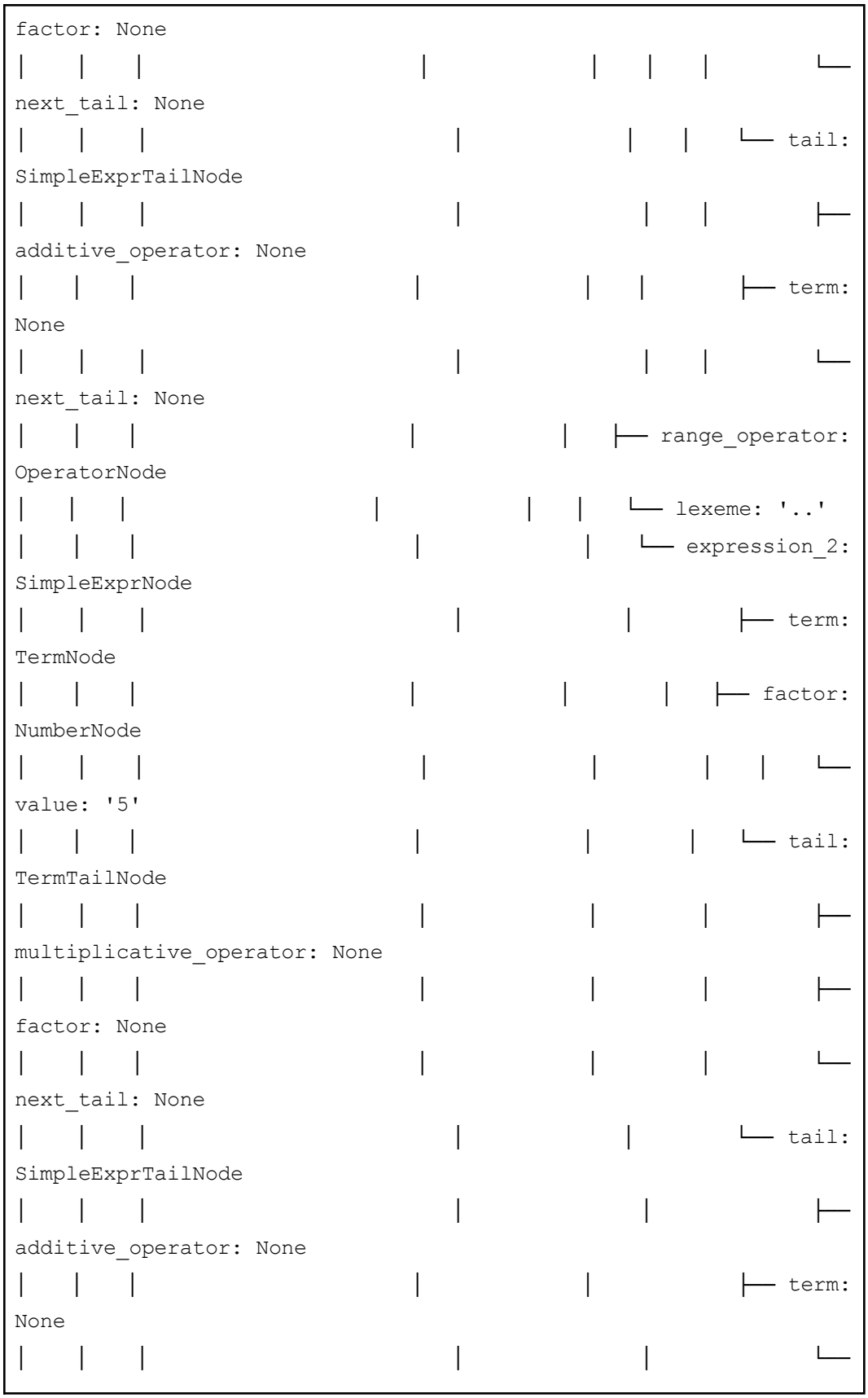

type_item: TypeItemNode
identifier: 'complex'
type_definition: RecordTypeNode
field_list: FieldListNode
identifier_list: IdentifierListNode
identifier: 're'
tail: IdentifierListTailNode
identifier: 'im'
next_tail: IdentifierListTailNode
identifier: None
next_tail: None
type_definition: TypeNode
name: 'real'
field_list_tail: FieldListTailNode
next_tail: TypeTailNode
type_item: None
next_tail: None
next_section: TypeSectionNode
type_declaration: None
next_section: None
var_section: VarSectionNode
var_declaration: VarDeclNode
var_item: VarItemNode
identifier_list: IdentifierListNode
identifier: 'i'
tail: IdentifierListTailNode
identifier: 'j'
next_tail: IdentifierListTailNode
identifier: None
next_tail: None
type_node: TypeNode
name: 'integer'
item_tail: VarTailNode
var_item: VarItemNode

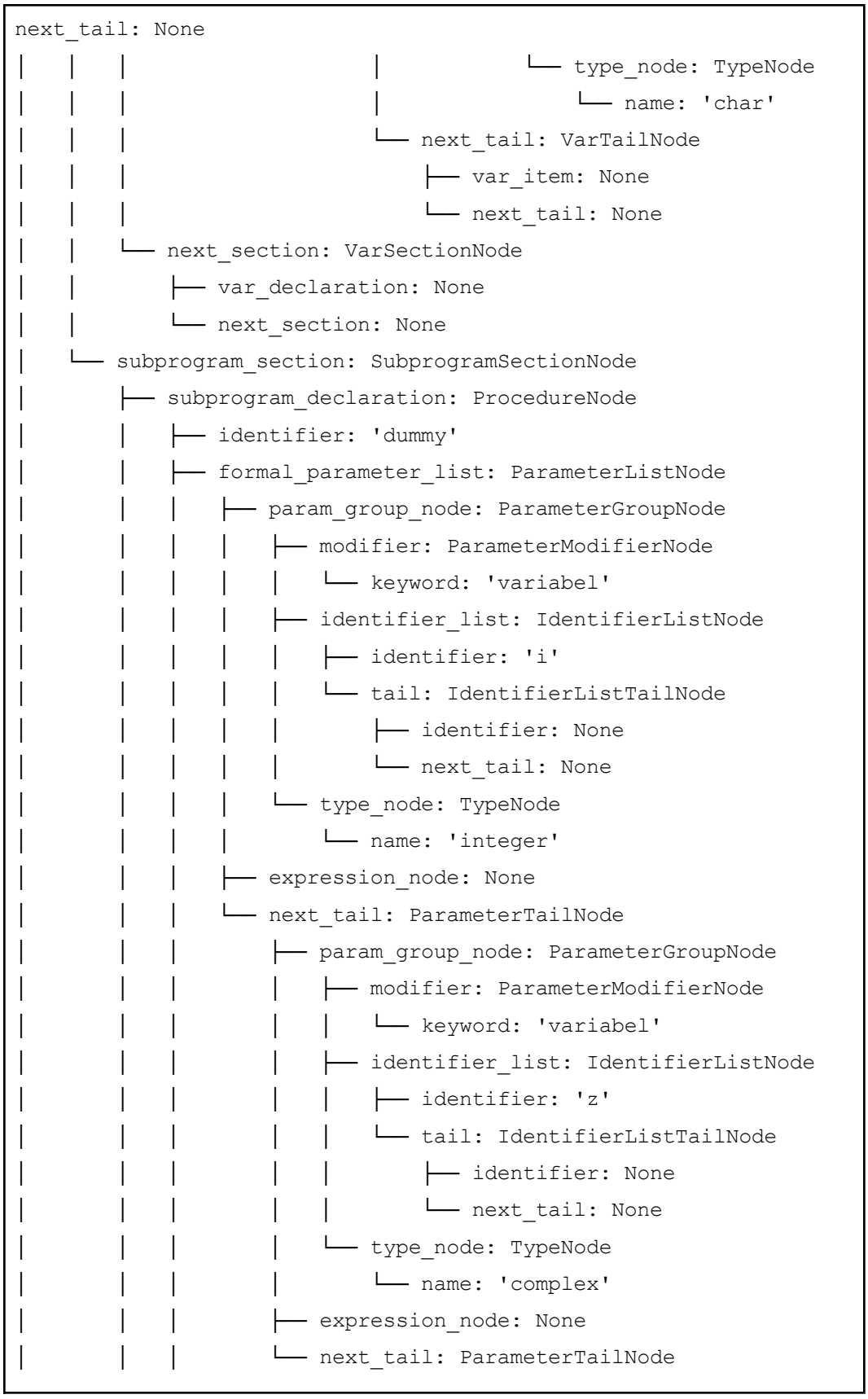

identifier_list: IdentifierListNode
identifier: 'p'
tail: IdentifierListTailNode
identifier: None
next_tail: None
type_node: TypeNode
name: 'boolean'
next_tail: VarTailNode
var_item: VarItemNode
identifier_list: IdentifierListNode
identifier: 'z'
tail: IdentifierListTailNode
identifier: None
next_tail: None
type_node: TypeNode
name: 'complex'
next_tail: VarTailNode
var_item: VarItemNode
identifier_list: IdentifierListNode
identifier: 'matrix'
tail: IdentifierListTailNode
identifier: None
next_tail: None
type_node: ArrayTypeNode
range_node: RangeNode
expression_1: UnaryOpNode
operator: OperatorNode
lexeme: '-'
term_node: TermNode
factor: NumberNode
value: '3'
tail: TermTailNode

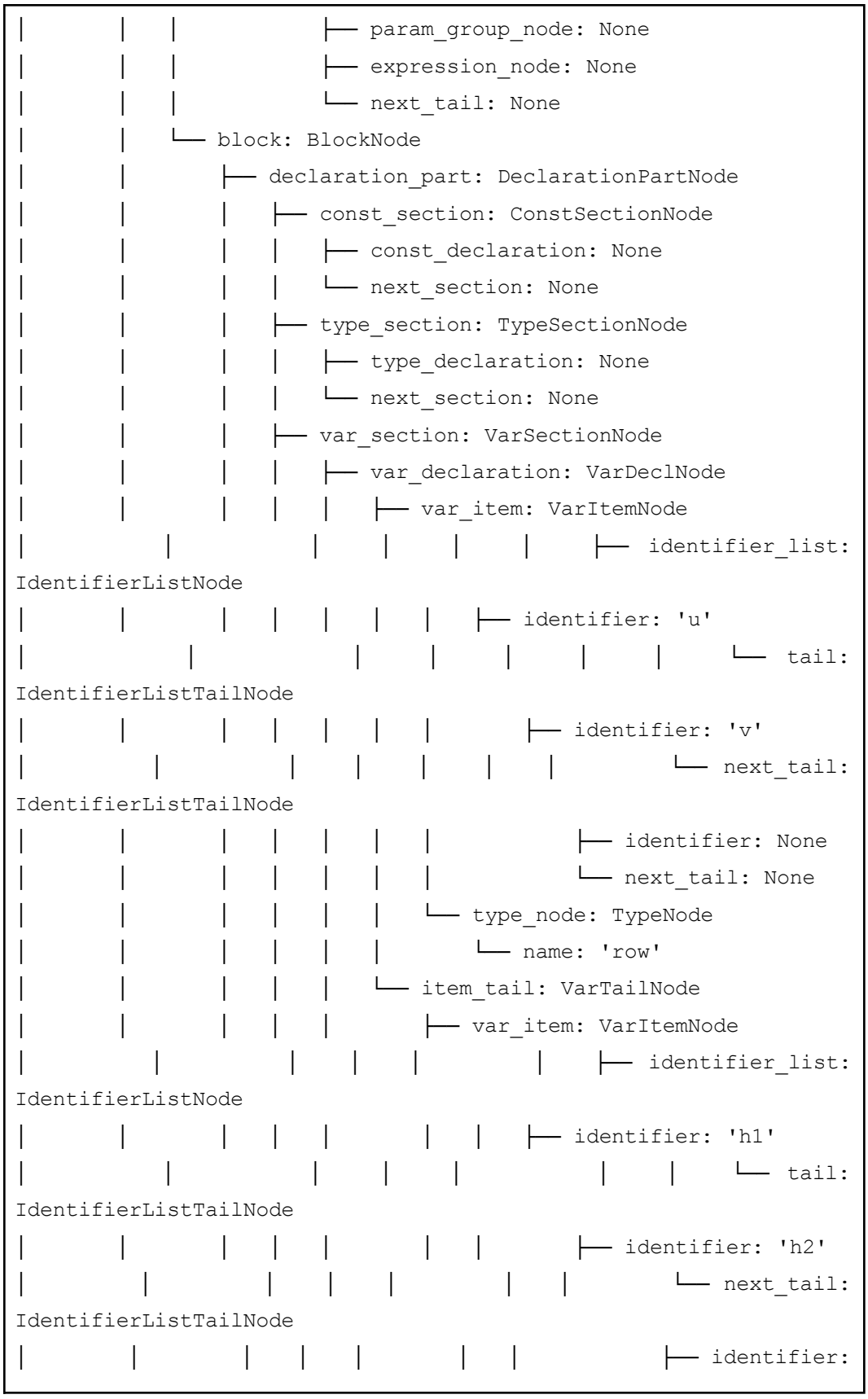

multiplicative_operator: None
| | | | | | | ├── factor: None
| | | | | | | └── next_tail:
None
| | | | | | ├── factor_node: None
| | | | | | | └── tail:
SimpleExprTailNode
| | | | | | | ├──
additive_operator: None
| | | | | | ├── term: None
| | | | | | └── next_tail: None
| | | | | | ├── range_operator:
OperatorNode
| | | | | | └── lexeme: '..'
| | | | | └── expression_2: UnaryOpNode
| | | | | | ├── operator:
OperatorNode
| | | | | | └── lexeme: '+'
| | | | | ├── term_node: TermNode
| | | | | | ├── factor:
NumberNode
| | | | | | | └── value: '3'
| | | | | | └── tail:
TermTailNode
| | | | | | | ├──
multiplicative_operator: None
| | | | | ├── factor: None
| | | | | └── next_tail:
None
| | | | | ├── factor_node: None
| | | | | | └── tail:
SimpleExprTailNode
| | | | | | ├──
additive_operator: None
| | | | | ├── term: None
| | | | | └── next_tail: None
| | | | └── type_node: TypeNode

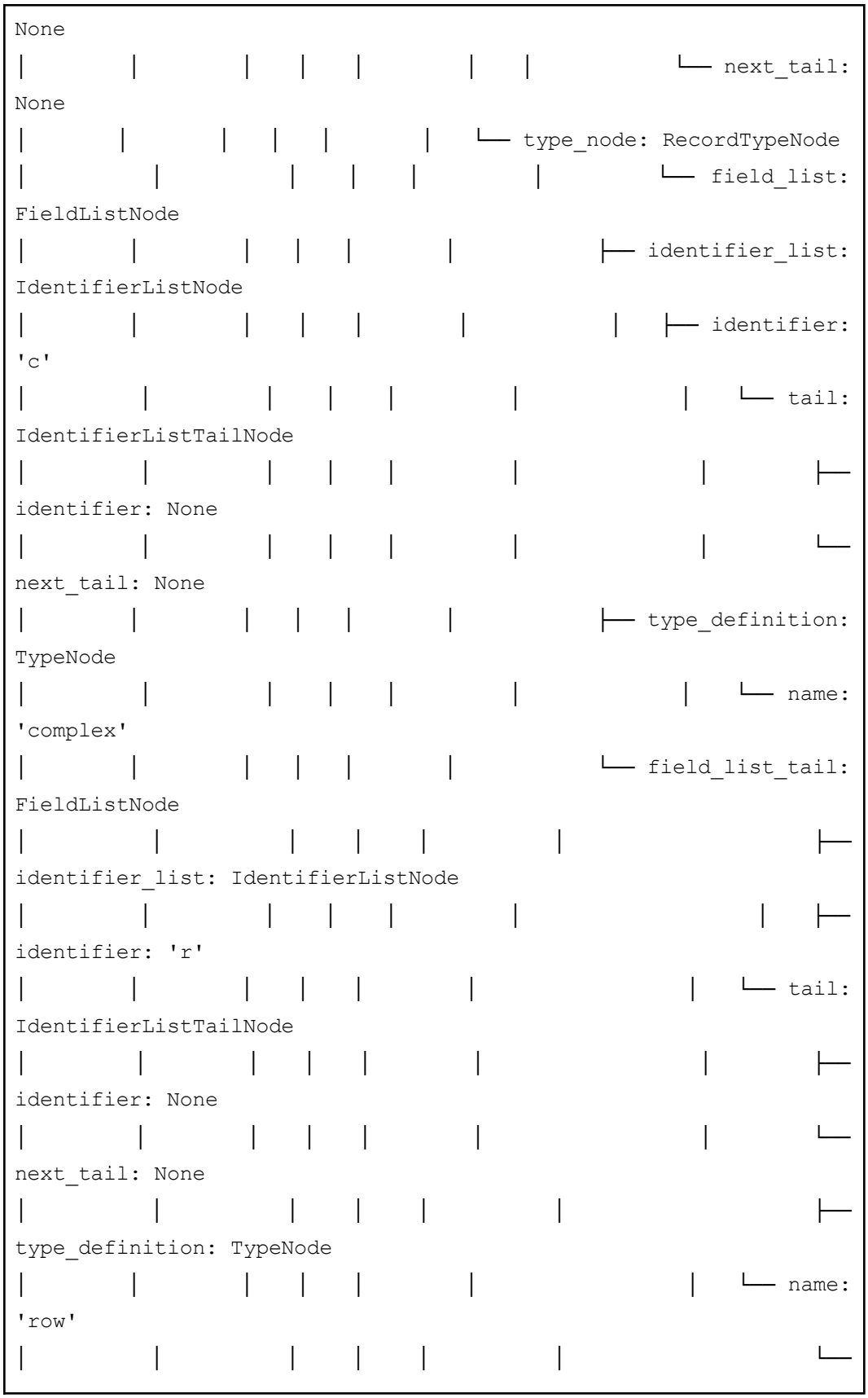

| | | | | └── name: 'row'
| | | | └── next_tail: VarTailNode
| | | | ├── var_item: VarItemNode
| | | | | ├── identifier_list: IdentifierListNode
| | | | | | ├── identifier: 'pattern'
| | | | | | └── tail: IdentifierListTailNode
| | | | | | ├── identifier: None
| | | | | | └── next_tail: None
| | | | | └── type_node: ArrayTypeNode
| | | | | ├── range_node: RangeNode
| | | | | | ├── expression_1: SimpleExprNode
| | | | | | | ├── term: TermNode
| | | | | | | | ├── factor: NumberNode
| | | | | | | | | └── value: '1'
| | | | | | | | └── tail: TermTailNode
| | | | | | | | ├── multiplicative_operator: None
| | | | | | | | ├── factor: None
| | | | | | | | └── next_tail: None
| | | | | | | └── tail: SimpleExprTailNode
| | | | | | | ├── additive_operator: None
| | | | | | | ├── term: None
| | | | | | | └── next_tail: None
| | | | | | ├── range_operator: OperatorNode
| | | | | | | └── lexeme: '..'

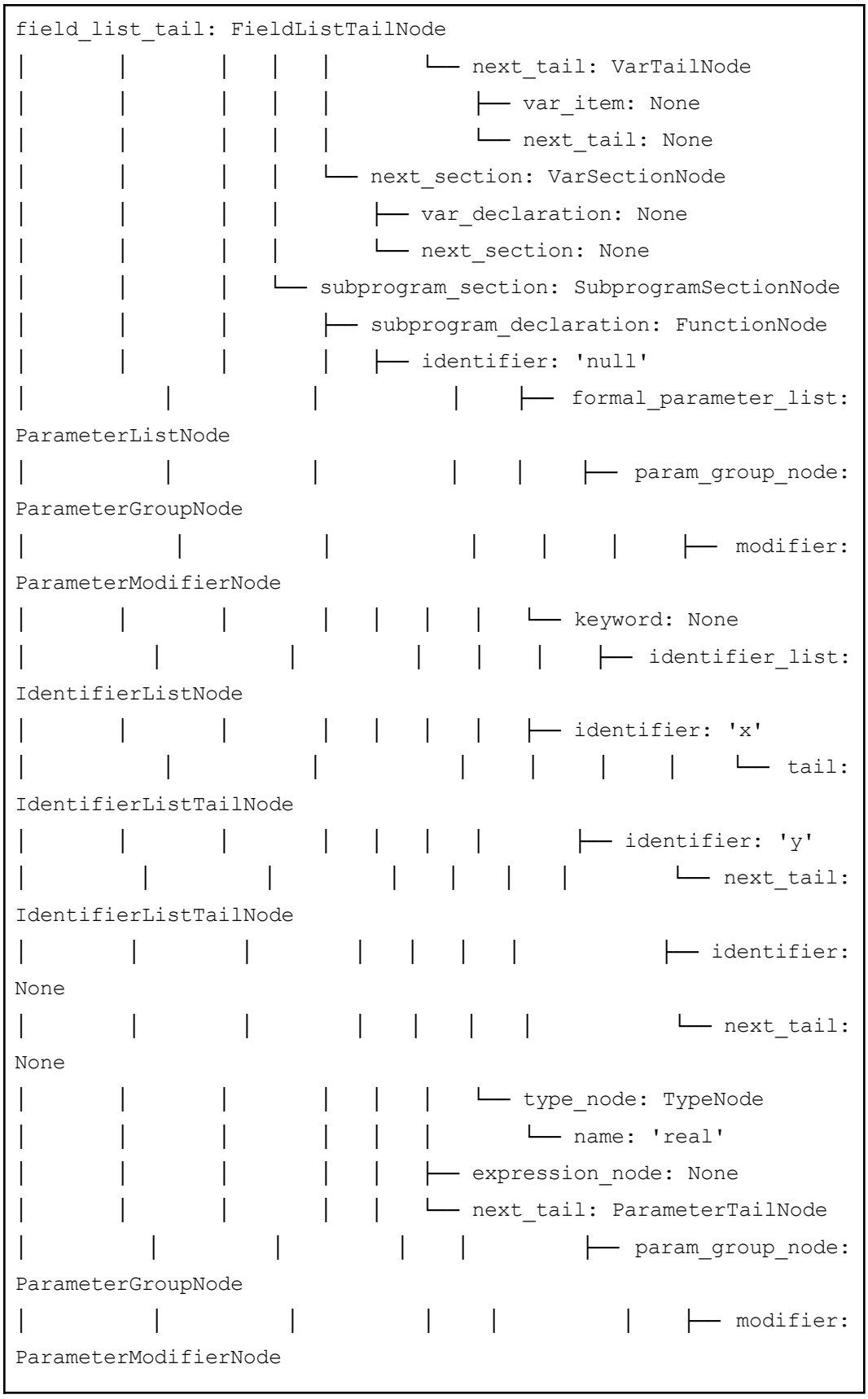
```
|   |   |                       |       |   └── expression_2:
SimpleExprNode
|   |   |                   |       |       ├── term: TermNode
|   |   |                     |       |       |   ├── factor:
NumberNode
|   |   |                     |       |       |   |   └── value:
'5'
|   |   |                     |       |       |   └── tail:
TermTailNode
|   |   |                     |       |       |       ├──
multiplicative_operator: None
|   |   |                     |       |       |       ├── factor:
None
|   |   |                     |       |       |       └──
next_tail: None
|   |   |                       |       |       └── tail:
SimpleExprTailNode
|   |   |                       |       |           ├──
additive_operator: None
|   |   |                   |       |           ├── term: None
|   |   |                     |       |           └── next_tail:
None
|   |   |                   |       └── type_node: ArrayTypeNode
|   |   |                   |           ├── range_node: RangeNode
|   |   |                     |           |   ├── expression_1:
SimpleExprNode
|   |   |                     |           |   |   ├── term:
TermNode
|   |   |                   |           |   |   ├── factor:
NumberNode
|   |   |                     |           |   |   |   └──
value: '2'
|   |   |                     |           |   |   └── tail:
TermTailNode
|   |   |                     |           |   |       ├──
multiplicative_operator: None
|   |   |                     |           |   |       ├──
```

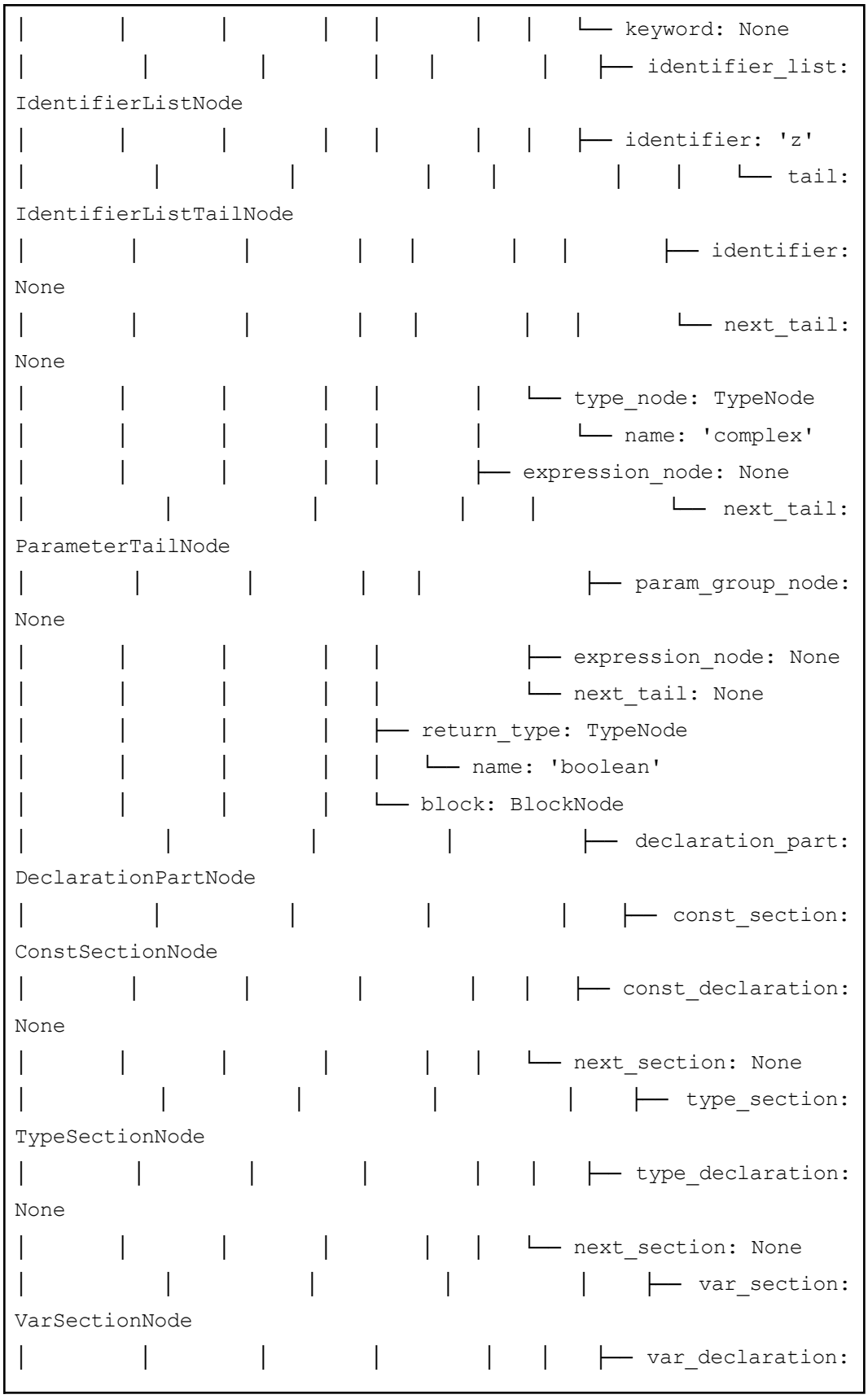

factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
range_operator: OperatorNode
lexeme: '..'
expression_2: SimpleExprNode
term: TermNode
factor: NumberNode
value: '5'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None

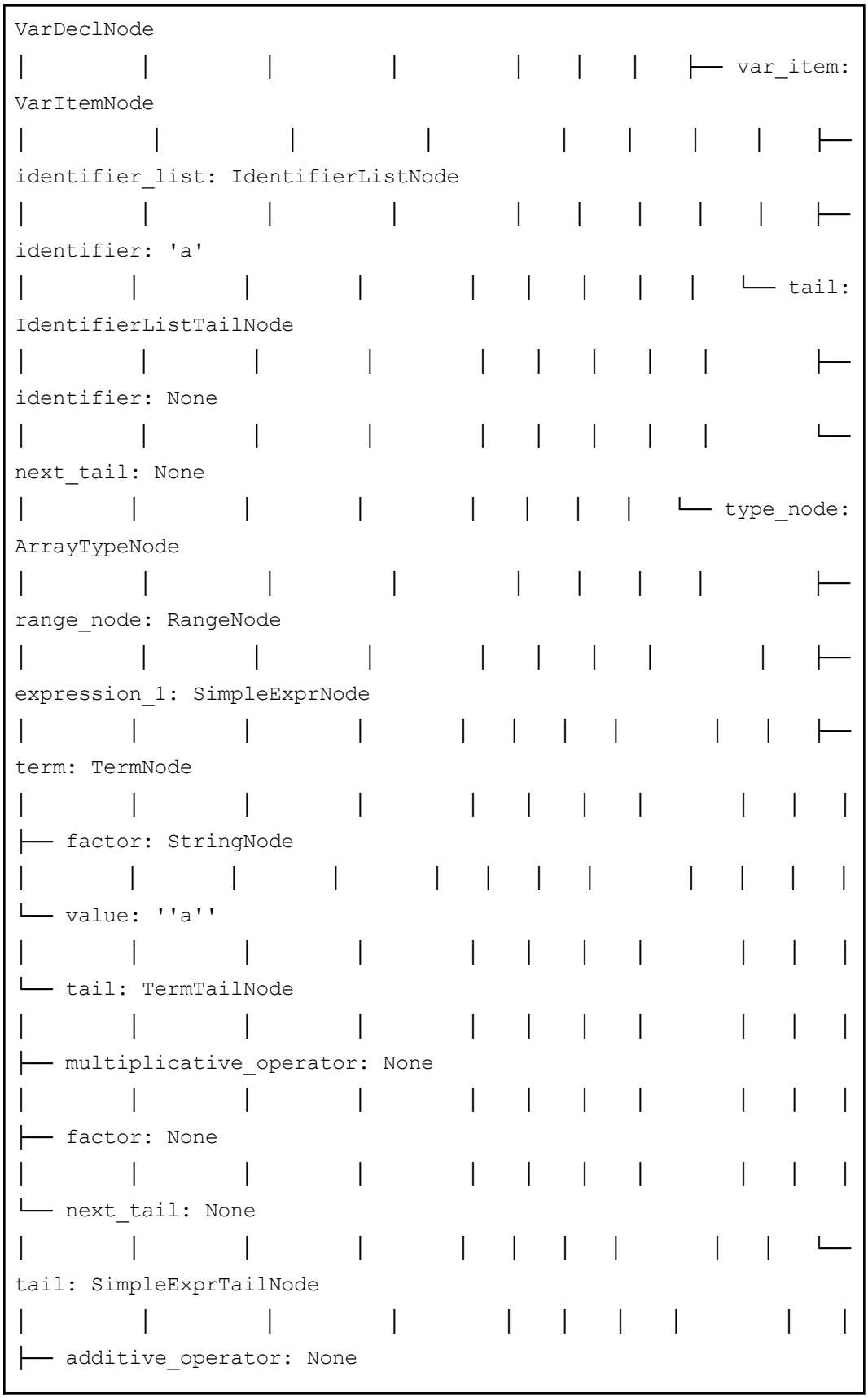

next_tail: None
type_node: TypeNode
name: 'char'
next_tail: VarTailNode
var_item: None
next_tail: None
next_section: VarSectionNode
var_declaration: None
next_section: None
subprogram_section: SubprogramSectionNode
subprogram_declaration: ProcedureNode
identifier: 'dummy'
formal_parameter_list: ParameterListNode
param_group_node: ParameterGroupNode
modifier: ParameterModifierNode
keyword: 'variabel'
identifier_list: IdentifierListNode
identifier: 'i'
tail: IdentifierListTailNode
identifier: None
next_tail: None
type_node: TypeNode
name: 'integer'
expression_node: None
next_tail: ParameterTailNode
param_group_node: ParameterGroupNode
modifier: ParameterModifierNode
keyword: 'variabel'
identifier_list: IdentifierListNode
identifier: 'z'
tail: IdentifierListTailNode
identifier: None
next_tail: None
type_node: TypeNode
name: 'complex'
expression_node: None
next_tail: ParameterTailNode

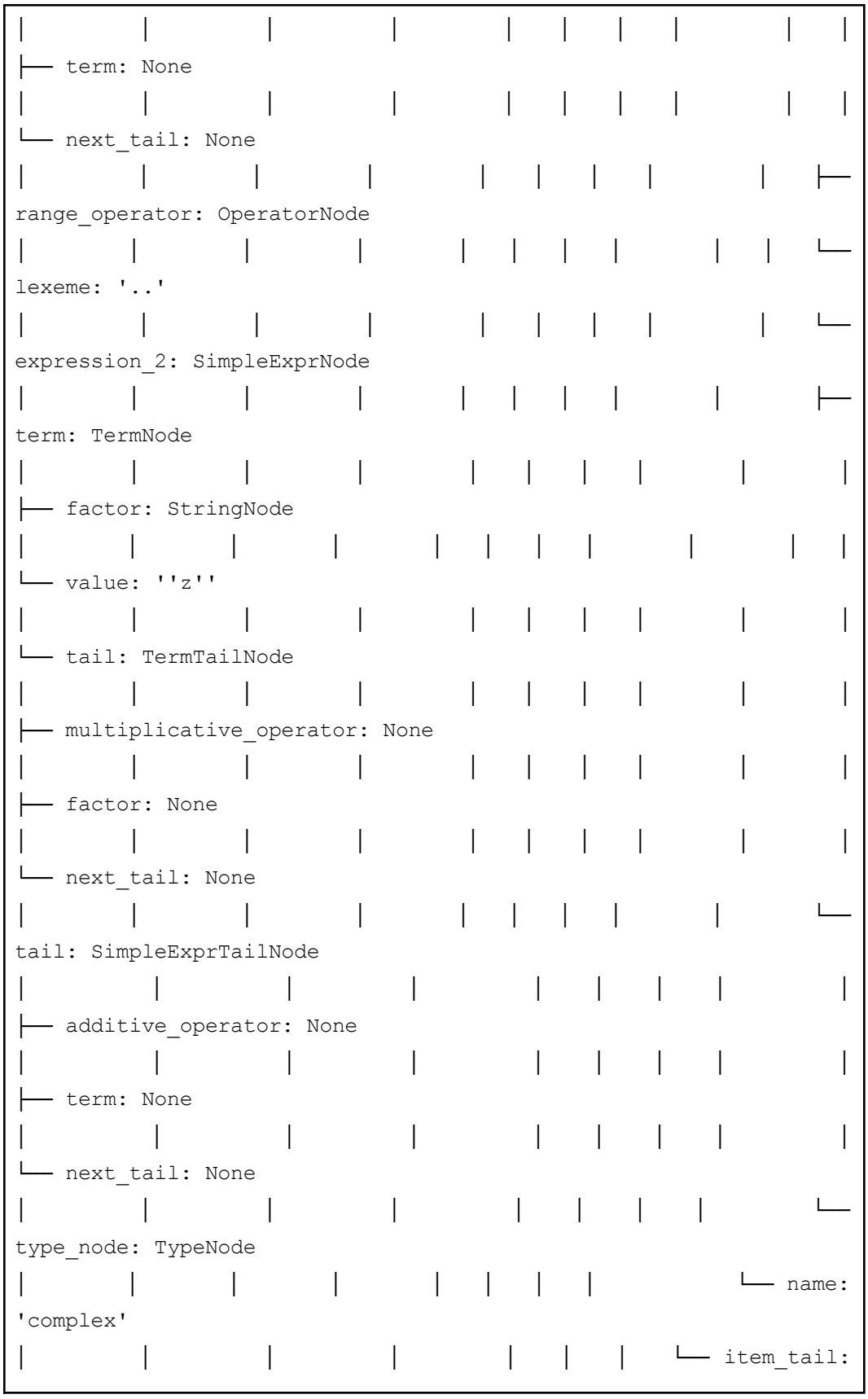

├── param_group_node: None
├── expression_node: None
└── next_tail: None
└── block: BlockNode
├── declaration_part: DeclarationPartNode
├── const_section: ConstSectionNode
├── const_declaration: None
└── next_section: None
├── type_section: TypeSectionNode
├── type_declaration: None
└── next_section: None
├── var_section: VarSectionNode
├── var_declaration: VarDeclNode
├── var_item: VarItemNode
├── identifier_list: IdentifierListNode
├── identifier: 'u'
└── tail: IdentifierListTailNode
├── identifier: 'v'
└── next_tail: IdentifierListTailNode
├── identifier: None
└── next_tail: None
└── type_node: TypeNode
└── name: 'row'
└── item_tail: VarTailNode
├── var_item: VarItemNode
├── identifier_list: IdentifierListNode
├── identifier: 'h1'
└── tail: IdentifierListTailNode
├── identifier: 'h2'
└── next_tail: IdentifierListTailNode
├── identifier:

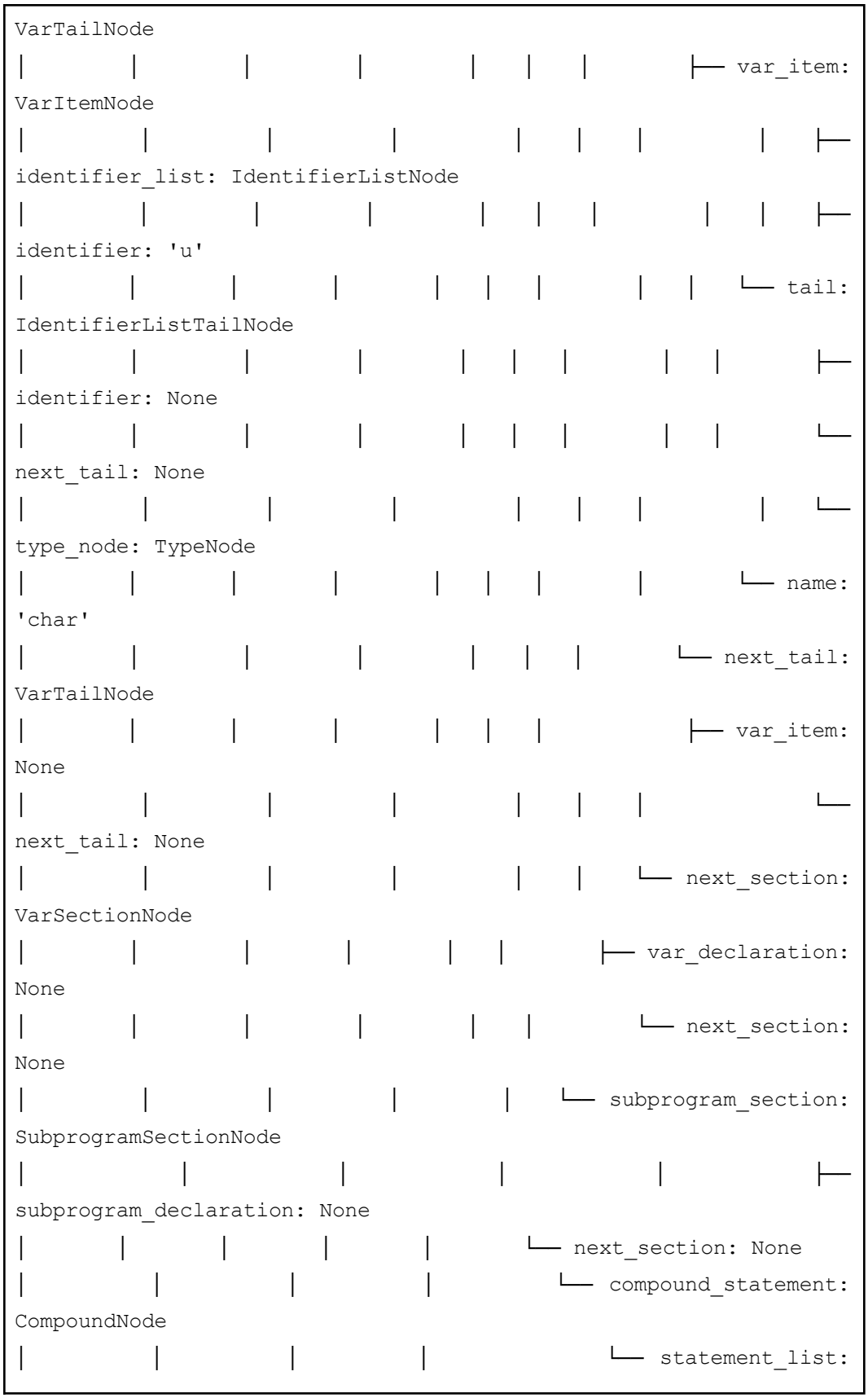

None
| | | | | | | └── next_tail: None
| | | | | | └── type_node: RecordTypeNode
| | | | | | └── field_list: FieldListNode
| | | | | | ├── identifier_list: IdentifierListNode
| | | | | | | ├── identifier: 'c'
| | | | | | | └── tail: IdentifierListTailNode
| | | | | | | ├── identifier: None
| | | | | | | └── next_tail: None
| | | | | | ├── type_definition: TypeNode
| | | | | | | └── name: 'complex'
| | | | | | └── field_list_tail: FieldListNode
| | | | | | ├── identifier_list: IdentifierListNode
| | | | | | | ├── identifier: 'r'
| | | | | | | └── tail: IdentifierListTailNode
| | | | | | | ├── identifier: None
| | | | | | | └── next_tail: None
| | | | | | ├── type_definition: TypeNode
| | | | | | | └── name: 'row'
| | | | | | └──

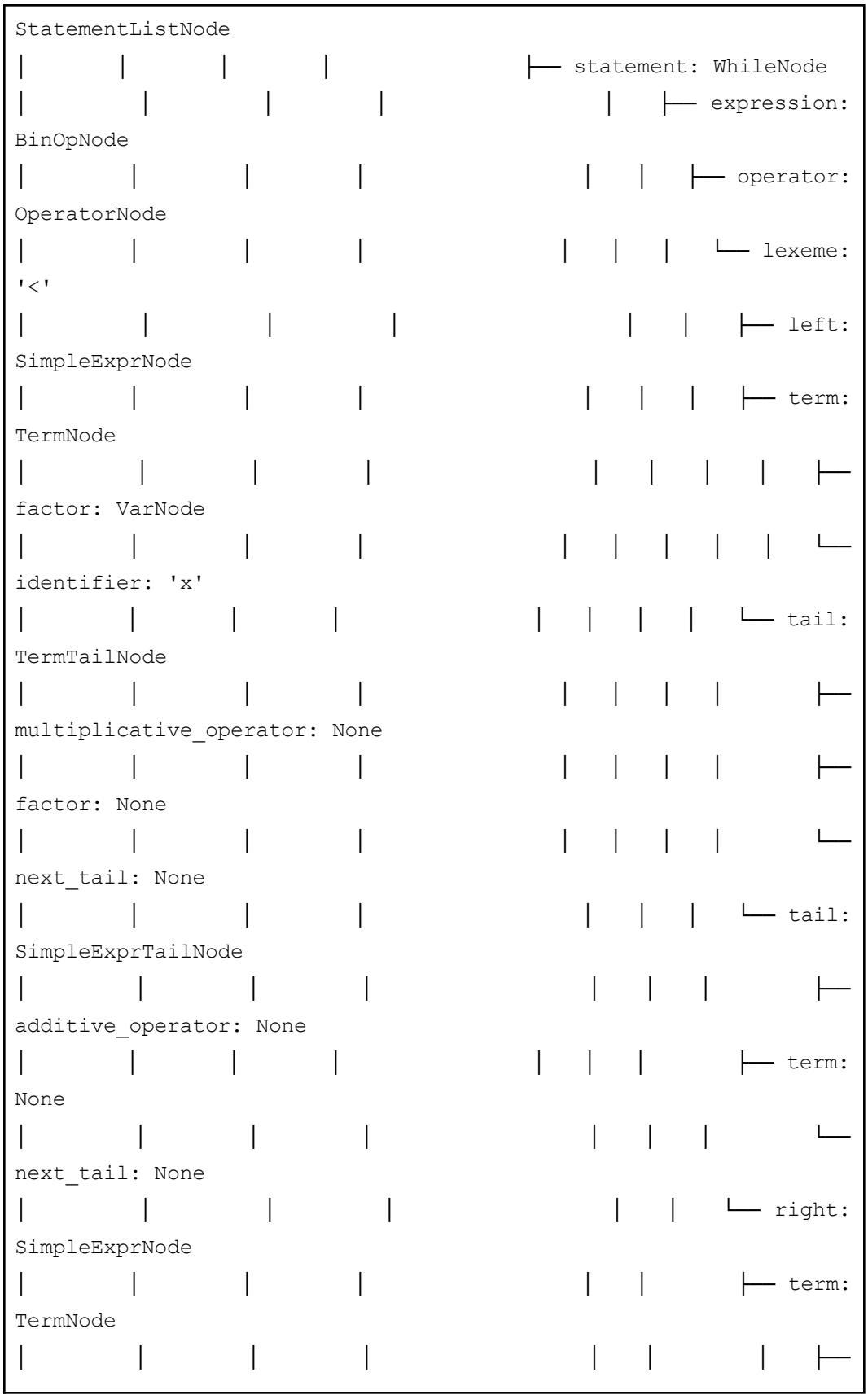

field_list_tail: FieldListTailNode
└── next_tail: VarTailNode
├── var_item: None
└── next_tail: None
└── next_section: VarSectionNode
├── var_declaration: None
└── next_section: None
└── subprogram_section: SubprogramSectionNode
├── subprogram_declaration: FunctionNode
├── identifier: 'null'
├── formal_parameter_list: ParameterListNode
├── param_group_node: ParameterGroupNode
├── modifier: ParameterModifierNode
└── keyword: None
├── identifier_list: IdentifierListNode
├── identifier: 'x'
└── tail: IdentifierListTailNode
├── identifier: 'y'
└── next_tail: IdentifierListTailNode
├── identifier: None
└── next_tail: None
└── type_node: TypeNode
└── name: 'real'
├── expression_node: None
└── next_tail: ParameterTailNode
├── param_group_node: ParameterGroupNode
├── modifier: ParameterModifierNode

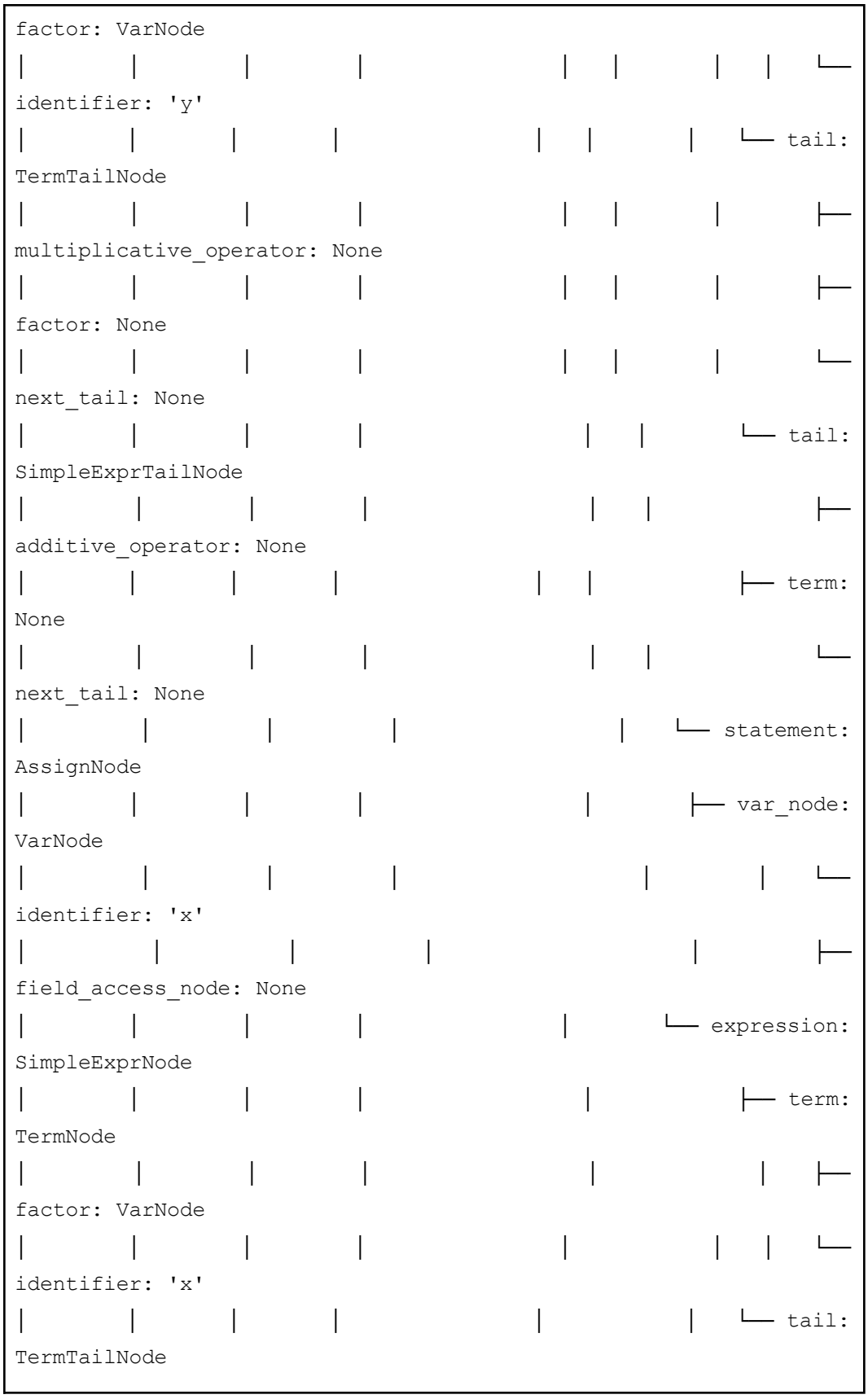

| | | | | | | └── keyword: None
| | | | | | ├── identifier_list:
IdentifierListNode
| | | | | | | ├── identifier: 'z'
| | | | | | | └── tail:
IdentifierListTailNode
| | | | | | | ├── identifier:
None
| | | | | | | └── next_tail:
None
| | | | | | └── type_node: TypeNode
| | | | | | └── name: 'complex'
| | | | | ├── expression_node: None
| | | | | └── next_tail:
ParameterTailNode
| | | | | ├── param_group_node:
None
| | | | | ├── expression_node: None
| | | | | └── next_tail: None
| | | | ├── return_type: TypeNode
| | | | | └── name: 'boolean'
| | | | └── block: BlockNode
| | | | ├── declaration_part:
DeclarationPartNode
| | | | | ├── const_section:
ConstSectionNode
| | | | | ├── const_declaration:
None
| | | | | └── next_section: None
| | | | | ├── type_section:
TypeSectionNode
| | | | | ├── type_declaration:
None
| | | | | └── next_section: None
| | | | | ├── var_section:
VarSectionNode
| | | | | ├── var_declaration:

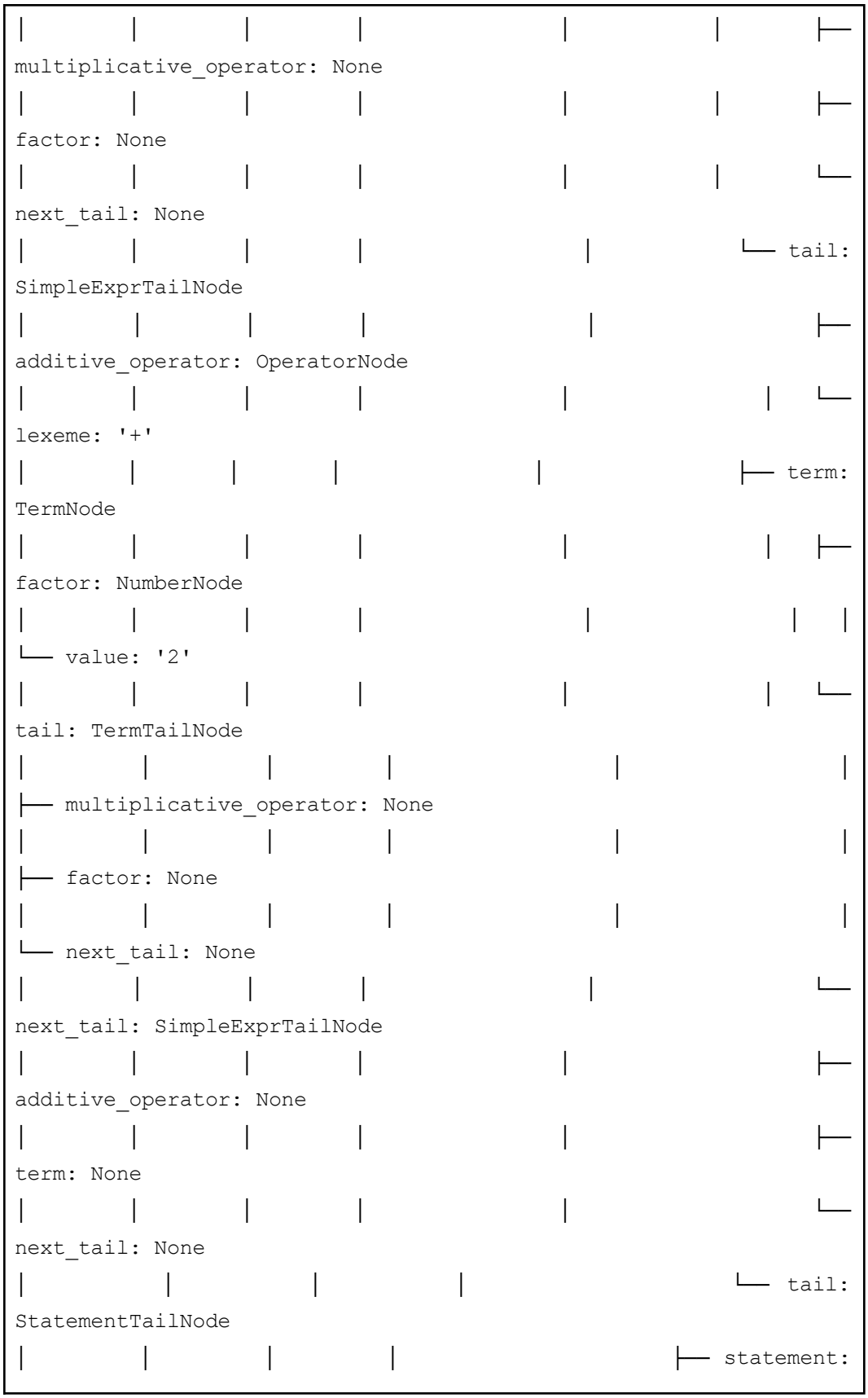

VarDeclNode
├── var_item: VarItemNode
│   ├── identifier_list: IdentifierListNode
│   │   ├── identifier: 'a'
│   │   └── tail: IdentifierListTailNode
│   │   ├── identifier: None
│   │   └── next_tail: None
│   └── type_node: ArrayTypeNode
│   ├── range_node: RangeNode
│   │   ├── expression_1: SimpleExprNode
│   │   │   ├── term: TermNode
│   │   │   │   ├── factor: StringNode
│   │   │   │   │   └── value: ''a''
│   │   │   │   └── tail: TermTailNode
│   │   │   │   ├── multiplicative_operator: None
│   │   │   │   ├── factor: None
│   │   │   │   └── next_tail: None
│   │   │   └── tail: SimpleExprTailNode
│   │   │   ├── additive_operator: None

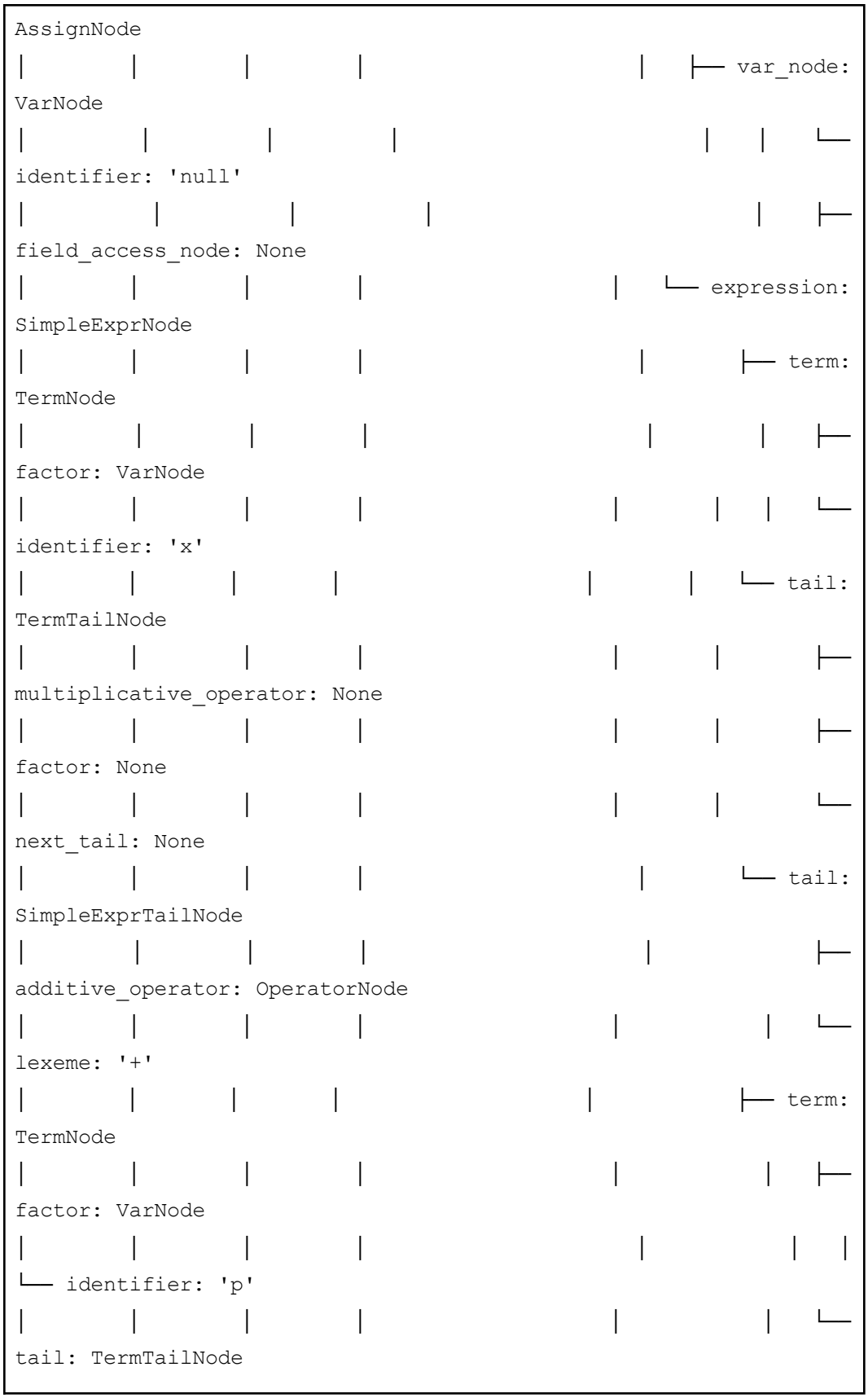

term: None
next_tail: None
range_operator: OperatorNode
lexeme: '..'
expression_2: SimpleExprNode
term: TermNode
factor: StringNode
value: ''z''
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
type_node: TypeNode
name: 'complex'
item_tail:

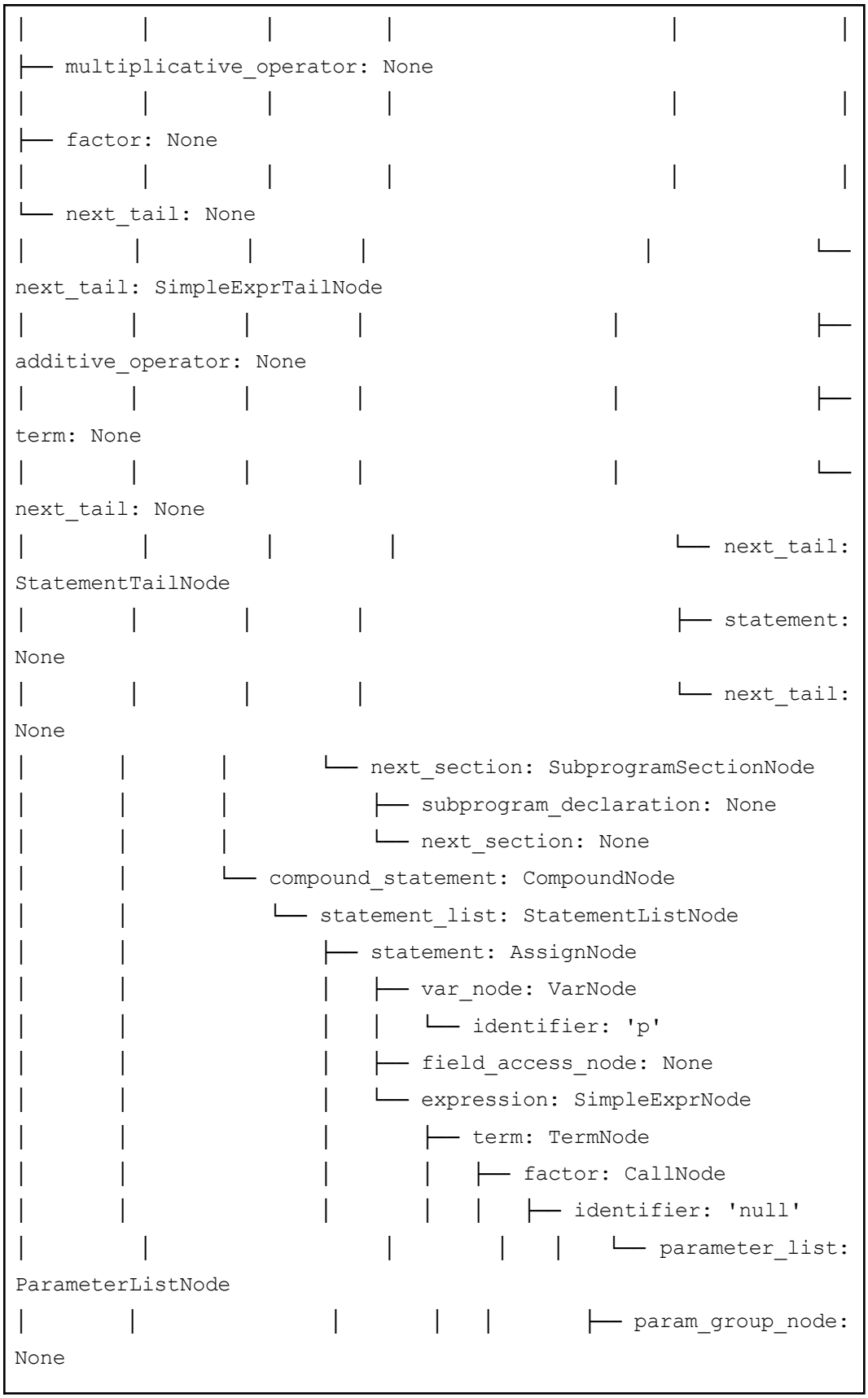

VarTailNode
var_item:
VarItemNode
identifier_list: IdentifierListNode
identifier: 'u'
tail:
IdentifierListTailNode
identifier: None
next_tail: None
type_node: TypeNode
name:
'char'
next_tail:
VarTailNode
var_item:
None
next_tail: None
next_section:
VarSectionNode
var_declaration:
None
next_section:
None
subprogram_section:
SubprogramSectionNode
subprogram_declaration: None
next_section: None
compound_statement:
CompoundNode
statement_list:

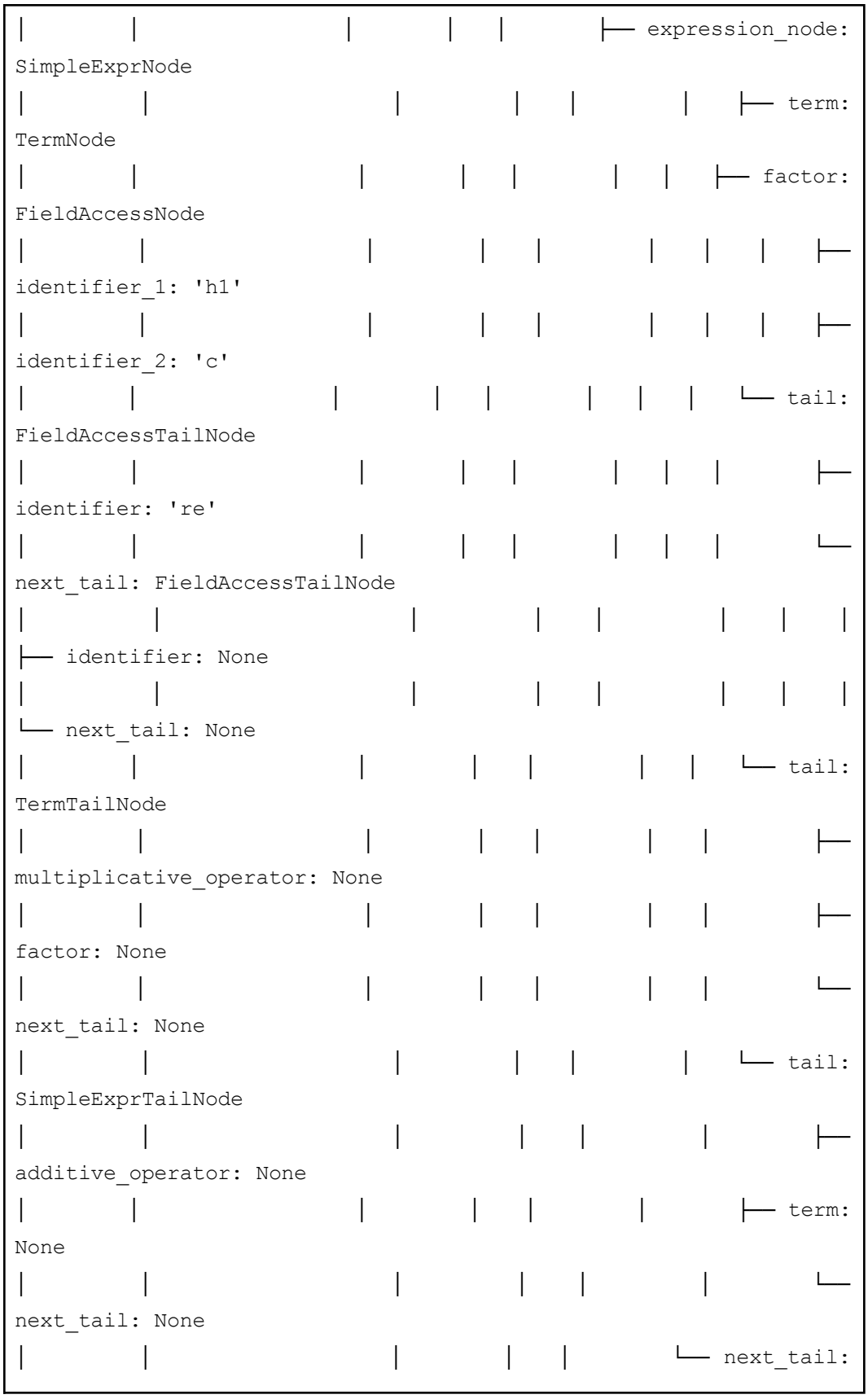

StatementListNode
├── statement: WhileNode
│ ├── expression: BinOpNode
│ │ ├── operator: OperatorNode
│ │ │ └── lexeme: '<'
│ │ ├── left: SimpleExprNode
│ │ │ ├── term: TermNode
│ │ │ │ ├── factor: VarNode
│ │ │ │ │ └── identifier: 'x'
│ │ │ │ └── tail: TermTailNode
│ │ │ │ ├── multiplicative_operator: None
│ │ │ │ ├── factor: None
│ │ │ │ └── next_tail: None
│ │ │ └── tail: SimpleExprTailNode
│ │ │ ├── additive_operator: None
│ │ │ ├── term: None
│ │ │ └── next_tail: None
│ │ └── right: SimpleExprNode
│ │ ├── term: TermNode
│ │ │ ├──

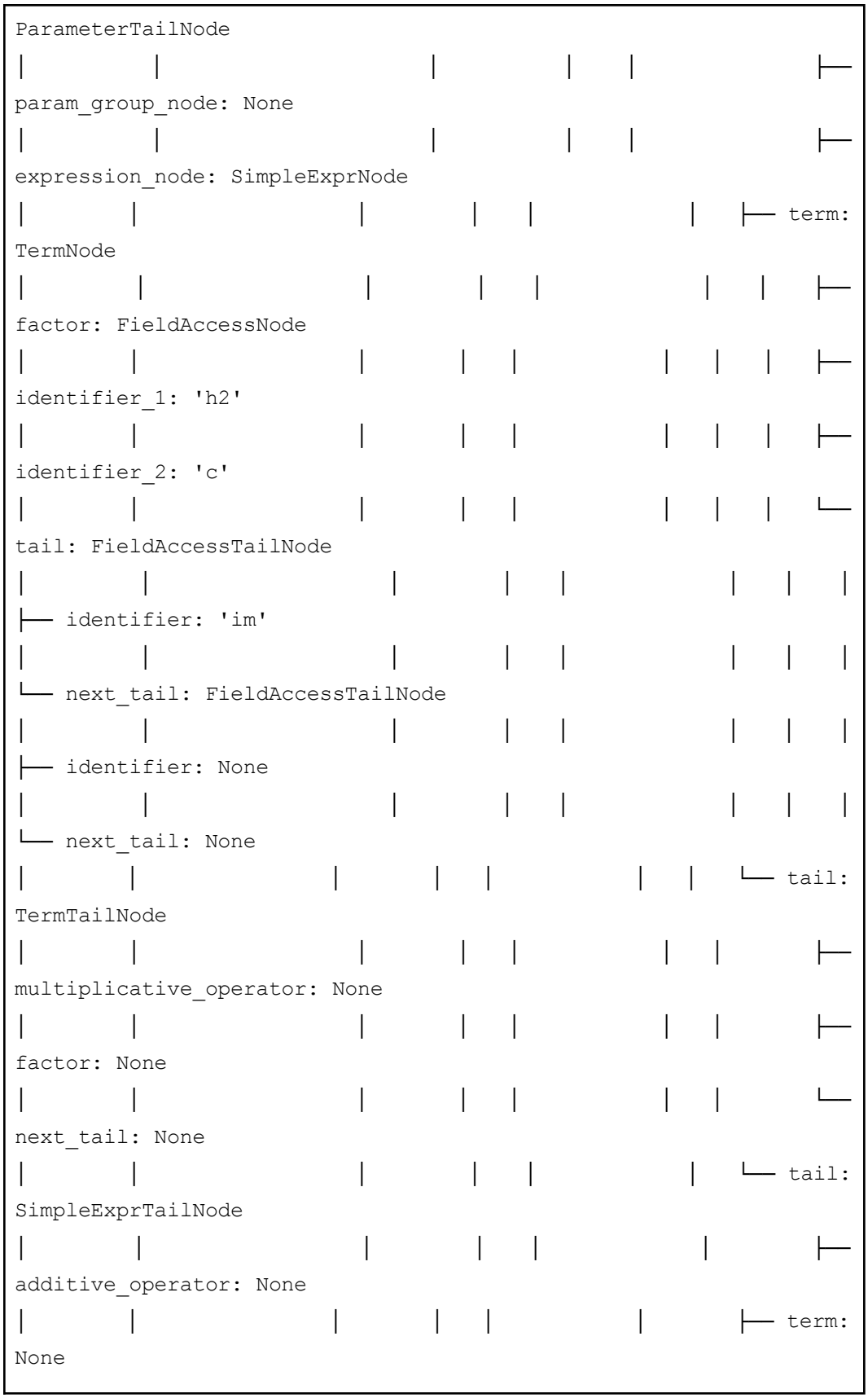

factor: VarNode
identifier: 'y'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
statement: AssignNode
var_node: VarNode
identifier: 'x'
field_access_node: None
expression: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'x'
tail: TermTailNode

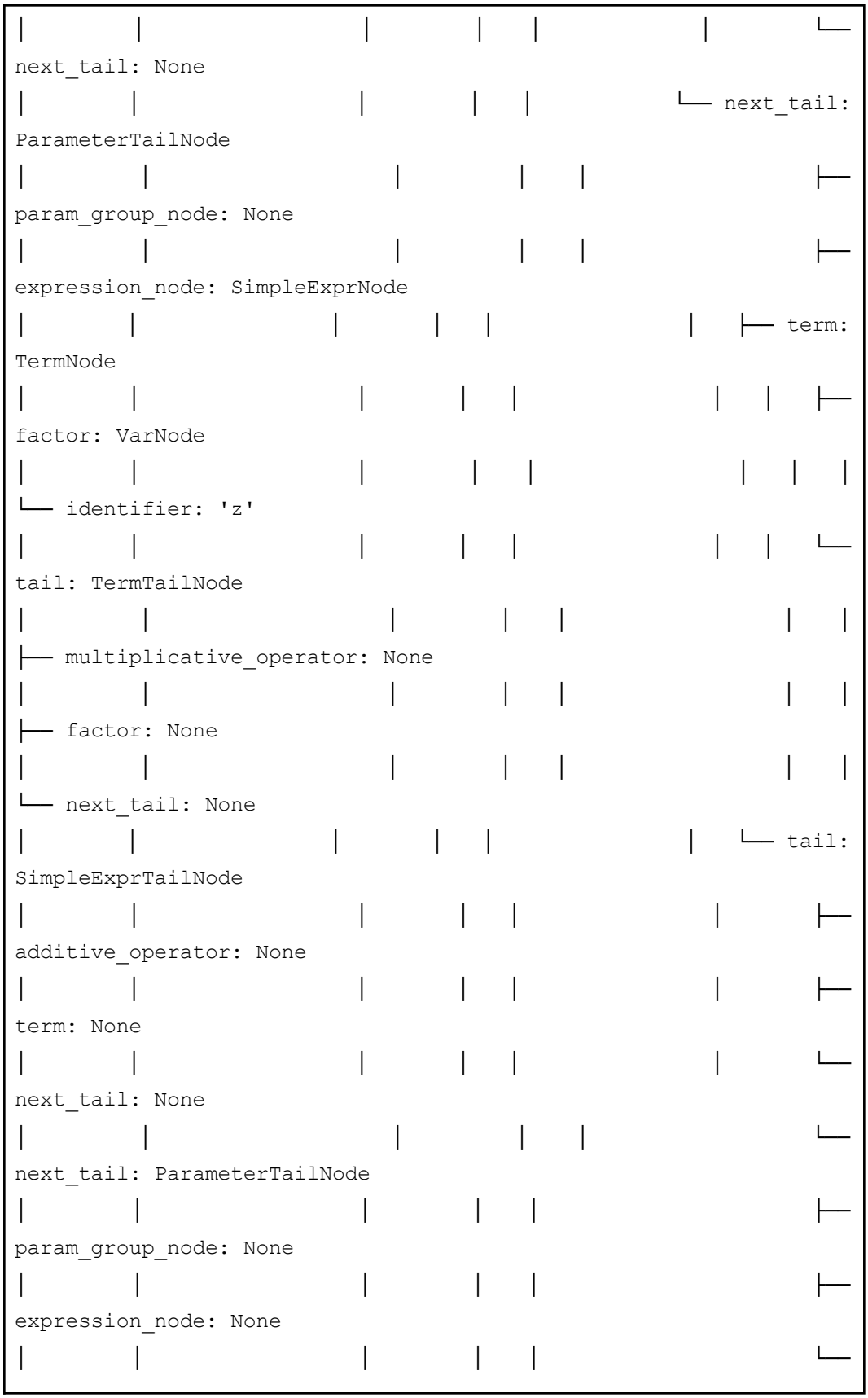

multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: OperatorNode
lexeme: '+'
term: TermNode
factor: NumberNode
value: '2'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
next_tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
tail: StatementTailNode
statement:


AssignNode
├── var_node: VarNode
│   └── identifier: 'null'
├── field_access_node: None
└── expression: SimpleExprNode
    ├── term: TermNode
    │   ├── factor: VarNode
    │   │   └── identifier: 'x'
    │   └── tail: TermTailNode
    │   ├── multiplicative_operator: None
    │   ├── factor: None
    │   └── next_tail: None
    └── tail: SimpleExprTailNode
        ├── additive_operator: OperatorNode
        │   └── lexeme: '+'
        └── term: TermNode
            ├── factor: VarNode
            │   └── identifier: 'p'
            └── tail: TermTailNode

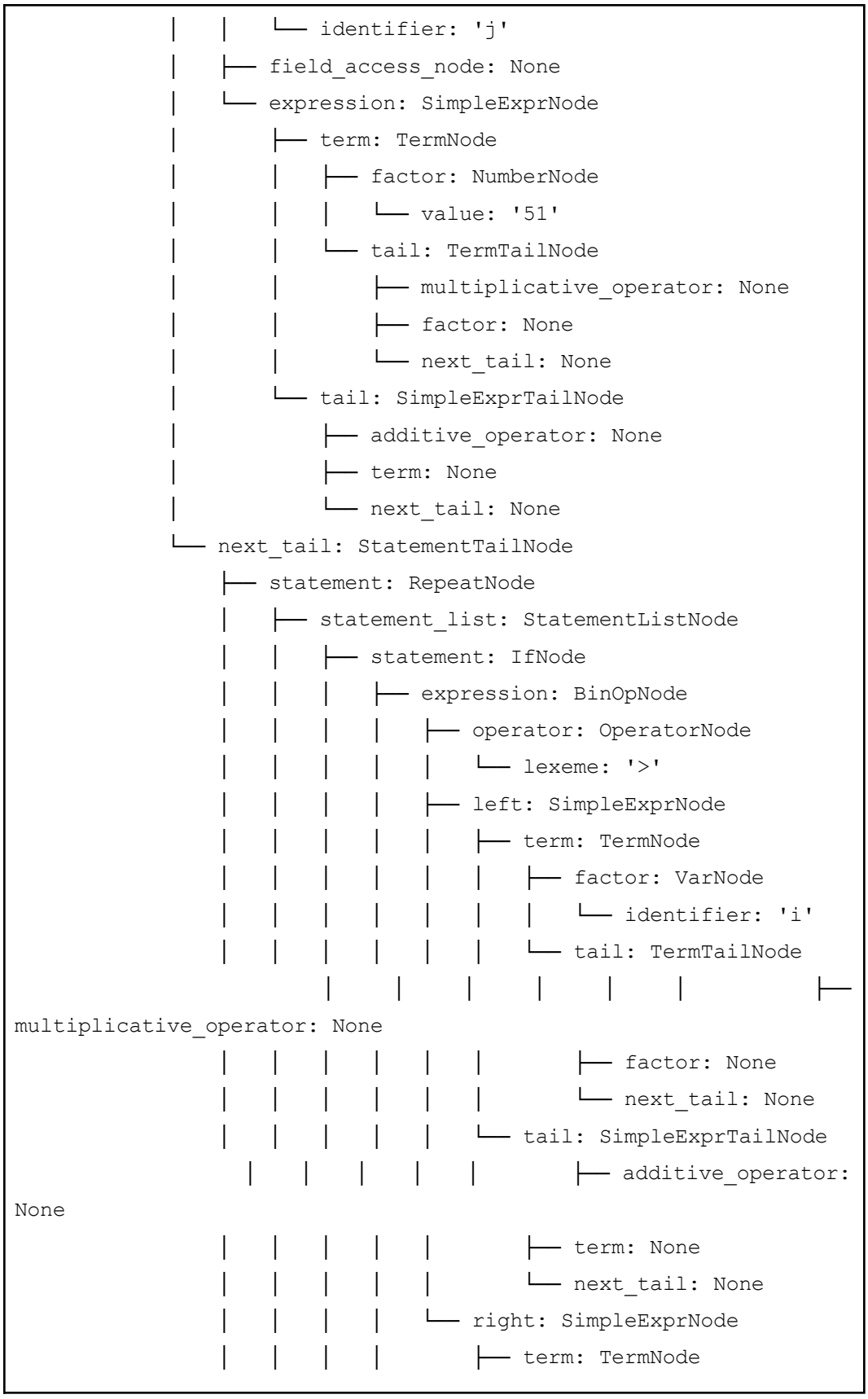

multiplicative_operator: None
factor: None
next_tail: None
next_tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
next_tail: StatementTailNode
statement: None
next_tail: None
next_section: SubprogramSectionNode
subprogram_declaration: None
next_section: None
compound_statement: CompoundNode
statement_list: StatementListNode
statement: AssignNode
var_node: VarNode
identifier: 'p'
field_access_node: None
expression: SimpleExprNode
term: TermNode
factor: CallNode
identifier: 'null'
parameter_list: ParameterListNode
param_group_node: None

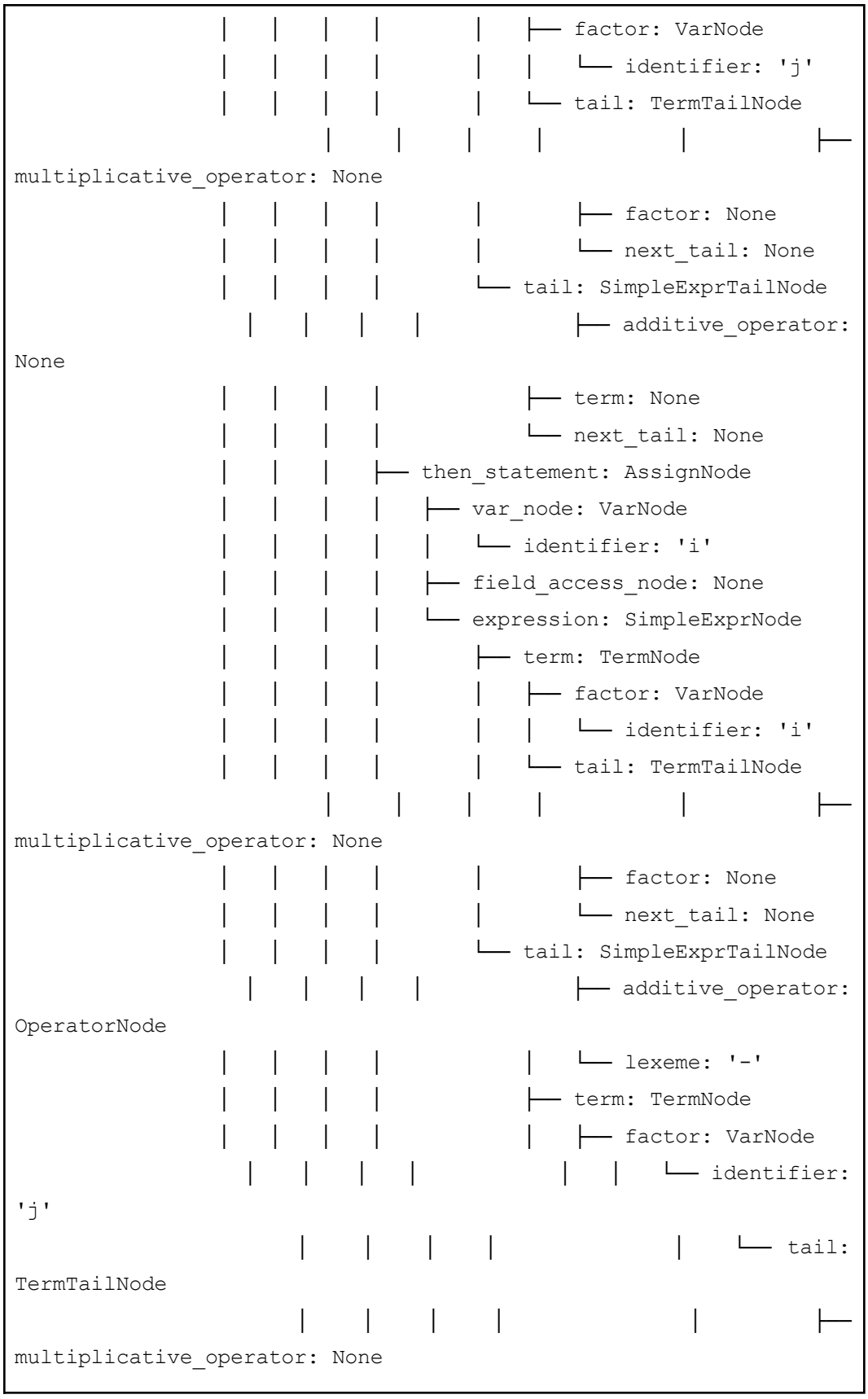

| | | | | ├── expression_node:
SimpleExprNode
| | | | | | ├── term:
TermNode
| | | | | | | ├── factor:
FieldAccessNode
| | | | | | | | ├──
identifier_1: 'h1'
| | | | | | | | ├──
identifier_2: 'c'
| | | | | | | | └── tail:
FieldAccessTailNode
| | | | | | | | ├──
identifier: 're'
| | | | | | | | └──
next_tail: FieldAccessTailNode
| | | | | | | ├──
identifier: None
| | | | | | | └──
next_tail: None
| | | | | | └── tail:
TermTailNode
| | | | | | ├──
multiplicative_operator: None
| | | | | | ├──
factor: None
| | | | | | └──
next_tail: None
| | | | | └── tail:
SimpleExprTailNode
| | | | | ├──
additive_operator: None
| | | | | ├── term:
None
| | | | | └──
next_tail: None
| | | | └── next_tail:

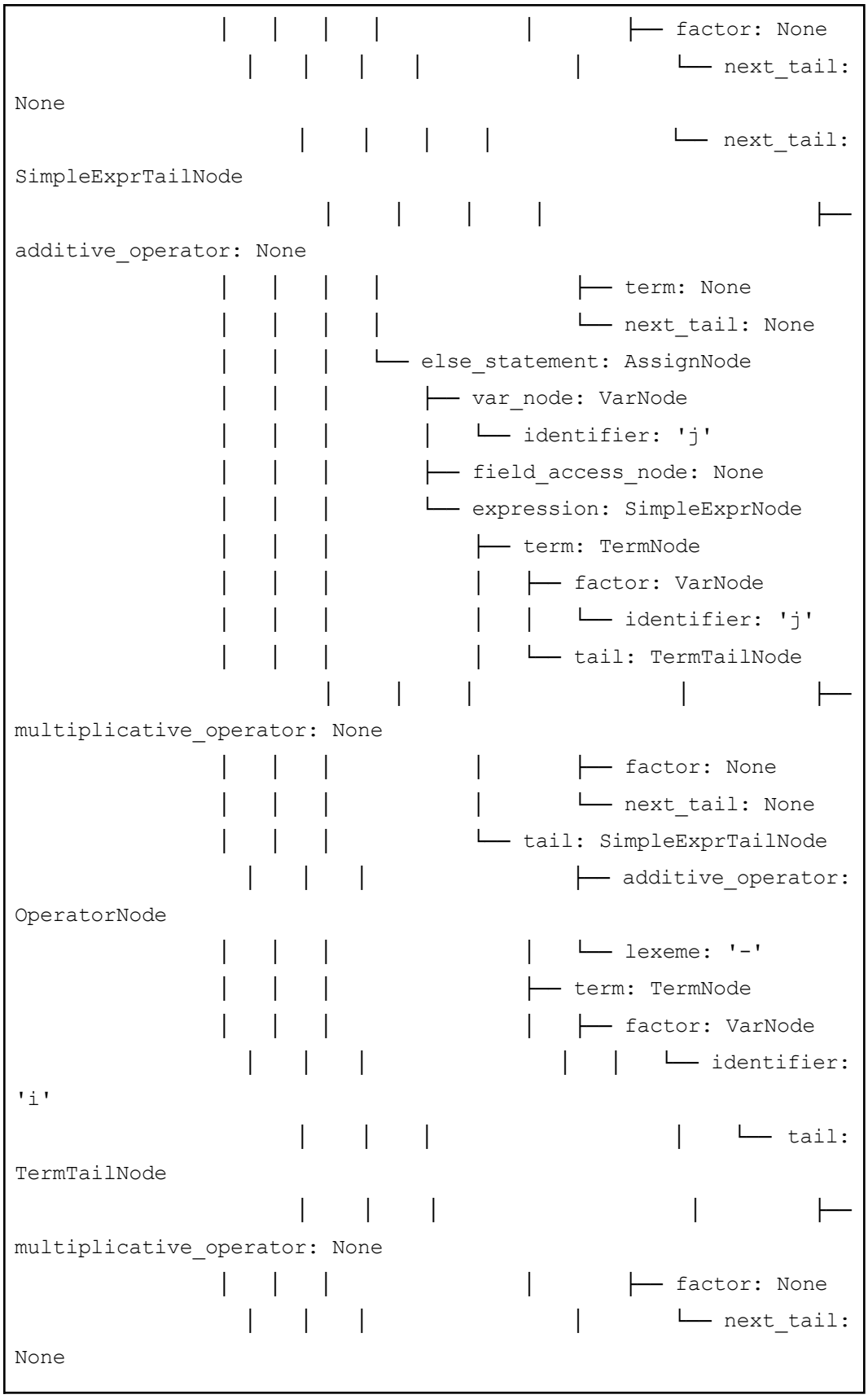

ParameterTailNode
param_group_node: None
expression_node: SimpleExprNode
term: TermNode
factor: FieldAccessNode
identifier_1: 'h2'
identifier_2: 'c'
tail: FieldAccessTailNode
identifier: 'im'
next_tail: FieldAccessTailNode
identifier: None
next_tail: None
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None

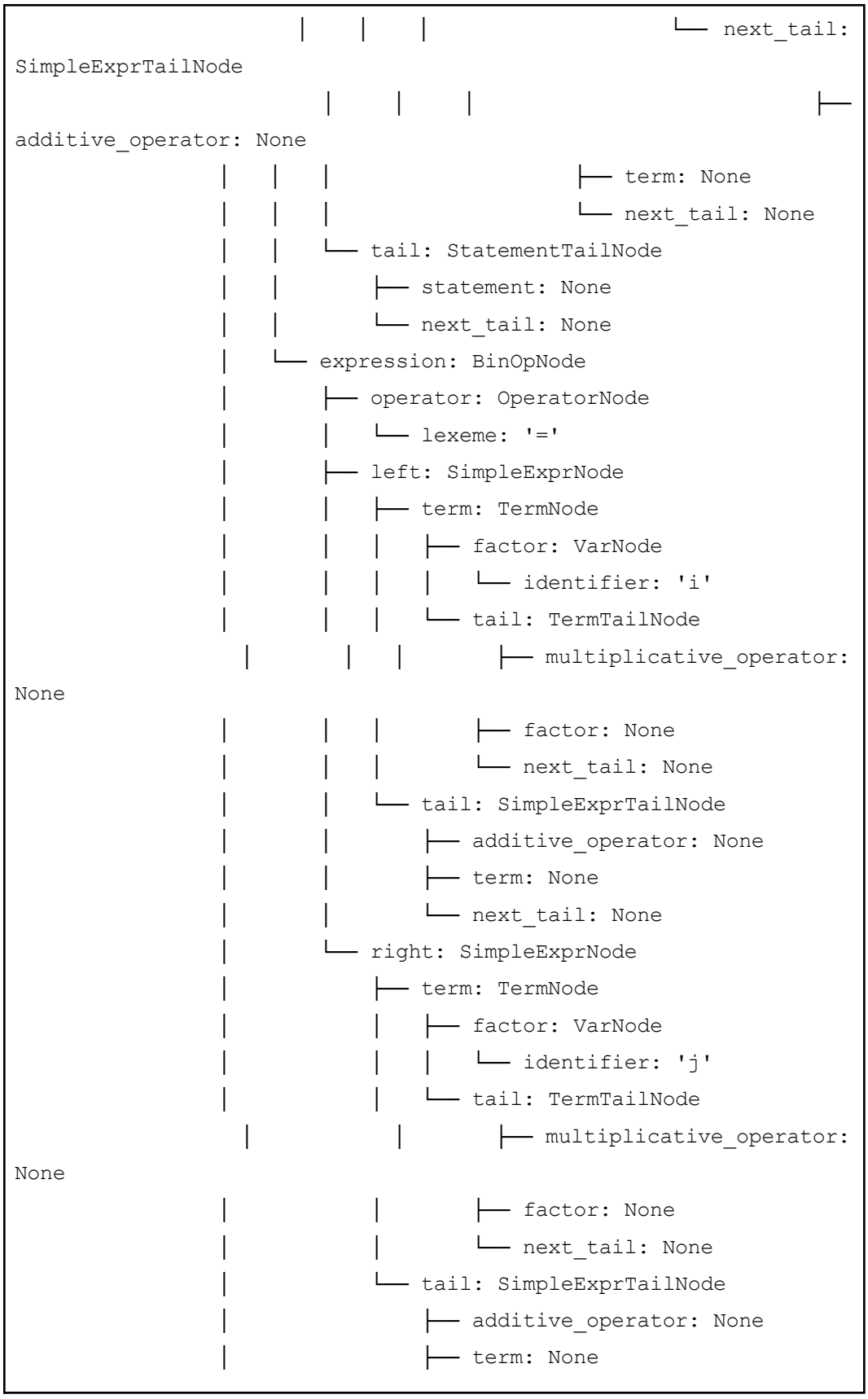

next_tail: None
└── next_tail: ParameterTailNode
├── param_group_node: None
├── expression_node: SimpleExprNode
├── term: TermNode
├── factor: VarNode
└── identifier: 'z'
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
└── next_tail: ParameterTailNode
├── param_group_node: None
├── expression_node: None
└──

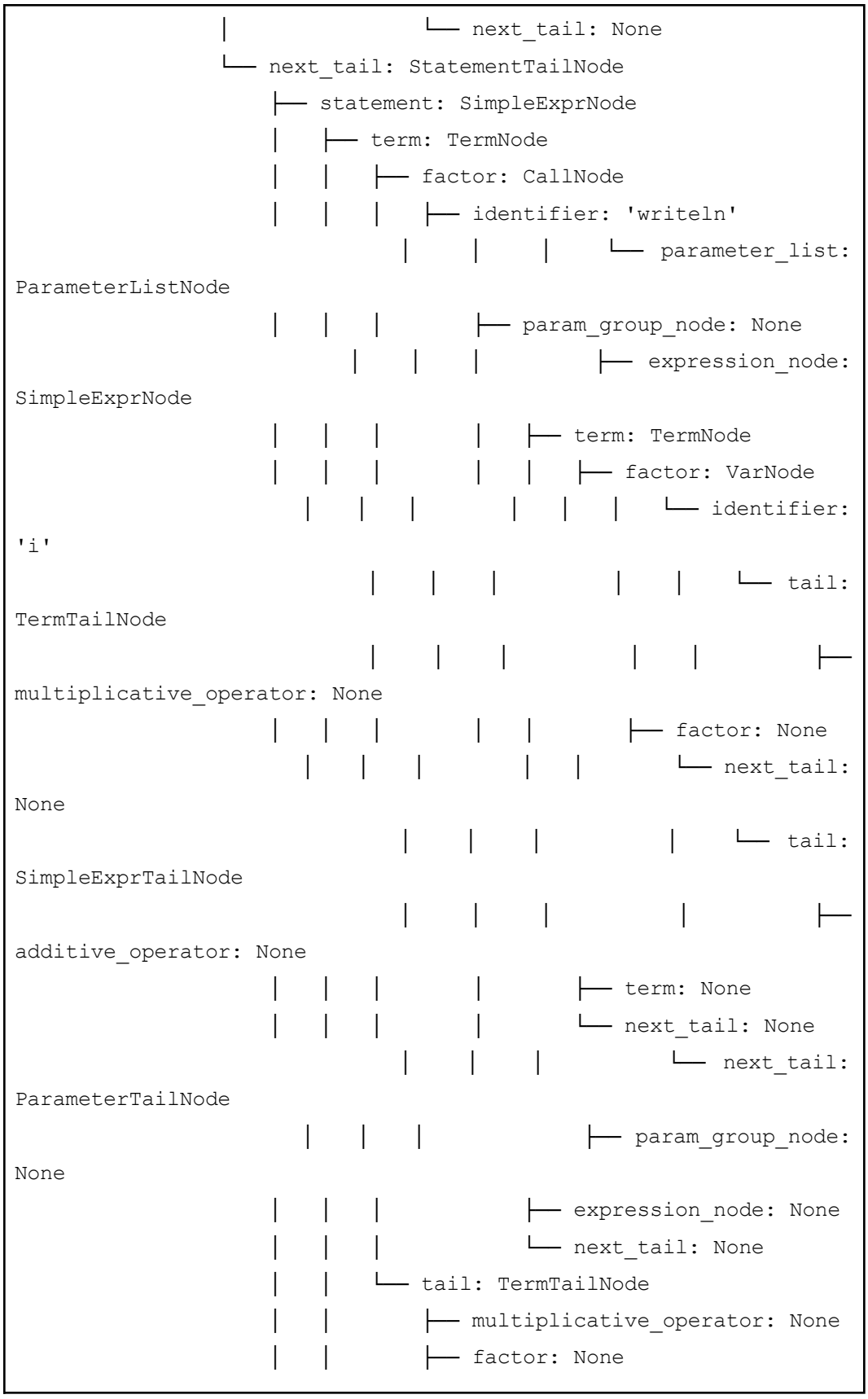
```
next_tail: None
|       |               |       |   └── tail: TermTailNode
|           |                       |           |           ├──
multiplicative_operator: None
|       |               |       |       ├── factor: None
|       |               |       |       └── next_tail: None
|       |               |       └── tail: SimpleExprTailNode
|       |               |           ├── additive_operator: None
|       |               |           ├── term: None
|       |               |           └── next_tail: None
|       |               └── tail: StatementTailNode
|       |                   ├── statement: None
|       |                   └── next_tail: None
|       └── next_section: SubprogramSectionNode
|           ├── subprogram_declaration: None
|           └── next_section: None
└── compound_statement: CompoundNode
    └── statement_list: StatementListNode
        ├── statement: AssignNode
        |   ├── var_node: VarNode
        |   |   └── identifier: 'i'
        |   ├── field_access_node: None
        |   └── expression: SimpleExprNode
        |       ├── term: TermNode
        |       |   ├── factor: NumberNode
        |       |   |   └── value: '85'
        |       |   └── tail: TermTailNode
        |       |       ├── multiplicative_operator: None
        |       |       ├── factor: None
        |       |       └── next_tail: None
        |       └── tail: SimpleExprTailNode
        |           ├── additive_operator: None
        |           ├── term: None
        |           └── next_tail: None
        └── tail: StatementTailNode
            ├── statement: AssignNode
            |   ├── var_node: VarNode
```


└── identifier: 'j'
├── field_access_node: None
└── expression: SimpleExprNode
├── term: TermNode
├── factor: NumberNode
└── value: '51'
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
└── next_tail: StatementTailNode
├── statement: RepeatNode
├── statement_list: StatementListNode
├── statement: IfNode
├── expression: BinOpNode
├── operator: OperatorNode
└── lexeme: '>'
├── left: SimpleExprNode
├── term: TermNode
├── factor: VarNode
└── identifier: 'i'
└── tail: TermTailNode
multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
None
├── additive_operator:
├── term: None
└── next_tail: None
└── right: SimpleExprNode
├── term: TermNode


│ │ │ │ │ ├── factor: VarNode
│ │ │ │ │ │ └── identifier: 'j'
│ │ │ │ │ └── tail: TermTailNode
│ │ │ │ │ │ ├──
multiplicative_operator: None
│ │ │ │ │ ├── factor: None
│ │ │ │ │ └── next_tail: None
│ │ │ │ └── tail: SimpleExprTailNode
│ │ │ │ ├── additive_operator:
None
│ │ │ │ ├── term: None
│ │ │ │ └── next_tail: None
│ │ │ ├── then_statement: AssignNode
│ │ │ │ ├── var_node: VarNode
│ │ │ │ │ └── identifier: 'i'
│ │ │ │ ├── field_access_node: None
│ │ │ │ └── expression: SimpleExprNode
│ │ │ │ ├── term: TermNode
│ │ │ │ │ ├── factor: VarNode
│ │ │ │ │ │ └── identifier: 'i'
│ │ │ │ │ └── tail: TermTailNode
│ │ │ │ │ │ ├──
multiplicative_operator: None
│ │ │ │ │ ├── factor: None
│ │ │ │ │ └── next_tail: None
│ │ │ │ └── tail: SimpleExprTailNode
│ │ │ │ ├── additive_operator:
OperatorNode
│ │ │ │ │ └── lexeme: '-'
│ │ │ │ ├── term: TermNode
│ │ │ │ │ ├── factor: VarNode
│ │ │ │ │ │ └── identifier:
'j'
│ │ │ │ │ └── tail:
TermTailNode
│ │ │ │ │ ├──
multiplicative_operator: None


│ │ │ │ │ ├── factor: None
│ │ │ │ │ └── next_tail: None
│ │ │ │ └── next_tail: SimpleExprTailNode
│ │ │ │ ├── additive_operator: None
│ │ │ │ ├── term: None
│ │ │ │ └── next_tail: None
│ │ │ └── else_statement: AssignNode
│ │ │ ├── var_node: VarNode
│ │ │ │ └── identifier: 'j'
│ │ │ ├── field_access_node: None
│ │ │ └── expression: SimpleExprNode
│ │ │ ├── term: TermNode
│ │ │ │ ├── factor: VarNode
│ │ │ │ │ └── identifier: 'j'
│ │ │ │ └── tail: TermTailNode
│ │ │ │ ├── multiplicative_operator: None
│ │ │ │ ├── factor: None
│ │ │ │ └── next_tail: None
│ │ │ └── tail: SimpleExprTailNode
│ │ │ ├── additive_operator: OperatorNode
│ │ │ │ └── lexeme: '-'
│ │ │ ├── term: TermNode
│ │ │ │ ├── factor: VarNode
│ │ │ │ │ └── identifier: 'i'
│ │ │ │ └── tail: TermTailNode
│ │ │ ├── multiplicative_operator: None
│ │ │ ├── factor: None
│ │ │ └── next_tail: None


└── next_tail:
SimpleExprTailNode
├──
additive_operator: None
├── term: None
└── next_tail: None
└── tail: StatementTailNode
├── statement: None
└── next_tail: None
└── expression: BinOpNode
├── operator: OperatorNode
│ └── lexeme: '='
├── left: SimpleExprNode
│ ├── term: TermNode
│ │ ├── factor: VarNode
│ │ │ └── identifier: 'i'
│ │ └── tail: TermTailNode
├── multiplicative_operator:
None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
└── right: SimpleExprNode
├── term: TermNode
│ ├── factor: VarNode
│ │ └── identifier: 'j'
│ └── tail: TermTailNode
├── multiplicative_operator:
None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: None
├── term: None

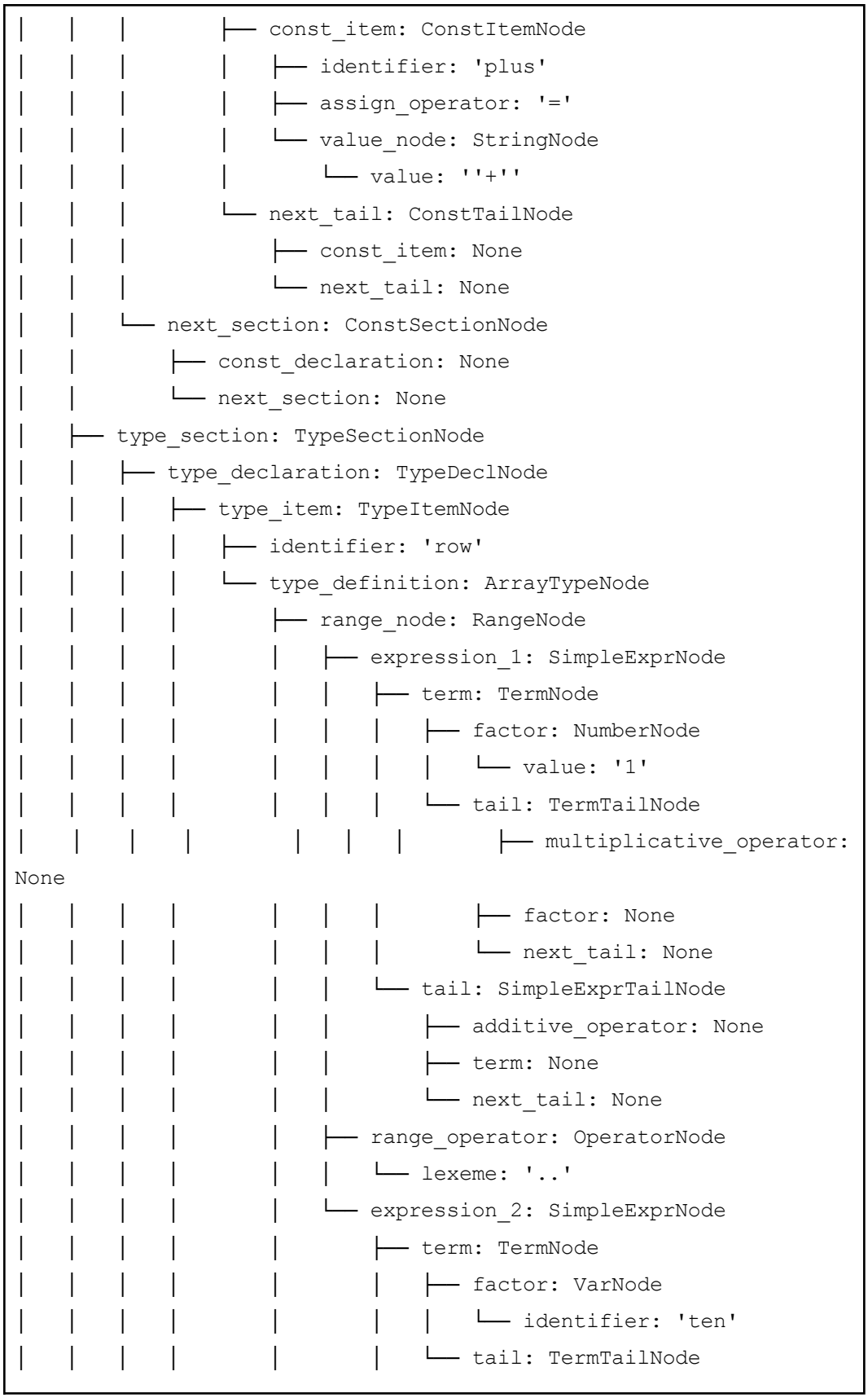

└── next_tail: None
└── next_tail: StatementTailNode
├── statement: SimpleExprNode
│ ├── term: TermNode
│ │ ├── factor: CallNode
│ │ │ ├── identifier: 'writeln'
│ │ │ └── parameter_list: ParameterListNode
│ │ │ ├── param_group_node: None
│ │ │ ├── expression_node: SimpleExprNode
│ │ │ ├── term: TermNode
│ │ │ ├── factor: VarNode
│ │ │ └── identifier: 'i'
│ │ │ └── tail: TermTailNode
│ │ │ ├── multiplicative_operator: None
│ │ │ ├── factor: None
│ │ │ └── next_tail: None
│ │ └── tail: SimpleExprTailNode
│ │ ├── additive_operator: None
│ │ ├── term: None
│ │ └── next_tail: None
│ └── next_tail: ParameterTailNode
│ ├── param_group_node: None
│ ├── expression_node: None
│ └── next_tail: None
└── tail: TermTailNode
├── multiplicative_operator: None
└── factor: None

```
                    │   │       └── next_tail: None
                    │   └── tail: SimpleExprTailNode
                    │       ├── additive_operator: None
                    │       ├── term: None
                    │       └── next_tail: None
                    └── next_tail: StatementTailNode
                        ├── statement: None
                        └── next_tail: None

================================================
Tahap: Semantic Analysis (Symbol Tables)
================================================
Semantic Error: Incompatible operands for '+'
```

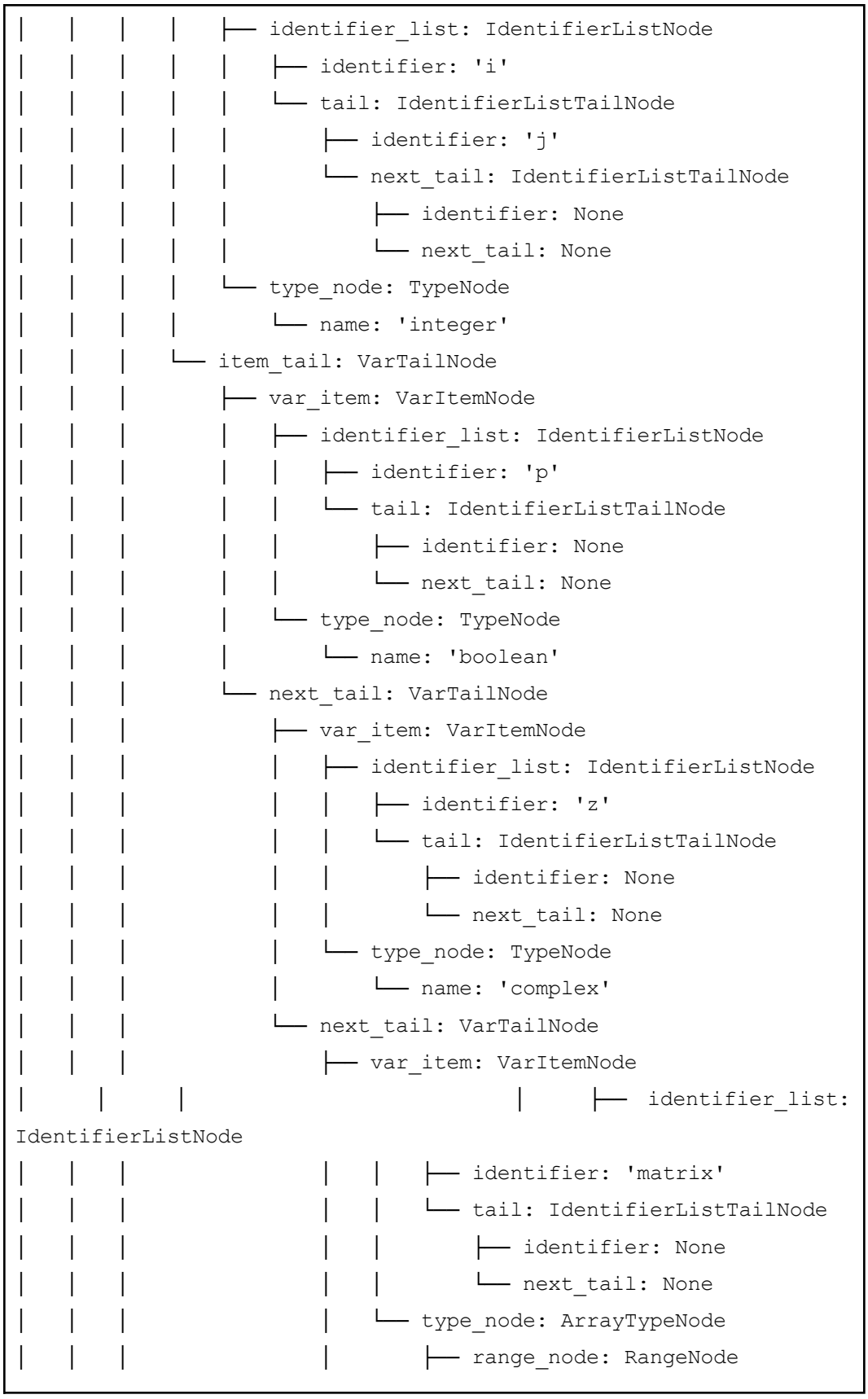
Pembahasan: Pengujian ini memverifikasi validasi tipe pada operasi aritmatika. Pesan error "Incompatible operands for '+'" muncul karena program mencoba menjumlahkan tipe `real` dengan `boolean`. Semantic Analyzer memastikan bahwa operasi matematika hanya dilakukan pada tipe data numerik yang valid.

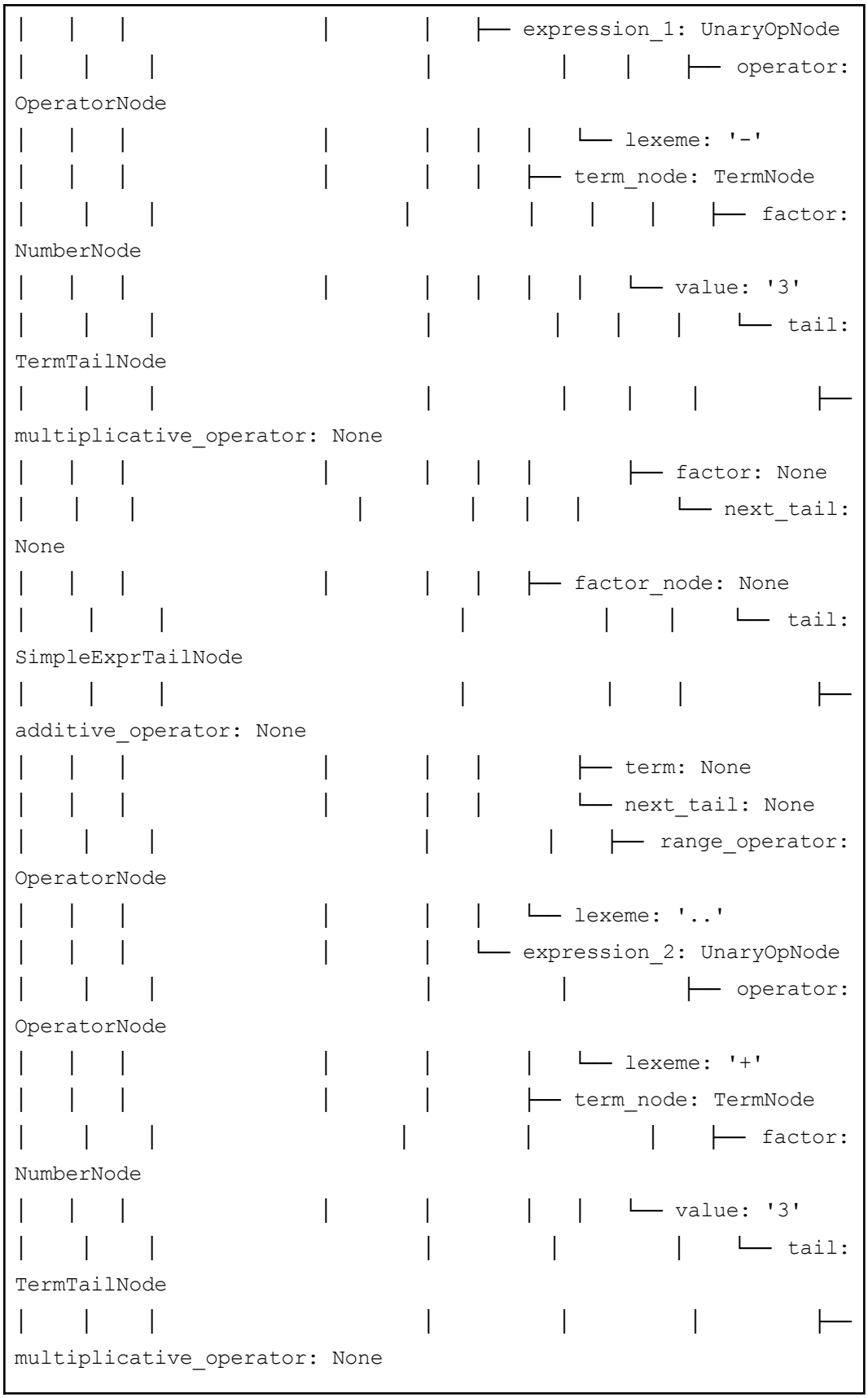
3.2.6 Hasil Uji 6

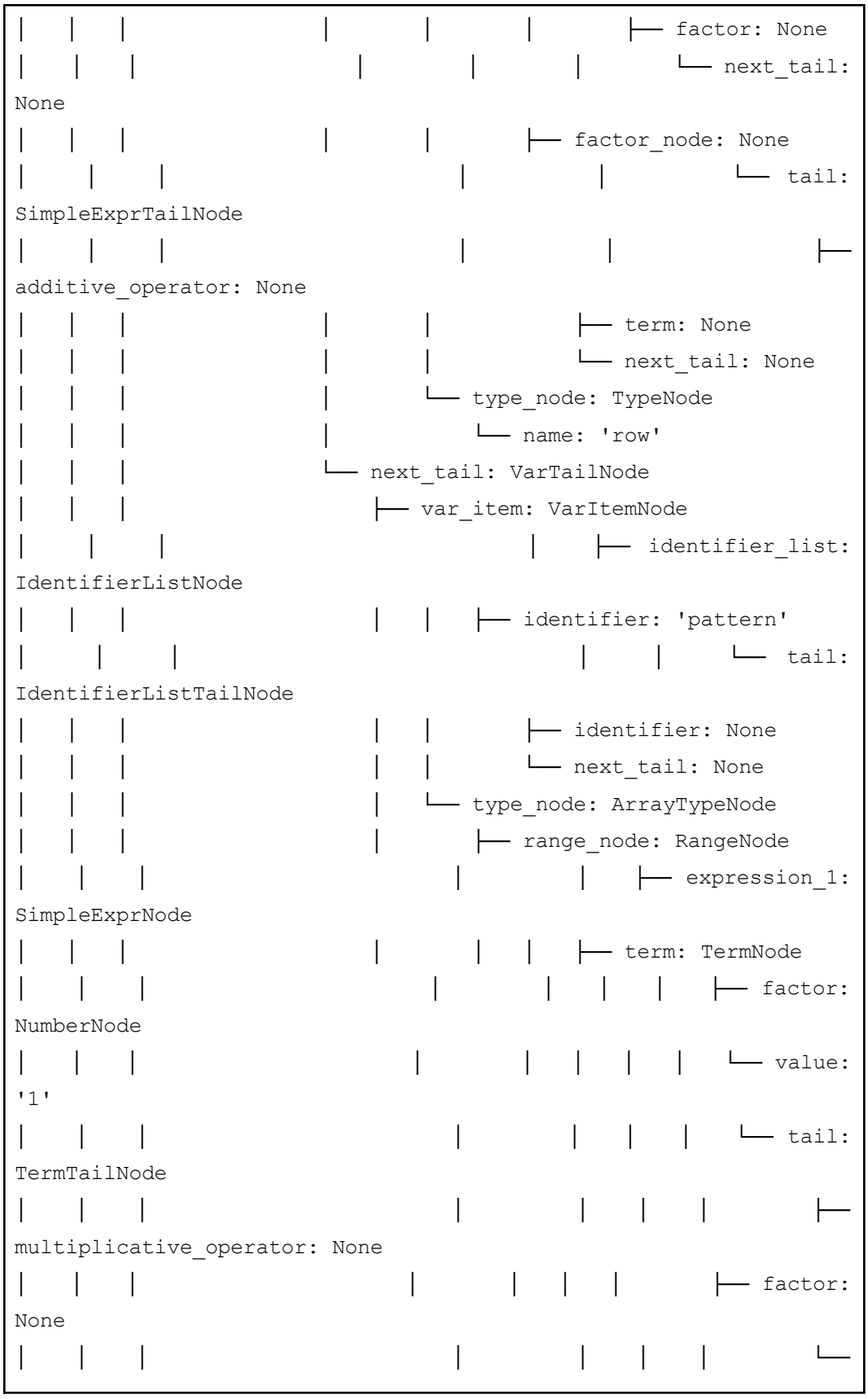
```
================================================
Tahap: AST Generation (Abstract Syntax Tree)
================================================
AST Structure:
------------------------------
ProgramNode
├── program_header: ProgramHeaderNode
│   └── identifier: 'test0'
├── declaration_part: DeclarationPartNode
│   ├── const_section: ConstSectionNode
│   │   ├── const_declaration: ConstDeclNode
│   │   │   ├── const_item: ConstItemNode
│   │   │   │   ├── identifier: 'ten'
│   │   │   │   ├── assign_operator: '='
│   │   │   │   └── value_node: NumberNode
│   │   │   │       └── value: '10'
│   │   │   └── item_tail: ConstTailNode
```

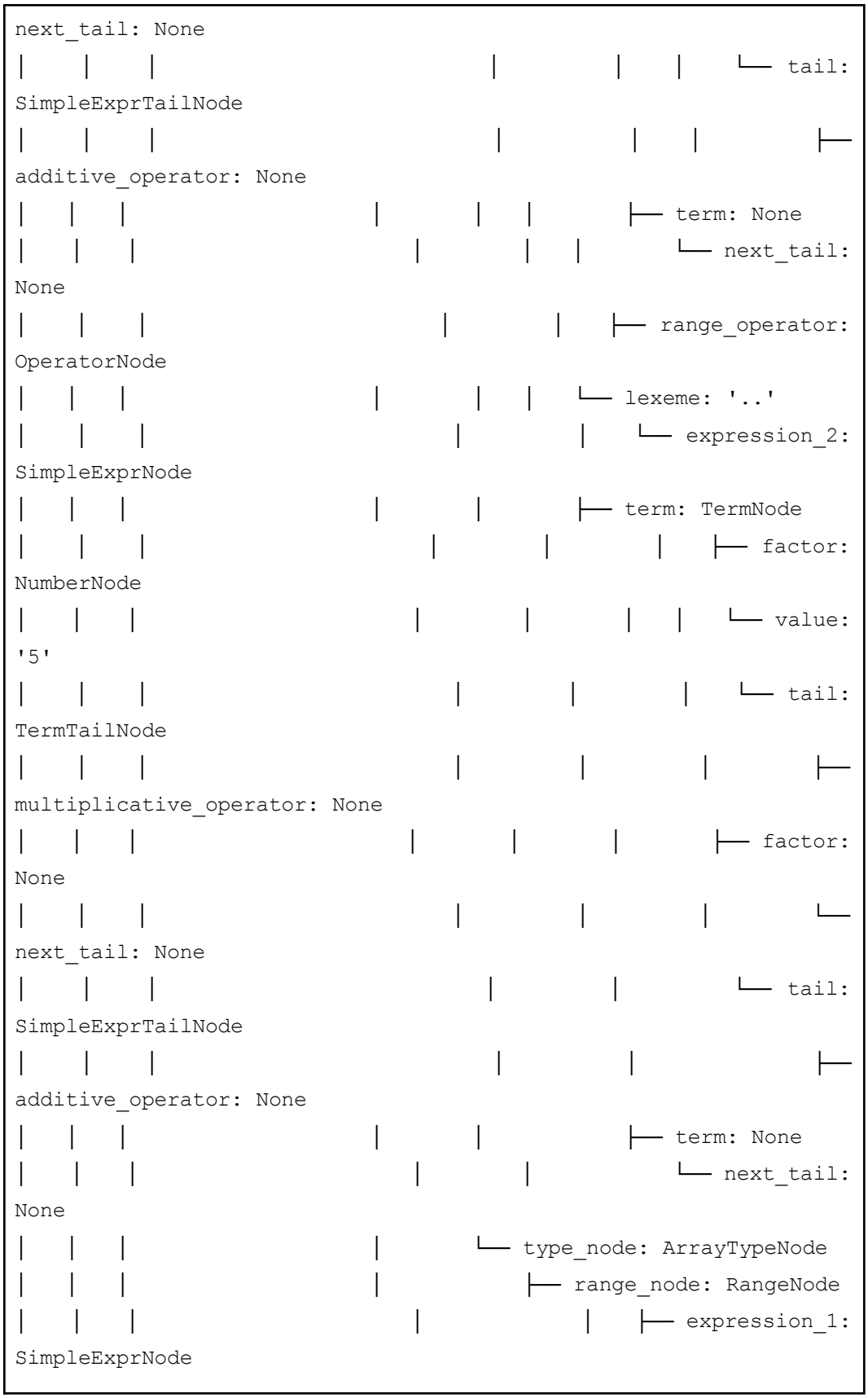

├── const_item: ConstItemNode
│ ├── identifier: 'plus'
│ ├── assign_operator: '='
│ └── value_node: StringNode
│ └── value: ''+''
└── next_tail: ConstTailNode
├── const_item: None
└── next_tail: None
└── next_section: ConstSectionNode
├── const_declaration: None
└── next_section: None
├── type_section: TypeSectionNode
├── type_declaration: TypeDeclNode
├── type_item: TypeItemNode
├── identifier: 'row'
└── type_definition: ArrayTypeNode
├── range_node: RangeNode
├── expression_1: SimpleExprNode
├── term: TermNode
├── factor: NumberNode
└── value: '1'
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
├── range_operator: OperatorNode
└── lexeme: '..'
└── expression_2: SimpleExprNode
├── term: TermNode
├── factor: VarNode
└── identifier: 'ten'
└── tail: TermTailNode

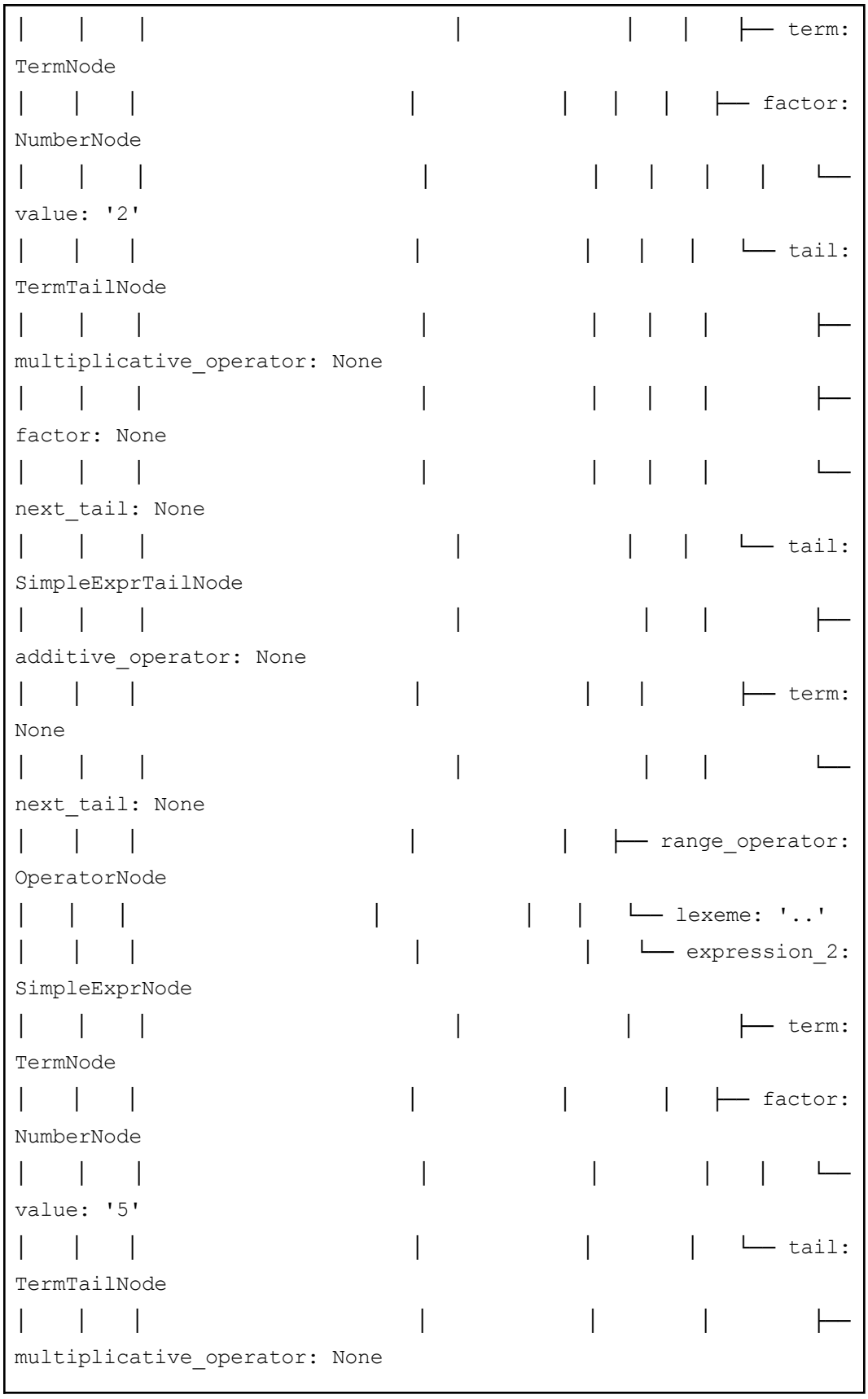
```
|   |   |   |       |       |       ├── multiplicative_operator:
None
|   |   |   |       |       |       ├── factor: None
|   |   |   |       |       |       └── next_tail: None
|   |   |   |       |       └── tail: SimpleExprTailNode
|   |   |   |       |           ├── additive_operator: None
|   |   |   |       |           ├── term: None
|   |   |   |       |           └── next_tail: None
|   |   |   |       └── type_node: TypeNode
|   |   |   |           └── name: 'real'
|   |   |   └── item_tail: TypeTailNode
|   |   |       ├── type_item: TypeItemNode
|   |   |       |   ├── identifier: 'complex'
|   |   |       |   └── type_definition: RecordTypeNode
|   |   |       |       └── field_list: FieldListNode
|     |     |           |                 ├── identifier_list:
IdentifierListNode
|   |   |       |           |   ├── identifier: 're'
|   |   |       |           |   └── tail: IdentifierListTailNode
|   |   |       |           |       ├── identifier: 'im'
|     |     |           |               |           └── next_tail:
IdentifierListTailNode
|   |   |       |           |           ├── identifier: None
|   |   |       |           |           └── next_tail: None
|   |   |       |           ├── type_definition: TypeNode
|   |   |       |           |   └── name: 'real'
|     |     |           |                 └── field_list_tail:
FieldListTailNode
|   |   |       └── next_tail: TypeTailNode
|   |   |           ├── type_item: None
|   |   |           └── next_tail: None
|   |   └── next_section: TypeSectionNode
|   |       ├── type_declaration: None
|   |       └── next_section: None
|   ├── var_section: VarSectionNode
|   |   ├── var_declaration: VarDeclNode
|   |   |   ├── var_item: VarItemNode
```


identifier_list: IdentifierListNode
identifier: 'i'
tail: IdentifierListTailNode
identifier: 'j'
next_tail: IdentifierListTailNode
identifier: None
next_tail: None
type_node: TypeNode
name: 'integer'
item_tail: VarTailNode
var_item: VarItemNode
identifier_list: IdentifierListNode
identifier: 'p'
tail: IdentifierListTailNode
identifier: None
next_tail: None
type_node: TypeNode
name: 'boolean'
next_tail: VarTailNode
var_item: VarItemNode
identifier_list: IdentifierListNode
identifier: 'z'
tail: IdentifierListTailNode
identifier: None
next_tail: None
type_node: TypeNode
name: 'complex'
next_tail: VarTailNode
var_item: VarItemNode
identifier_list: IdentifierListNode
identifier: 'matrix'
tail: IdentifierListTailNode
identifier: None
next_tail: None
type_node: ArrayTypeNode
range_node: RangeNode

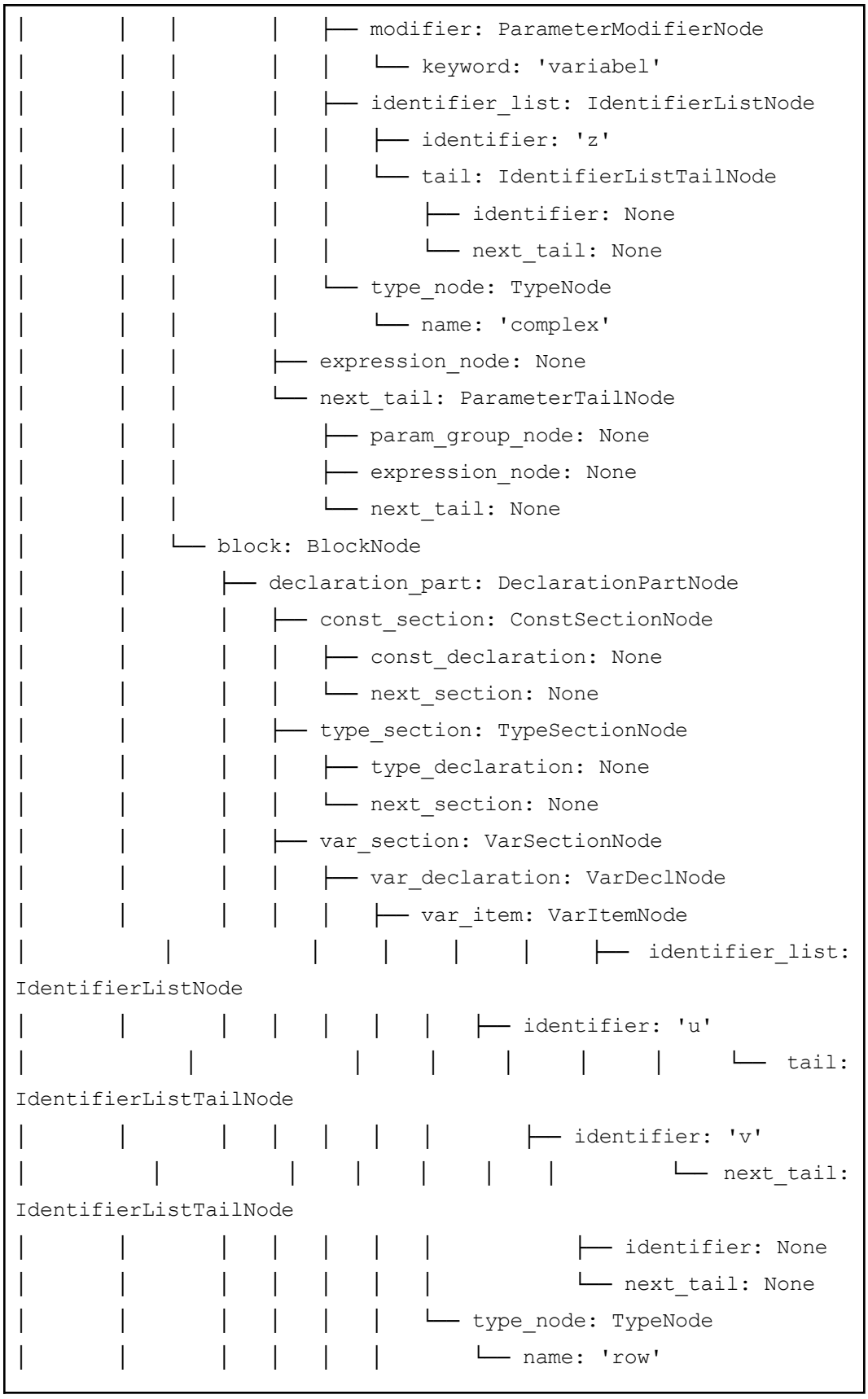

| | | | | | ├── expression_1: UnaryOpNode
| | | | | | | ├── operator:
OperatorNode
| | | | | | | | └── lexeme: '-'
| | | | | | | ├── term_node: TermNode
| | | | | | | | ├── factor:
NumberNode
| | | | | | | | | └── value: '3'
| | | | | | | | └── tail:
TermTailNode
| | | | | | | | ├──
multiplicative_operator: None
| | | | | | | ├── factor: None
| | | | | | | └── next_tail:
None
| | | | | | ├── factor_node: None
| | | | | | └── tail:
SimpleExprTailNode
| | | | | | ├──
additive_operator: None
| | | | | | ├── term: None
| | | | | | └── next_tail: None
| | | | | ├── range_operator:
OperatorNode
| | | | | | └── lexeme: '..'
| | | | | └── expression_2: UnaryOpNode
| | | | | | ├── operator:
OperatorNode
| | | | | | └── lexeme: '+'
| | | | | ├── term_node: TermNode
| | | | | | ├── factor:
NumberNode
| | | | | | └── value: '3'
| | | | | | └── tail:
TermTailNode
| | | | | | ├──
multiplicative_operator: None

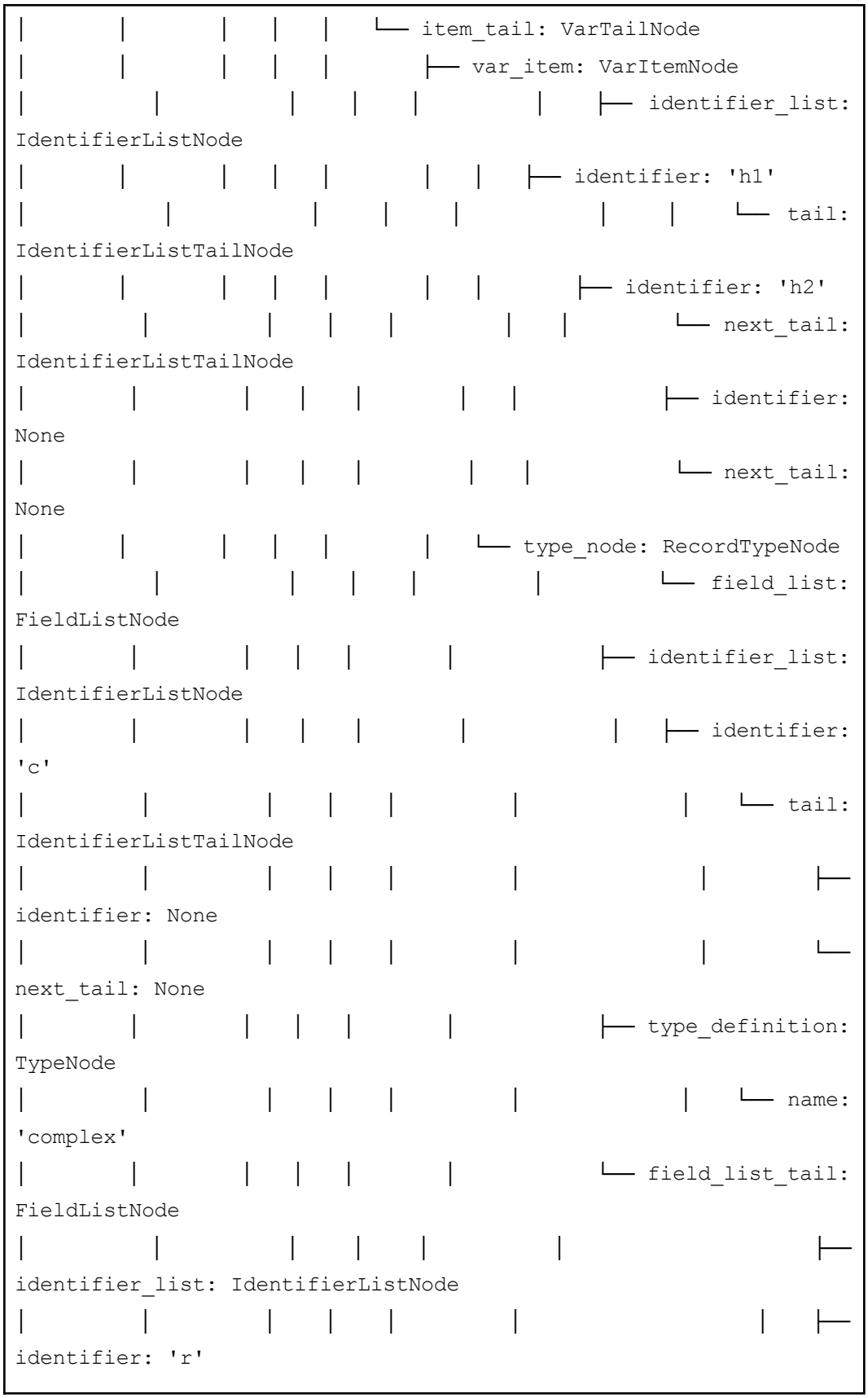

| | | | | | ├── factor: None
| | | | | | └── next_tail:
None
| | | | | ├── factor_node: None
| | | | | | └── tail:
SimpleExprTailNode
| | | | | | ├──
additive_operator: None
| | | | | ├── term: None
| | | | | └── next_tail: None
| | | | └── type_node: TypeNode
| | | | └── name: 'row'
| | | └── next_tail: VarTailNode
| | | ├── var_item: VarItemNode
| | | | ├── identifier_list:
IdentifierListNode
| | | | | ├── identifier: 'pattern'
| | | | | └── tail:
IdentifierListTailNode
| | | | | ├── identifier: None
| | | | | └── next_tail: None
| | | | └── type_node: ArrayTypeNode
| | | | ├── range_node: RangeNode
| | | | | ├── expression_1:
SimpleExprNode
| | | | | ├── term: TermNode
| | | | | | ├── factor:
NumberNode
| | | | | | | └── value:
'1'
| | | | | | └── tail:
TermTailNode
| | | | | ├──
multiplicative_operator: None
| | | | | ├── factor:
None
| | | | | └──

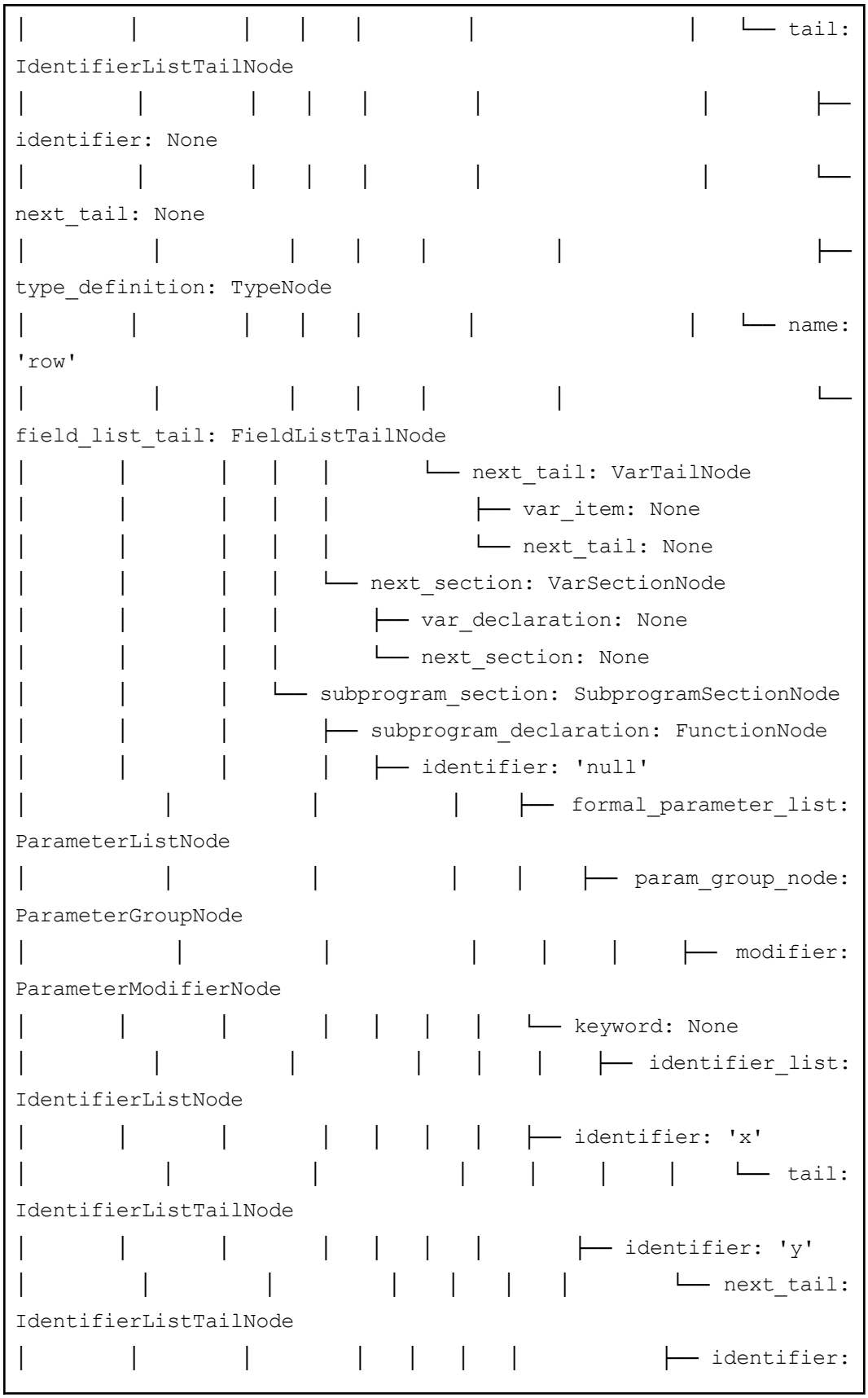

next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
range_operator: OperatorNode
lexeme: '..'
expression_2: SimpleExprNode
term: TermNode
factor: NumberNode
value: '5'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
type_node: ArrayTypeNode
range_node: RangeNode
expression_1: SimpleExprNode

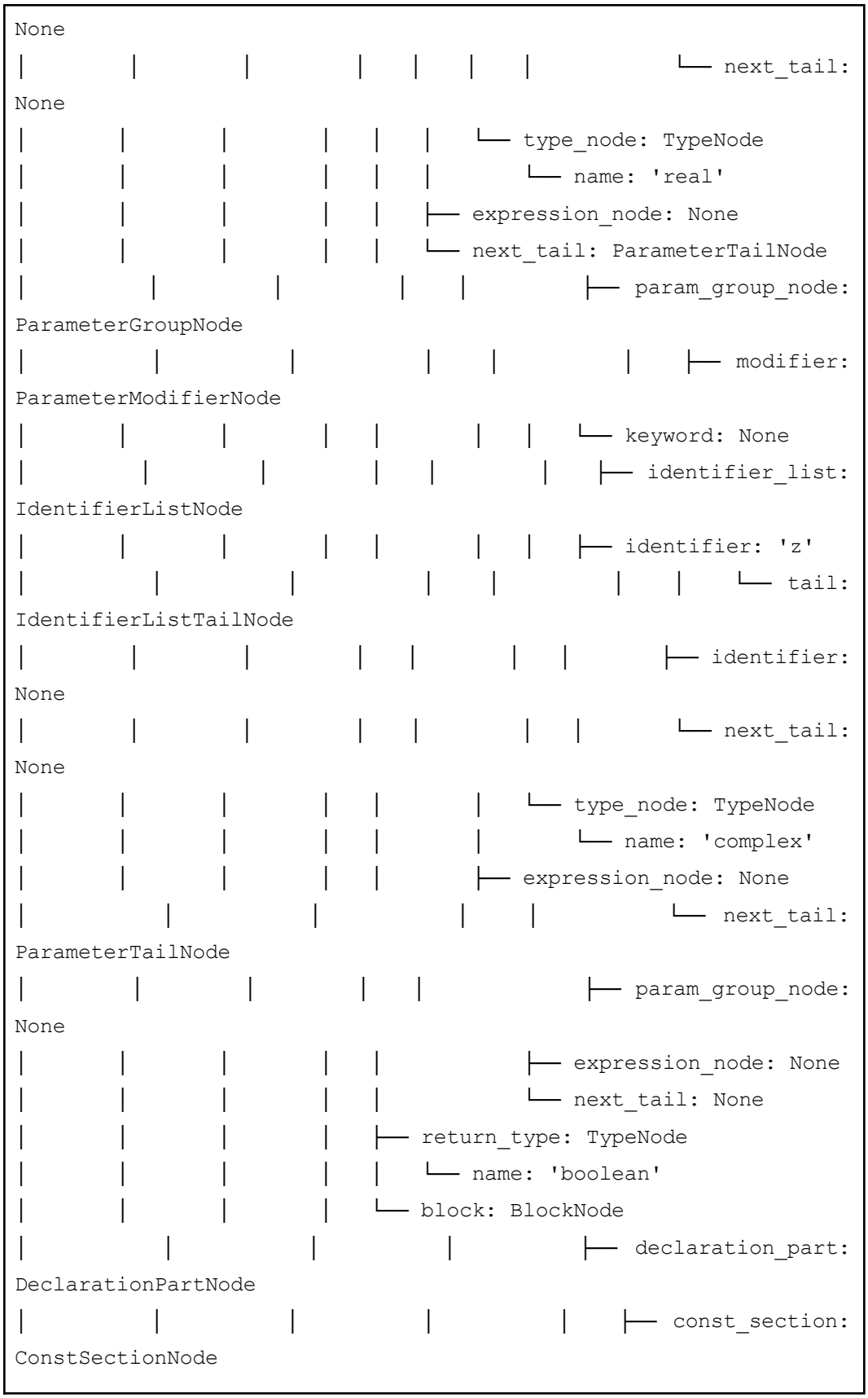

| | | | | | ├── term:
TermNode
| | | | | | | ├── factor:
NumberNode
| | | | | | | | └──
value: '2'
| | | | | | | └── tail:
TermTailNode
| | | | | | | ├──
multiplicative_operator: None
| | | | | | | ├──
factor: None
| | | | | | | └──
next_tail: None
| | | | | | └── tail:
SimpleExprTailNode
| | | | | | ├──
additive_operator: None
| | | | | | ├── term:
None
| | | | | | └──
next_tail: None
| | | | | ├── range_operator:
OperatorNode
| | | | | | └── lexeme: '..'
| | | | | └── expression_2:
SimpleExprNode
| | | | | ├── term:
TermNode
| | | | | | ├── factor:
NumberNode
| | | | | | | └──
value: '5'
| | | | | | └── tail:
TermTailNode
| | | | | | ├──
multiplicative_operator: None

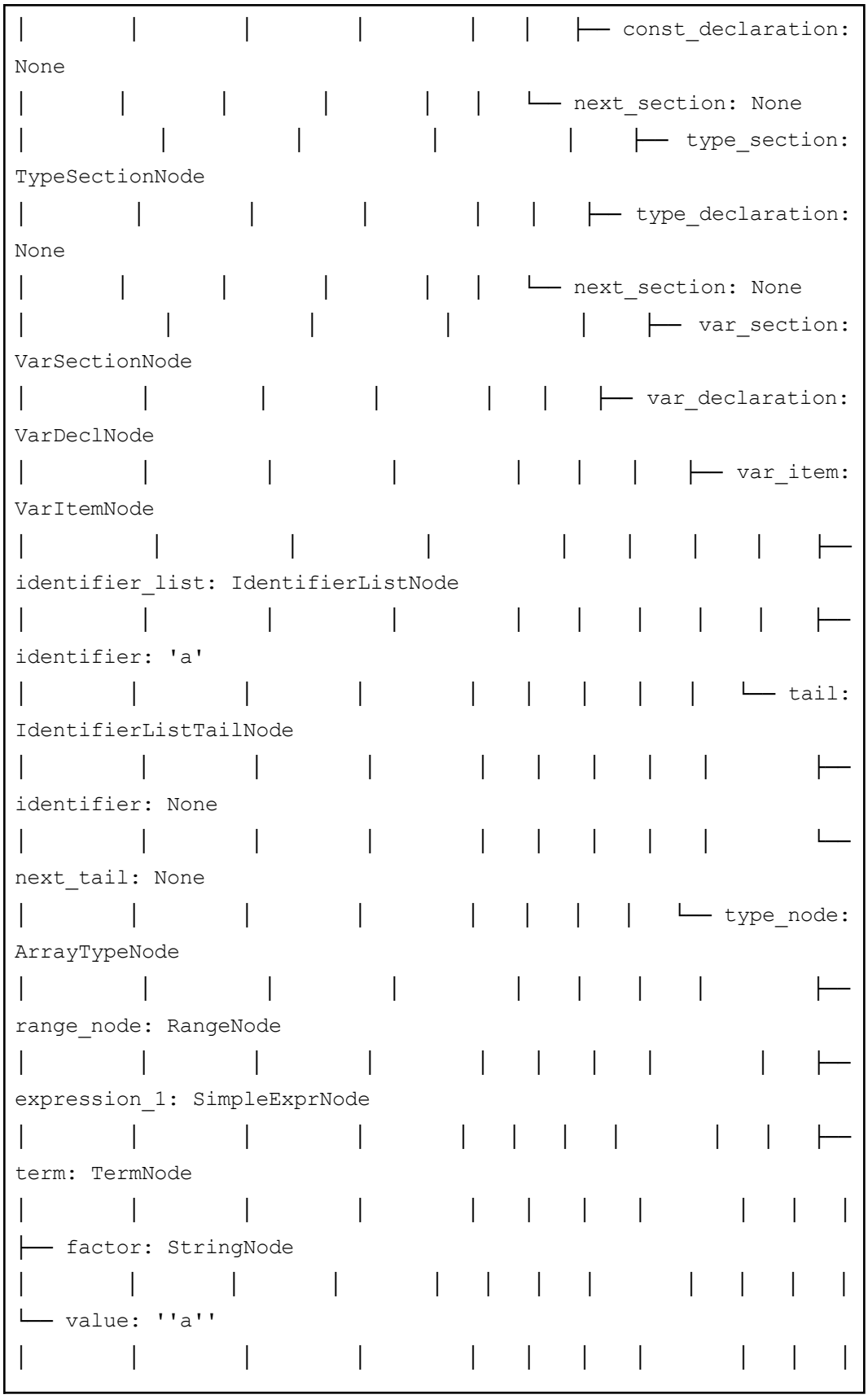
```
|   |   |                     |           |       |       ├─
factor: None
|   |   |                     |           |       |       └─
next_tail: None
|   |   |                       |           |       └─ tail:
SimpleExprTailNode
|   |   |                       |           |           ├─
additive_operator: None
|   |   |                     |           |           ├─ term:
None
|   |   |                       |           |           └─
next_tail: None
|   |   |                   |           └─ type_node: TypeNode
|   |   |                   |               └─ name: 'char'
|   |   |                   └─ next_tail: VarTailNode
|   |   |                       ├─ var_item: None
|   |   |                       └─ next_tail: None
|   |   └─ next_section: VarSectionNode
|   |       ├─ var_declaration: None
|   |       └─ next_section: None
|   └─ subprogram_section: SubprogramSectionNode
|       ├─ subprogram_declaration: ProcedureNode
|       |   ├─ identifier: 'dummy'
|       |   ├─ formal_parameter_list: ParameterListNode
|       |   |   ├─ param_group_node: ParameterGroupNode
|       |   |   |   ├─ modifier: ParameterModifierNode
|       |   |   |   |   └─ keyword: 'variabel'
|       |   |   |   ├─ identifier_list: IdentifierListNode
|       |   |   |   |   ├─ identifier: 'i'
|       |   |   |   |   └─ tail: IdentifierListTailNode
|       |   |   |   |       ├─ identifier: None
|       |   |   |   |       └─ next_tail: None
|       |   |   |   └─ type_node: TypeNode
|       |   |   |       └─ name: 'integer'
|       |   |   ├─ expression_node: None
|       |   |   └─ next_tail: ParameterTailNode
|       |   |       ├─ param_group_node: ParameterGroupNode
```

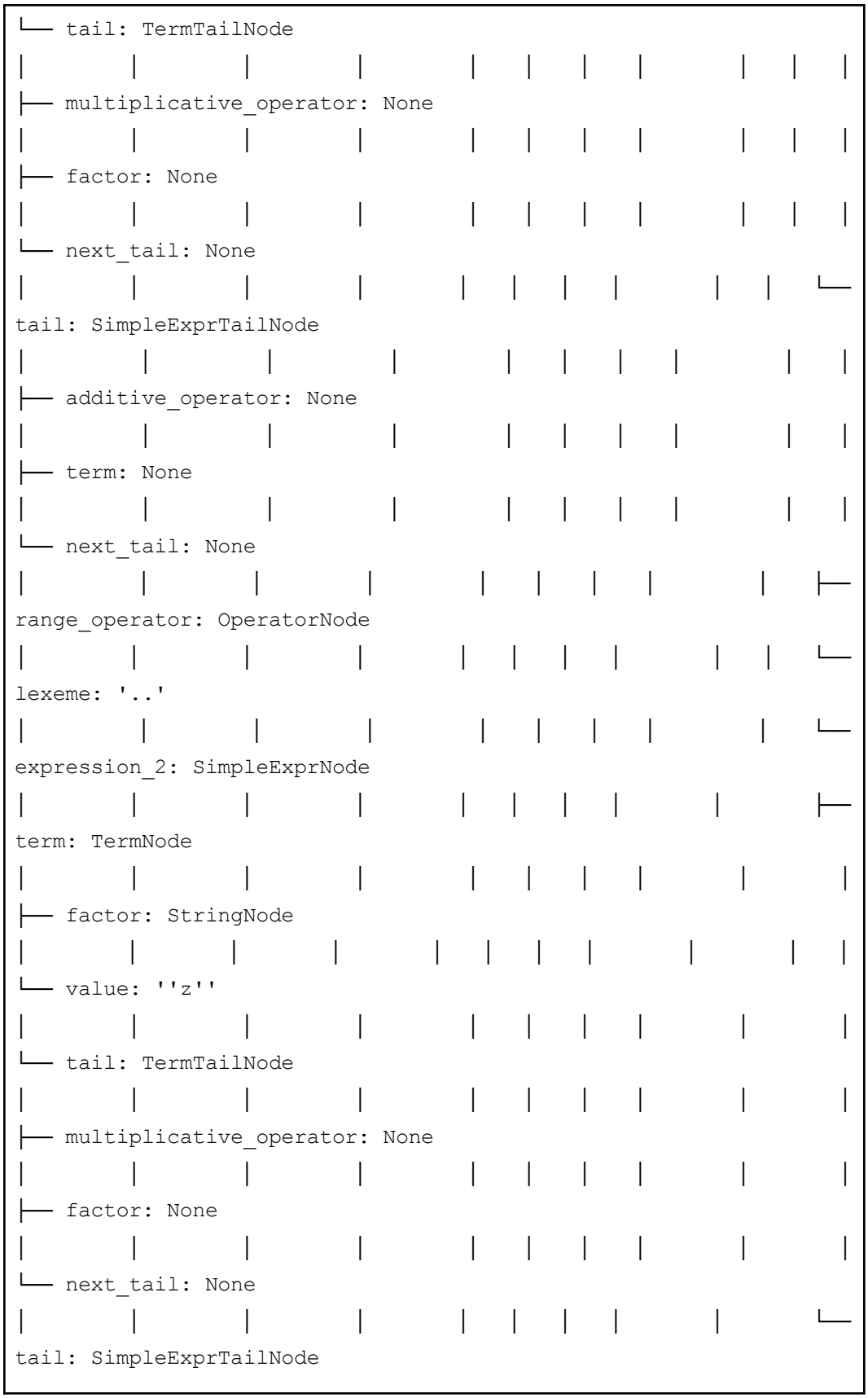

modifier: ParameterModifierNode
keyword: 'variabel'
identifier_list: IdentifierListNode
identifier: 'z'
tail: IdentifierListTailNode
identifier: None
next_tail: None
type_node: TypeNode
name: 'complex'
expression_node: None
next_tail: ParameterTailNode
param_group_node: None
expression_node: None
next_tail: None
block: BlockNode
declaration_part: DeclarationPartNode
const_section: ConstSectionNode
const_declaration: None
next_section: None
type_section: TypeSectionNode
type_declaration: None
next_section: None
var_section: VarSectionNode
var_declaration: VarDeclNode
var_item: VarItemNode
identifier_list: IdentifierListNode
identifier: 'u'
tail: IdentifierListTailNode
identifier: 'v'
next_tail: IdentifierListTailNode
identifier: None
next_tail: None
type_node: TypeNode
name: 'row'

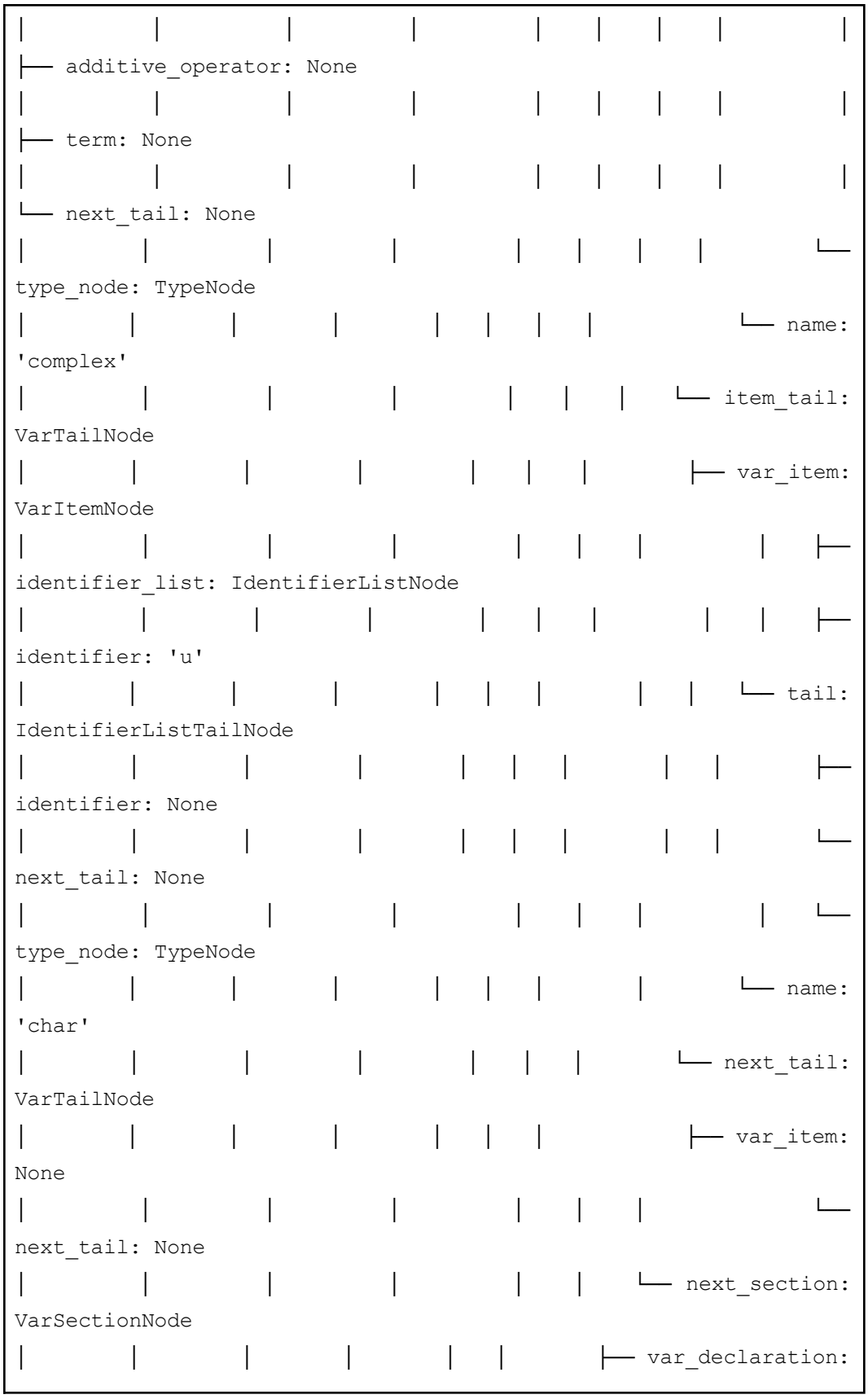

└── item_tail: VarTailNode
├── var_item: VarItemNode
├── identifier_list: IdentifierListNode
├── identifier: 'h1'
└── tail: IdentifierListTailNode
├── identifier: 'h2'
└── next_tail: IdentifierListTailNode
├── identifier: None
└── next_tail: None
└── type_node: RecordTypeNode
└── field_list: FieldListNode
├── identifier_list: IdentifierListNode
├── identifier: 'c'
└── tail: IdentifierListTailNode
├── identifier: None
└── next_tail: None
├── type_definition: TypeNode
└── name: 'complex'
└── field_list_tail: FieldListNode
├── identifier_list: IdentifierListNode
├── identifier: 'r'

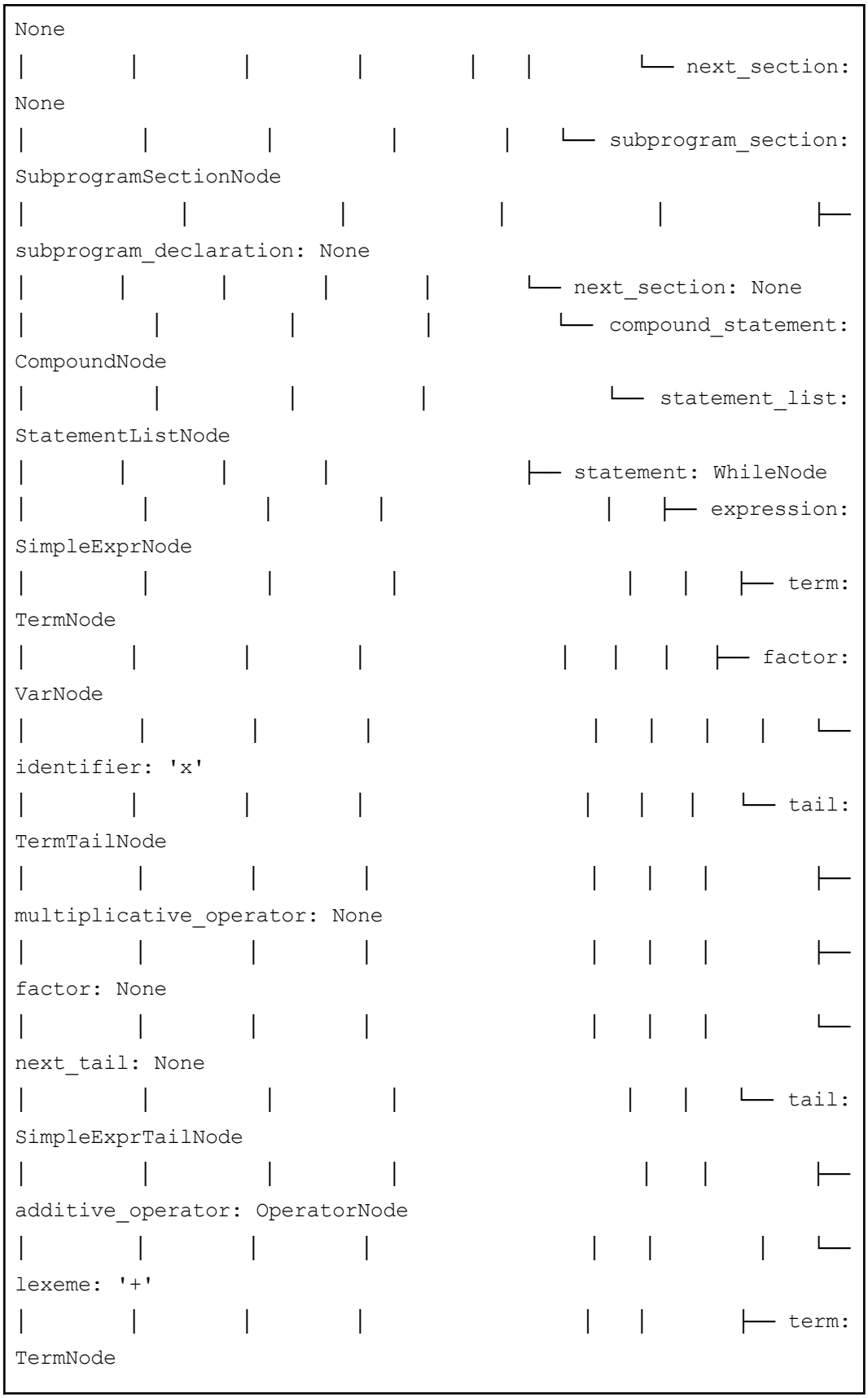

tail:
IdentifierListTailNode
identifier: None
next_tail: None
type_definition: TypeNode
name:
'row'
field_list_tail: FieldListTailNode
next_tail: VarTailNode
var_item: None
next_tail: None
next_section: VarSectionNode
var_declaration: None
next_section: None
subprogram_section: SubprogramSectionNode
subprogram_declaration: FunctionNode
identifier: 'null'
formal_parameter_list:
ParameterListNode
param_group_node:
ParameterGroupNode
modifier:
ParameterModifierNode
keyword: None
identifier_list:
IdentifierListNode
identifier: 'x'
tail:
IdentifierListTailNode
identifier: 'y'
next_tail:
IdentifierListTailNode
identifier:

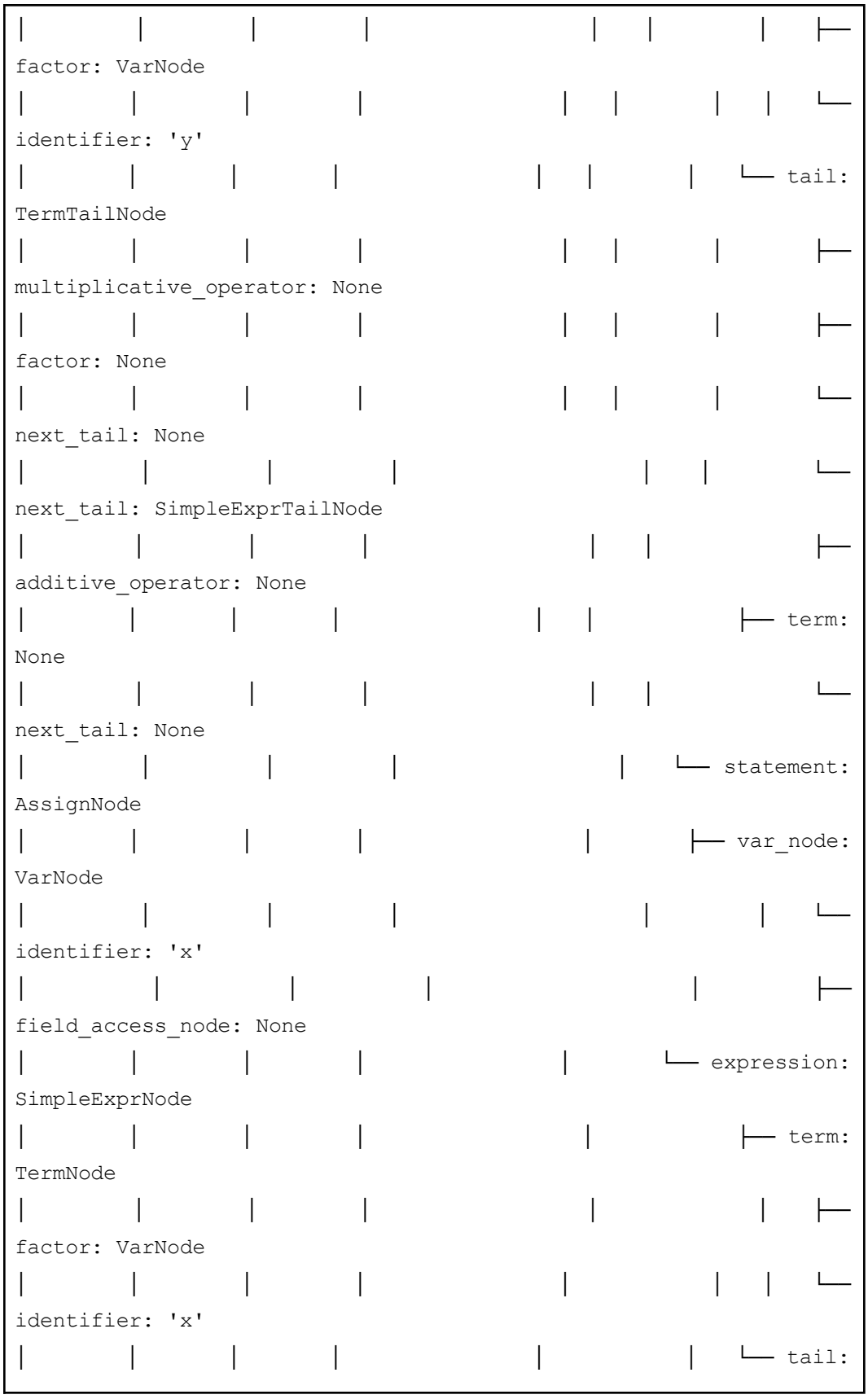

None
└── next_tail: None
└── type_node: TypeNode
└── name: 'real'
├── expression_node: None
└── next_tail: ParameterTailNode
├── param_group_node: ParameterGroupNode
├── modifier: ParameterModifierNode
└── keyword: None
├── identifier_list: IdentifierListNode
├── identifier: 'z'
└── tail: IdentifierListTailNode
├── identifier: None
└── next_tail: None
└── type_node: TypeNode
└── name: 'complex'
├── expression_node: None
└── next_tail: ParameterTailNode
├── param_group_node: None
├── expression_node: None
└── next_tail: None
├── return_type: TypeNode
└── name: 'boolean'
└── block: BlockNode
├── declaration_part: DeclarationPartNode
├── const_section: ConstSectionNode

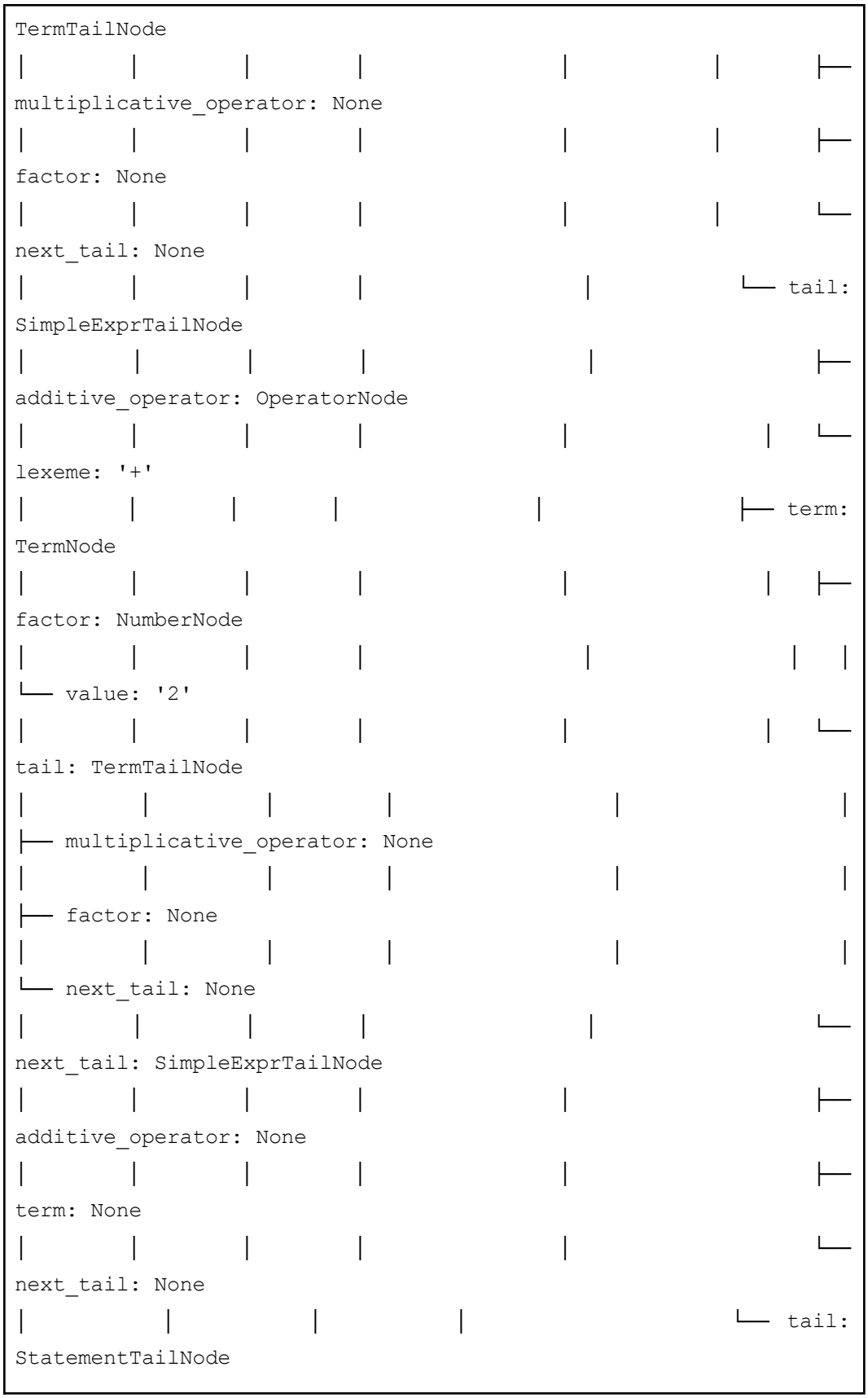

| | | | | | ├── const_declaration: None
| | | | | | └── next_section: None
| | | | | ├── type_section: TypeSectionNode
| | | | | | ├── type_declaration: None
| | | | | | └── next_section: None
| | | | | ├── var_section: VarSectionNode
| | | | | | ├── var_declaration: VarDeclNode
| | | | | | | ├── var_item: VarItemNode
| | | | | | | | ├── identifier_list: IdentifierListNode
| | | | | | | | | ├── identifier: 'a'
| | | | | | | | | └── tail: IdentifierListTailNode
| | | | | | | | | ├── identifier: None
| | | | | | | | | └── next_tail: None
| | | | | | | | └── type_node: ArrayTypeNode
| | | | | | | | ├── range_node: RangeNode
| | | | | | | | | ├── expression_1: SimpleExprNode
| | | | | | | | | | ├── term: TermNode
| | | | | | | | | | | ├── factor: StringNode
| | | | | | | | | | | | └── value: ''a''
| | | | | | | | | | | |

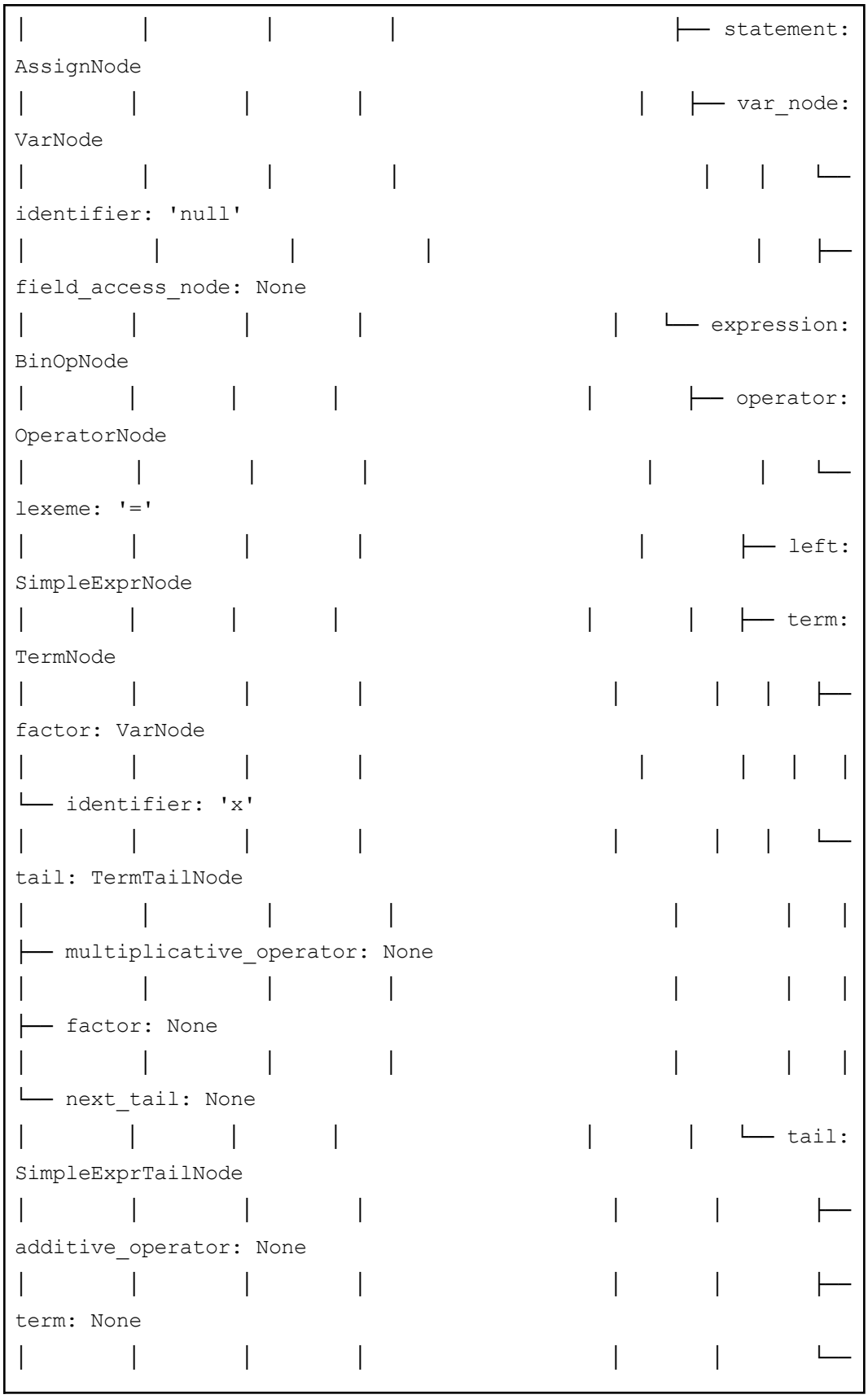

└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
range_operator: OperatorNode
lexeme: '..'
expression_2: SimpleExprNode
term: TermNode
├── factor: StringNode
└── value: ''z''
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
tail: SimpleExprTailNode

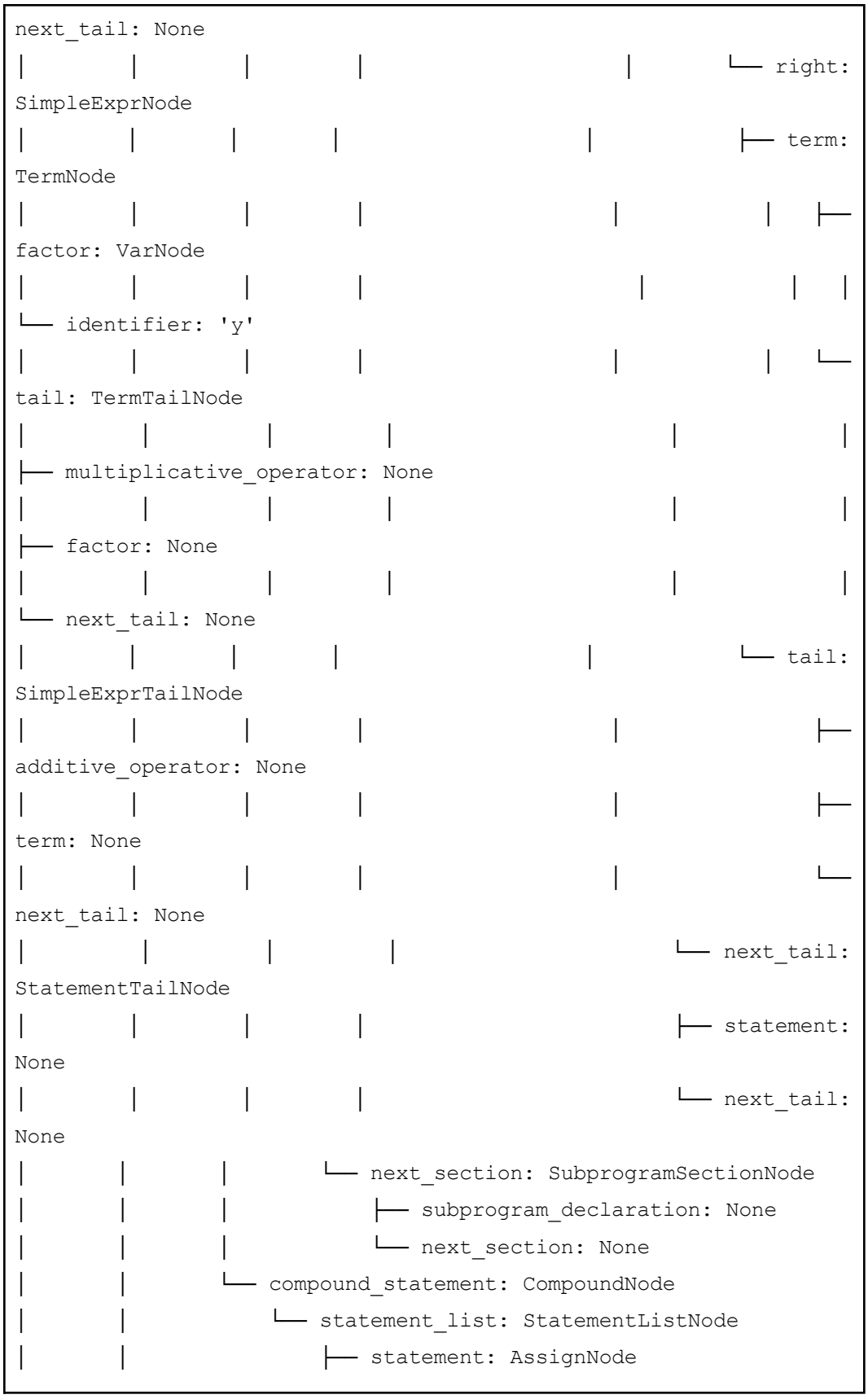

├── additive_operator: None
├── term: None
└── next_tail: None
type_node: TypeNode
└── name: 'complex'
└── item_tail: VarTailNode
├── var_item: VarItemNode
├── identifier_list: IdentifierListNode
├── identifier: 'u'
└── tail: IdentifierListTailNode
├── identifier: None
└── next_tail: None
type_node: TypeNode
└── name: 'char'
└── next_tail: VarTailNode
├── var_item: None
└── next_tail: None
└── next_section: VarSectionNode
├── var_declaration:

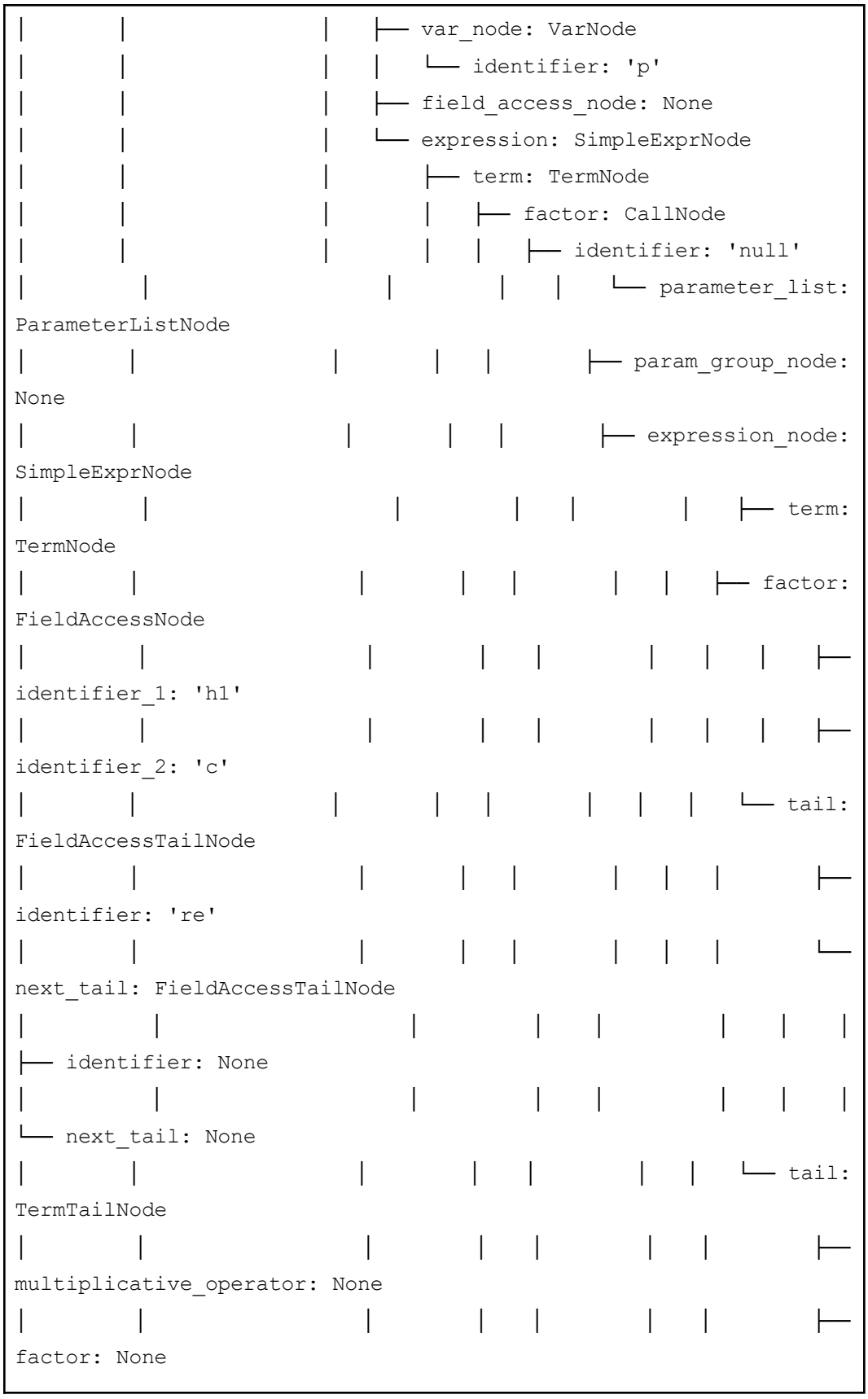

None
└── next_section: None
└── subprogram_section: SubprogramSectionNode
├── subprogram_declaration: None
└── next_section: None
└── compound_statement: CompoundNode
└── statement_list: StatementListNode
├── statement: WhileNode
├── expression: SimpleExprNode
├── term: TermNode
├── factor: VarNode
└── identifier: 'x'
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: OperatorNode
└── lexeme: '+'
├── term: TermNode

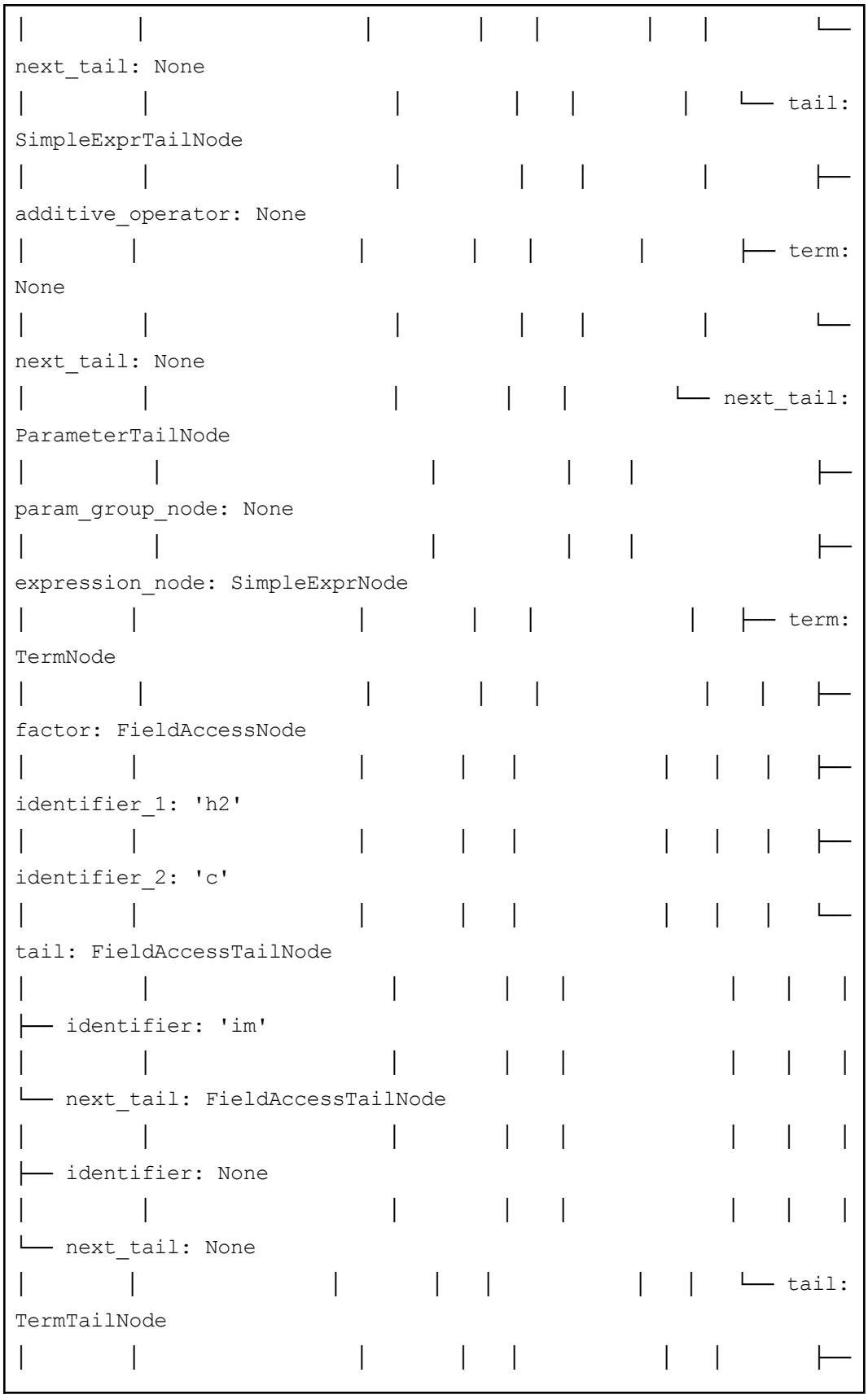

factor: VarNode
identifier: 'y'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
next_tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
statement: AssignNode
var_node: VarNode
identifier: 'x'
field_access_node: None
expression: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'x'
tail:

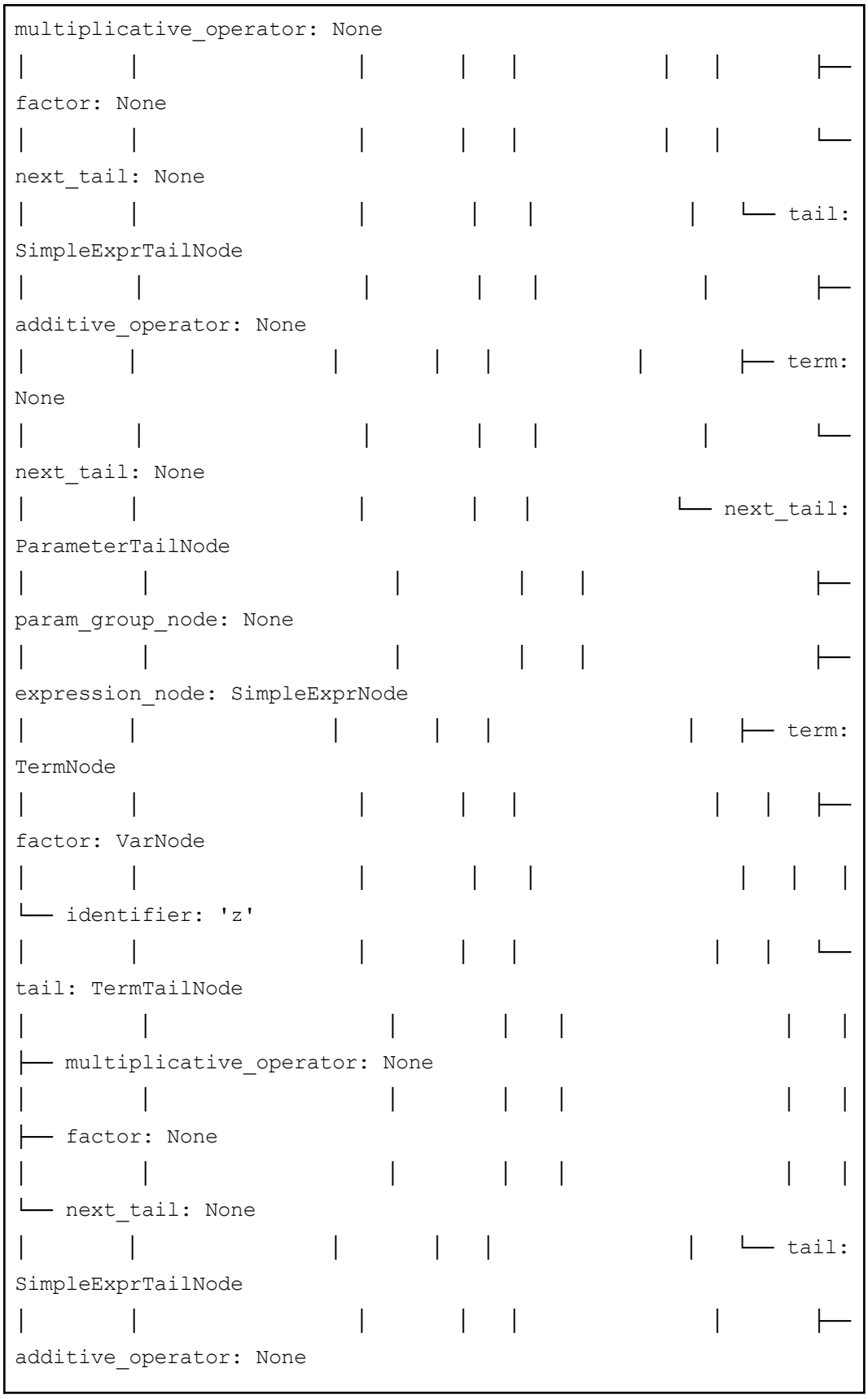

TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: OperatorNode
lexeme: '+'
term: TermNode
factor: NumberNode
value: '2'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
next_tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
tail: StatementTailNode

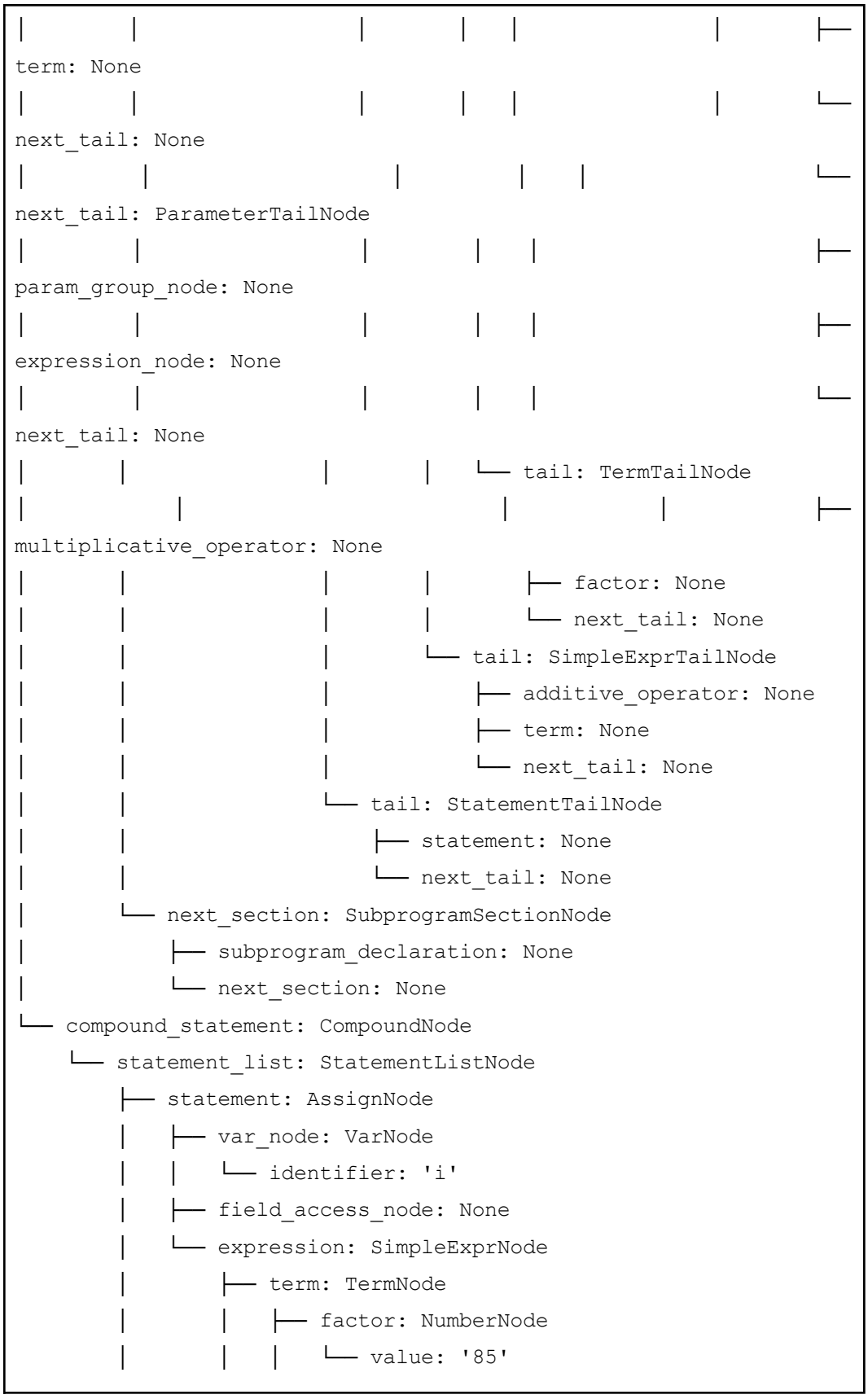

| | | | ├── statement:
AssignNode
| | | | | ├── var_node:
VarNode
| | | | | | └──
identifier: 'null'
| | | | | ├──
field_access_node: None
| | | | | └── expression:
BinOpNode
| | | | | | ├── operator:
OperatorNode
| | | | | | └──
lexeme: '='
| | | | | ├── left:
SimpleExprNode
| | | | | | ├── term:
TermNode
| | | | | | | ├──
factor: VarNode
| | | | | | | | └── identifier: 'x'
| | | | | | | └──
tail: TermTailNode
| | | | | | | ├── multiplicative_operator: None
| | | | | | | ├── factor: None
| | | | | | | └── next_tail: None
| | | | | | └── tail:
SimpleExprTailNode
| | | | | | ├──
additive_operator: None
| | | | | | ├──
term: None
| | | | | | └──

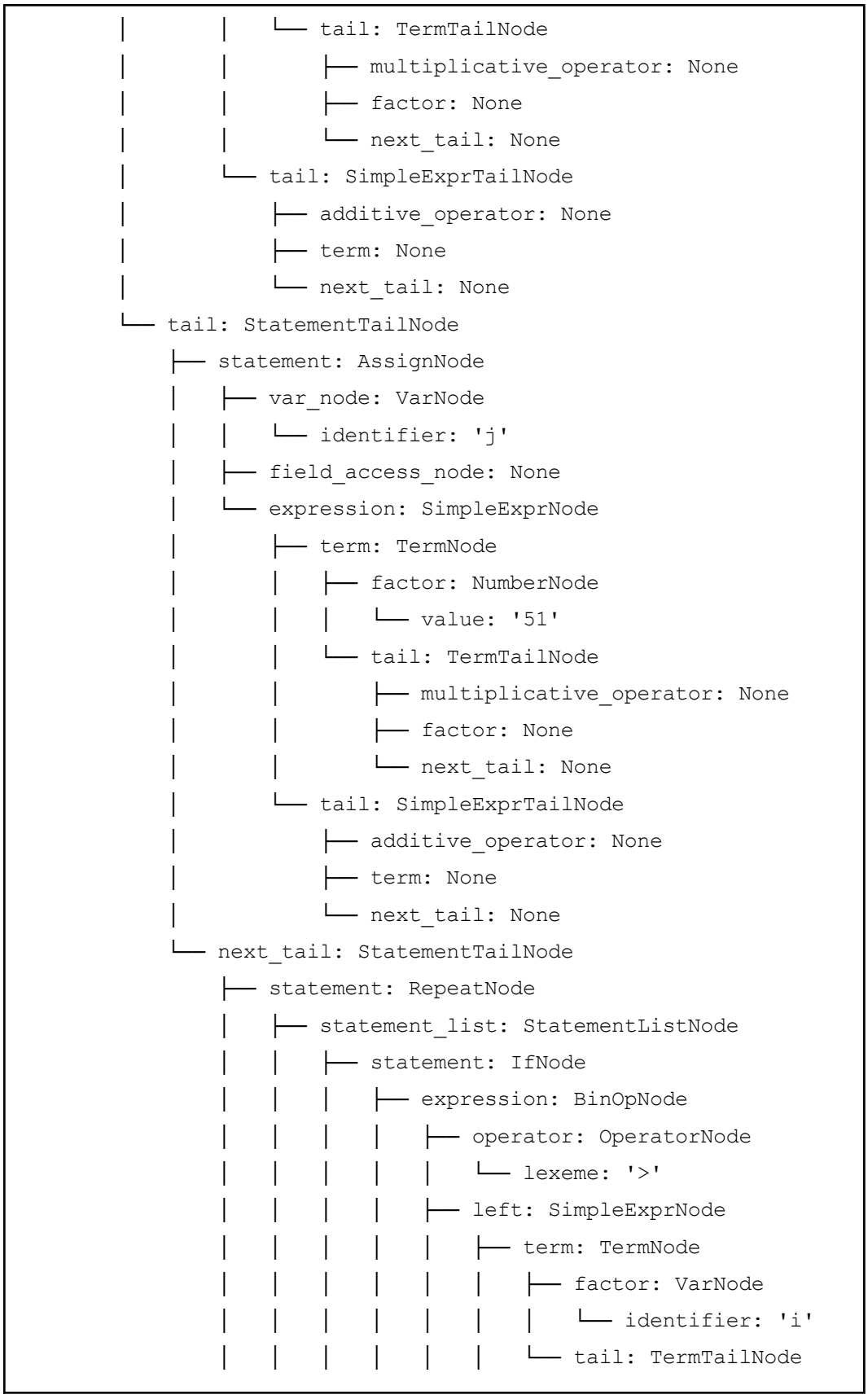

next_tail: None
└── right: SimpleExprNode
├── term: TermNode
├── factor: VarNode
└── identifier: 'y'
└── tail: TermTailNode
├── multiplicative_operator: None
├── factor: None
└── next_tail: None
└── tail: SimpleExprTailNode
├── additive_operator: None
├── term: None
└── next_tail: None
└── next_tail: StatementTailNode
├── statement: None
└── next_tail: None
└── next_section: SubprogramSectionNode
├── subprogram_declaration: None
└── next_section: None
└── compound_statement: CompoundNode
└── statement_list: StatementListNode
├── statement: AssignNode

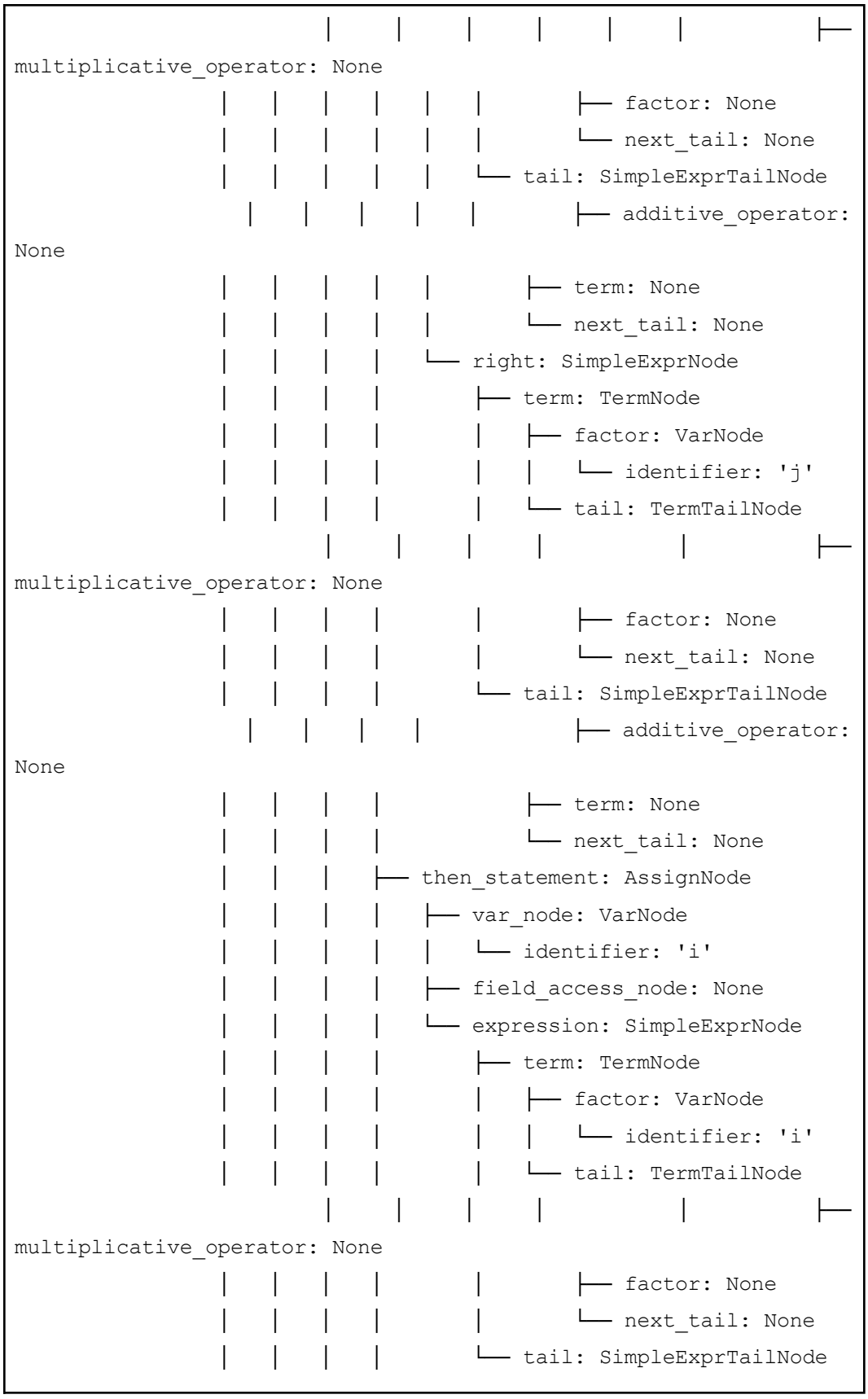

| | | ├── var_node: VarNode
| | | | └── identifier: 'p'
| | | ├── field_access_node: None
| | | └── expression: SimpleExprNode
| | | ├── term: TermNode
| | | | ├── factor: CallNode
| | | | | ├── identifier: 'null'
| | | | | └── parameter_list: ParameterListNode
| | | | | ├── param_group_node: None
| | | | | ├── expression_node: SimpleExprNode
| | | | | ├── term: TermNode
| | | | | | ├── factor: FieldAccessNode
| | | | | | | ├── identifier_1: 'h1'
| | | | | | | ├── identifier_2: 'c'
| | | | | | | └── tail: FieldAccessTailNode
| | | | | | | ├── identifier: 're'
| | | | | | | └── next_tail: FieldAccessTailNode
| | | | | | | ├── identifier: None
| | | | | | | └── next_tail: None
| | | | | | └── tail: TermTailNode
| | | | | | ├── multiplicative_operator: None
| | | | | | ├── factor: None

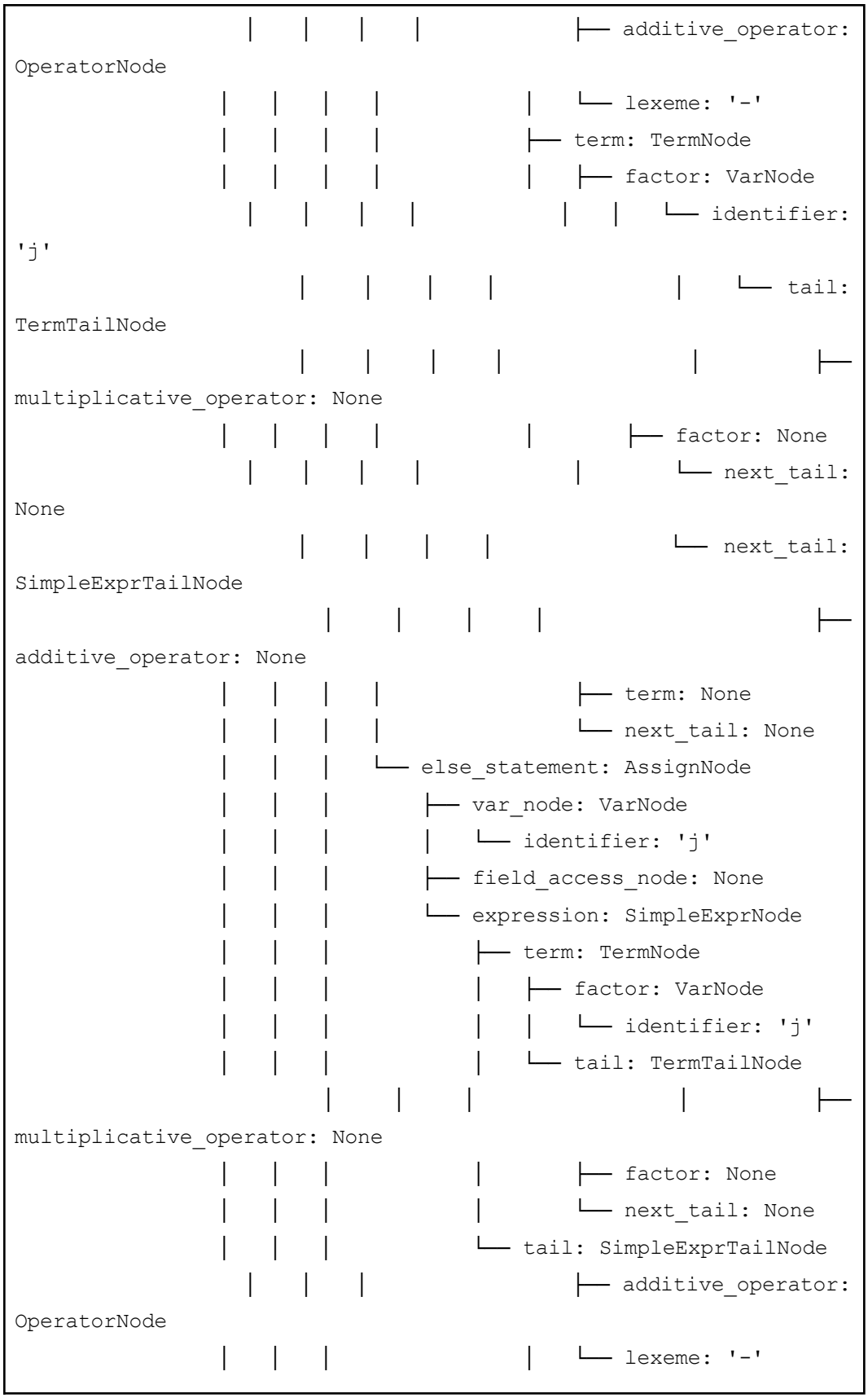

next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
next_tail: ParameterTailNode
param_group_node: None
expression_node: SimpleExprNode
term: TermNode
factor: FieldAccessNode
identifier_1: 'h2'
identifier_2: 'c'
tail: FieldAccessTailNode
identifier: 'im'
next_tail: FieldAccessTailNode
identifier: None
next_tail: None
tail: TermTailNode

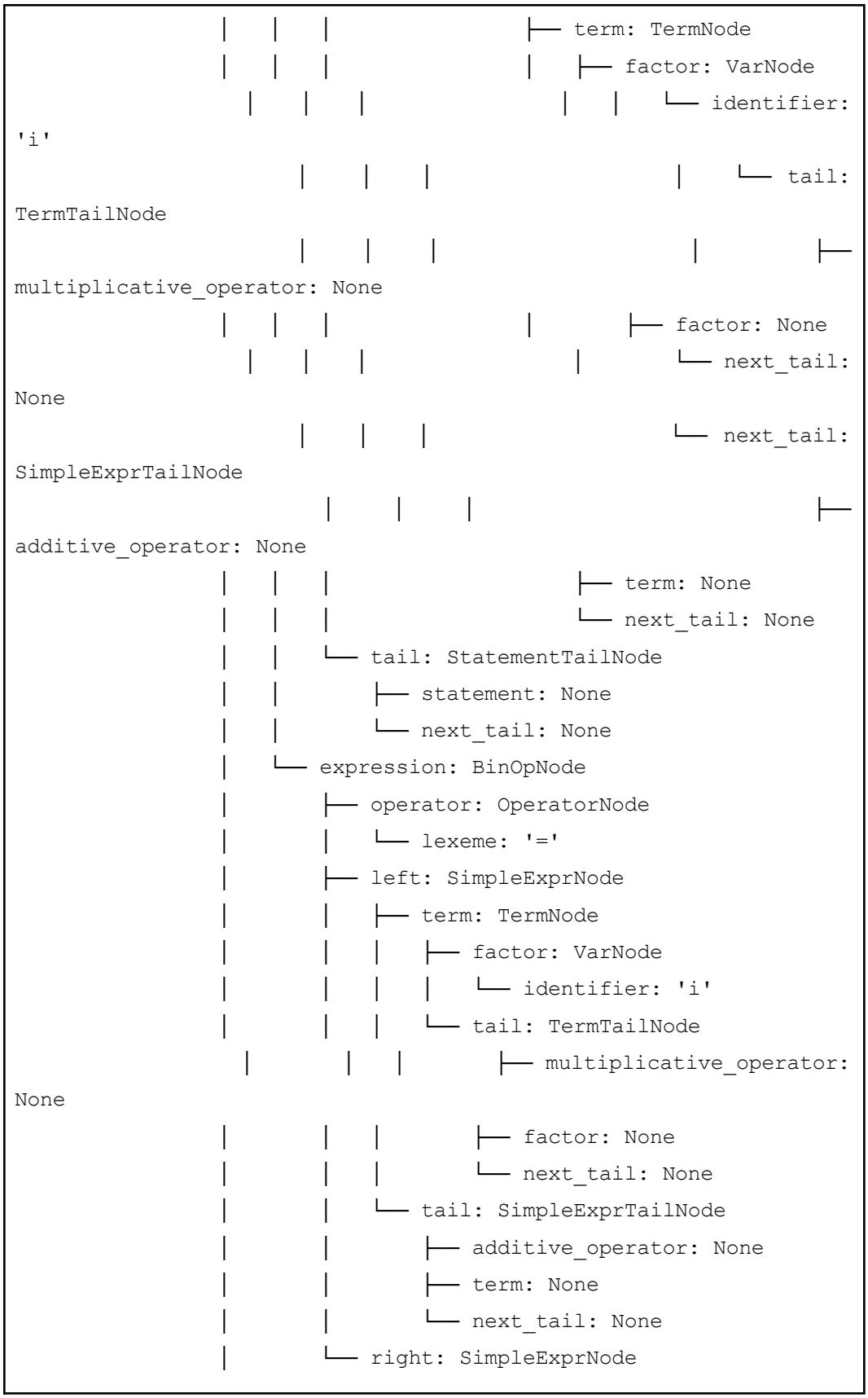

multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
next_tail: ParameterTailNode
param_group_node: None
expression_node: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'z'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None

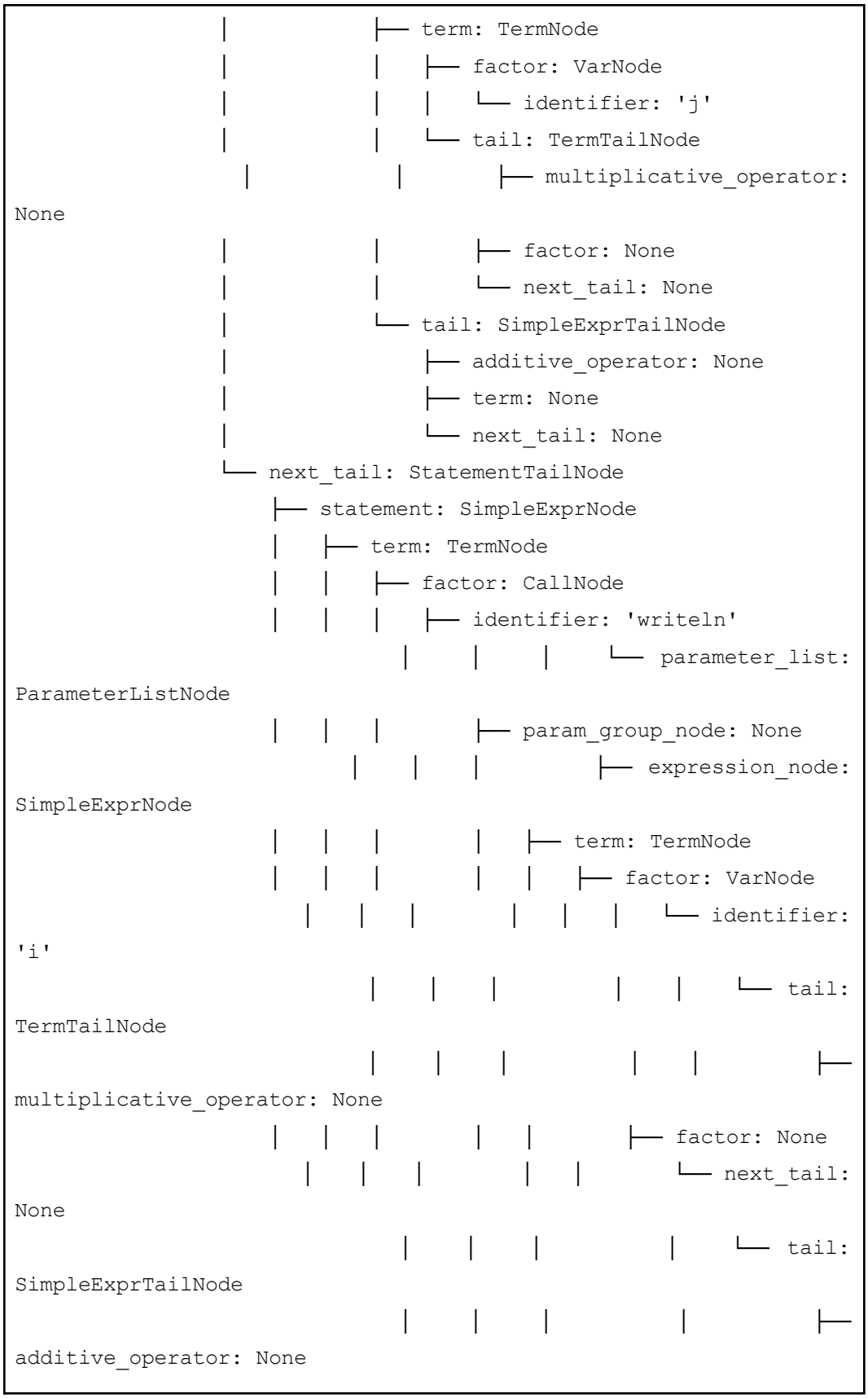

term: None
next_tail: None
next_tail: ParameterTailNode
param_group_node: None
expression_node: None
next_tail: None
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
tail: StatementTailNode
statement: None
next_tail: None
next_section: SubprogramSectionNode
subprogram_declaration: None
next_section: None
compound_statement: CompoundNode
statement_list: StatementListNode
statement: AssignNode
var_node: VarNode
identifier: 'i'
field_access_node: None
expression: SimpleExprNode
term: TermNode
factor: NumberNode
value: '85'


tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
tail: StatementTailNode
statement: AssignNode
var_node: VarNode
identifier: 'j'
field_access_node: None
expression: SimpleExprNode
term: TermNode
factor: NumberNode
value: '51'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
next_tail: StatementTailNode
statement: RepeatNode
statement_list: StatementListNode
statement: IfNode
expression: BinOpNode
operator: OperatorNode
lexeme: '>'
left: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'i'
tail: TermTailNode


multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
right: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'j'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
then_statement: AssignNode
var_node: VarNode
identifier: 'i'
field_access_node: None
expression: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'i'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode


| | | | ├── additive_operator: OperatorNode
| | | | | └── lexeme: '-'
| | | | ├── term: TermNode
| | | | | ├── factor: VarNode
| | | | | | └── identifier: 'j'
| | | | | └── tail: TermTailNode
| | | | | ├── multiplicative_operator: None
| | | | | ├── factor: None
| | | | | └── next_tail: None
| | | | └── next_tail: SimpleExprTailNode
| | | | ├── additive_operator: None
| | | | ├── term: None
| | | | └── next_tail: None
| | | └── else_statement: AssignNode
| | | ├── var_node: VarNode
| | | | └── identifier: 'j'
| | | ├── field_access_node: None
| | | └── expression: SimpleExprNode
| | | ├── term: TermNode
| | | | ├── factor: VarNode
| | | | | └── identifier: 'j'
| | | | └── tail: TermTailNode
| | | | ├── multiplicative_operator: None
| | | | ├── factor: None
| | | | └── next_tail: None
| | | └── tail: SimpleExprTailNode
| | | ├── additive_operator: OperatorNode
| | | | └── lexeme: '-'


| | | ├── term: TermNode
| | | | ├── factor: VarNode
| | | | | └── identifier: 'i'
| | | | └── tail: TermTailNode
| | | | ├── multiplicative_operator: None
| | | | ├── factor: None
| | | | └── next_tail: None
| | | └── next_tail: SimpleExprTailNode
| | | ├── additive_operator: None
| | | ├── term: None
| | | └── next_tail: None
| | └── tail: StatementTailNode
| | ├── statement: None
| | └── next_tail: None
| └── expression: BinOpNode
| ├── operator: OperatorNode
| | └── lexeme: '='
| ├── left: SimpleExprNode
| | ├── term: TermNode
| | | ├── factor: VarNode
| | | | └── identifier: 'i'
| | | └── tail: TermTailNode
| | | ├── multiplicative_operator: None
| | | ├── factor: None
| | | └── next_tail: None
| | └── tail: SimpleExprTailNode
| | ├── additive_operator: None
| | ├── term: None
| | └── next_tail: None
| └── right: SimpleExprNode


term: TermNode
factor: VarNode
identifier: 'j'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None
term: None
next_tail: None
next_tail: StatementTailNode
statement: SimpleExprNode
term: TermNode
factor: CallNode
identifier: 'writeln'
parameter_list: ParameterListNode
param_group_node: None
expression_node: SimpleExprNode
term: TermNode
factor: VarNode
identifier: 'i'
tail: TermTailNode
multiplicative_operator: None
factor: None
next_tail: None
tail: SimpleExprTailNode
additive_operator: None

```
                    |   |   |       |       ├── term: None
                    |   |   |       |       └── next_tail: None
                                |   |   |           └── next_tail:
ParameterTailNode
                      |   |   |           ├── param_group_node:
None
                    |   |   |           ├── expression_node: None
                    |   |   |           └── next_tail: None
                    |   |   └── tail: TermTailNode
                    |   |       ├── multiplicative_operator: None
                    |   |       ├── factor: None
                    |   |       └── next_tail: None
                    |   └── tail: SimpleExprTailNode
                    |       ├── additive_operator: None
                    |       ├── term: None
                    |       └── next_tail: None
                    └── next_tail: StatementTailNode
                        ├── statement: None
                        └── next_tail: None

================================================
Tahap: Semantic Analysis (Symbol Tables)
================================================
Semantic Error: Condition of WHILE must be boolean
```

Pembahasan: Hasil uji terakhir menunjukkan bahwa Semantic Analyzer memvalidasi tipe ekspresi kondisi pada loop `selama` (while). Ekspresi `x + y` menghasilkan nilai `real`, bukan `boolean`. Analyzer dengan tepat melaporkan "Condition of WHILE must be boolean", menegaskan aturan bahwa kondisi kontrol alur harus bernilai kebenaran (true/false).

Bab IV

Penutup

4.1 Kesimpulan

Berdasarkan perancangan, implementasi, dan pengujian yang telah dilakukan, dapat disimpulkan bahwa implementasi Semantic Analyzer berhasil diintegrasikan dengan Lexer dan Parser sehingga melengkapi keseluruhan fungsionalitasnya compiler Pascal-S. Penggunaan pola Visitor pada AST terbukti efektif dalam melakukan analisis semantik. Selain itu, struktur data Symbol Table yang terdiri dari tab, atab, dan btab mampu mengelola manajemen scope, termasuk nested scope, serta memeriksa tipe data yang kompleks seperti array dan record. Sistem Type Checking juga sudah mampu menjalankan aturan *strong typing* dengan baik, mencegah operasi ilegal seperti assignment antartipe yang tidak kompatibel maupun operasi aritmatika pada tipe yang bukan numerik. Secara keseluruhan compiler sudah mampu mendeteksi berbagai kesalahan semantik serta memberikan pesan error yang jelas kepada *user*.

4.2 Saran

Meskipun terlihat kompleks, masih banyak bug yang dialami. Seharusnya desain algoritma visit function lebih diperhatikan agar tidak terlalu banyak *refactor* yang dilakukan berdasarkan hasil test case. Selain itu, pendefinisian grammar sering berubah seiring berjalannya waktu selama pengembangan sehingga banyak menghabiskan waktu. Hal ini terjadi banyak kasus yang belum terbayangkan saat mendesain definisi grammar, sehingga ketika terbayangkan kasus baru tersebut saat sudah membuat pohon AST bahkan semantik analisisnya, perombakan ulang diperlukan. Kode yang mendefinisikan grammar dalam *source code* sudah dibuat readable, akan tetapi belum untuk AST nodes, AST transformer, dan AST analyzer. Sebaiknya pendefinisian kelas lebih terstruktur dibanding implementasi sekarang agar pengembang dapat mengerti kode secara intuitif.

Lampiran

Link Github Repository : https://github.com/fliegenhaan/JJK-Tubes-IF2224.git

Link Diagram : Diagram-3-JJK.png

Pembagian Tugas :

No	NIM	Tugas	Persentase kontribusi
1	13523124	Implementasi kode program; Perancangan DFA rules; Dokumentasi laporan; Perancangan diagram	25%
2	13523136	Implementasi kode program; Pembuatan Test Case unik; Perancangan DFA rules; Dokumentasi laporan	25%
3	13523142	Perancangan DFA rules; Dokumentasi laporan; Perancangan diagram	25%
4	13523155	Dokumentasi laporan	25%

Daftar Pustaka

Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung. Spesifikasi Tugas Besar IF2224 - Formal Languages and Automata Theory, Milestone 1 - Lexical Analysis. Diakses pada 17 Oktober 2025, dari https://docs.google.com/document/d/1w0GmHW5L0gKZQWbgmtJPFmOzlpSWBknNPdugucn4eII/edit?usp=sharing

Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung. Spesifikasi Tugas Besar IF2224 - Formal Languages and Automata Theory, Milestone 2 - Syntax Analysis. Diakses pada 12 November 2025, dari https://docs.google.com/document/d/1G_pC2dltQ5Q-iQ3xOpmLIl3XWLDcLfodN8jdiaJBA0Q/edit?usp=sharing

Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung. Spesifikasi Tugas Besar IF2224 - Formal Languages and Automata Theory, Milestone 3 - Semantic Analysis. Diakses pada 30 November 2025, dari https://docs.google.com/document/d/16c2v6VRhfcsrZYTPumyqmnY8RWbI95U_S8YLRed0MVs/edit?usp=sharing

GeeksforGeeks. "Introduction to Lexical Analysis." GeeksforGeeks. Diakses pada 14 Oktober 2025, dari https://www.geeksforgeeks.org/introduction-of-lexical-analysis/

GeeksforGeeks. "Deterministic Finite Automaton (DFA)." GeeksforGeeks. Diakses pada 15 Oktober 2025, dari https://www.geeksforgeeks.org/introduction-of-finite-automata/

TutorialsPoint. "Compiler Design - Lexical Analysis." TutorialsPoint. Diakses pada 15 Oktober 2025, dari https://www.tutorialspoint.com/compiler_design/compiler_design_lexical_analysis.htm

Free Pascal Documentation. "Pascal Language Reference." Free Pascal Wiki. Diakses pada 15 Oktober 2025, dari https://www.freepascal.org/docs.html

Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. Compilers: Principles, Techniques, and Tools (2nd Edition). Pearson Education, 2006.

GeeksforGeeks. "Introduction to Syntax Analysis in Compiler Design." Diakses pada 4 November 2025, dari https://www.geeksforgeeks.org/introduction-to-syntax-analysis-in-compiler-design/

GeeksforGeeks. "Parsing | Set 1 (Introduction, Ambiguity and Parsers)." GeeksforGeeks. Diakses pada 6 November 2025, dari https://www.geeksforgeeks.org/parsing-set-1-introduction-ambiguity-and-parsers/

GeeksforGeeks. "Recursive Descent Parser." Diakses pada 7 November 2025, dari https://www.geeksforgeeks.org/recursive-descent-parser/

"Writing a Recursive Descent Parser." Medium. Diakses pada 7 November 2025, dari https://medium.com/@chetaniam/writing-a-recursive-descent-parser-d3c5d6626b07

Tutorialspoint. "Compiler Design - Top-Down Parser." Diakses pada 7 November 2025, dari https://www.tutorialspoint.com/compiler_design/compiler_design_top_down_parser.htm

"Building a Parser from Scratch." Medium. Diakses pada 5 November 2025, dari https://medium.com/@chetaniam/building-a-parser-from-scratch-8f9b8e8e0a5d

GeeksforGeeks. "Semantic Analysis in Compiler Design." GeeksforGeeks. Diakses pada 28 November 2025, dari https://www.geeksforgeeks.org/compiler-design/semantic-analysis-in-compiler-design/

Tutorials Point. "Compiler Design - Semantic Analysis." Tutorials Point. Diakses pada 28 November 2025, dari https://www.tutorialspoint.com/compiler_design/compiler_design_semantic_analysis.htm

Geurts, Pierre. "Semantic Analysis (Slides for Compiler Course 2015–2016)." Montefiore Institute, University of Liège. Diakses pada 30 November 2025, dari https://people.montefiore.uliege.be/geurts/Cours/compil/2015/04-semantic-2015-2016.pdf