ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN TỐT NGHIỆP

NGÀNH: CÔNG NGHỆ THÔNG TIN

CHUYÊN NGÀNH: CÔNG NGHỆ PHẦN MỀM

ĐỀ TÀI:

NGÔN NGỮ LẬP TRÌNH TIẾNG VIỆT
VieLang

Người hướng dẫn:**PGS TS. NGUYỄN THANH BÌNH**
Sinh viên thực hiện: **HUỲNH ĐINH HOÀNG VIÊN**
Số thẻ sinh viên: **102200160**
Lớp**: 20TCLC_DT3**

Đà Nẵng, 06/2024

NHẬN XÉT CỦA NGƯỜI HƯỚNG DẪN

....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....

Đà Nẵng, ngày   tháng 7 năm 2023

Người hướng dẫn

PGS. TS. NGUYỄN THANH BÌNH

NHẬN XÉT CỦA NGƯỜI PHẢN BIỆN

…………………………………………………………………………………………
……………………………………………………………………………………….
……………………………………………………………………………………….
……………………………………………………………………………………….
……………………………………………………………………………………….
……………………………………………………………………………………….
……………………………………………………………………………………….
……………………………………………………………………………………….
……………………………………………………………………………………….
……………………………………………………………………………………….
……………………………………………………………………………………….
……………………………………………………………………………………….
……………………………………………………………………………………….
……………………………………………………………………………………….
……………………………………………………………………………………….
……………………………………………………………………………………….
……………………………………………………………………………………….
…………………………………………………………………………………………
……………………………………………………………………………………….
……………………………………………………………………………………….
……………………………………………………………………………………….
……………………………………………………………………………………….
……………………………………………………………………………………….
……………………………………………………………………………………….
…………………………………………………………………………………

Đà Nẵng, ngày tháng 07 năm 2023

Người phản biện

HUỲNH ĐINH HOÀNG VIÊN

TÓM TẮT

Tên đề tài: Ngôn ngữ lập trình tiếng Việt - VieLang

Sinh viên thực hiện: Huỳnh Đinh Hoàng Viên

Số thẻ SV: 102200160

Lớp: 20TCLC_DT3

Báo cáo này trình bày về ngôn ngữ lập trình tiếng Việt - VieLang. Hệ thống sử dụng cây cú pháp trừu tượng (AST) để phân tích và biểu diễn cấu trúc cú pháp của đoạn mã đầu vào; sau đó xây dựng cấu trúc dữ liệu tương tự sơ đồ khối sử dụng các thông tin được trích xuất từ AST; xác định các thành phần của các cú pháp; cuối cùng, sau đó sử dụng bộ thực thi của môi trường NodeJS. Hệ thống đã phát triển thành công các sản phẩm bao gồm một ứng dụng web có thể lập trình trực tiếp từ trình duyệt. Mục đích chính của sản phẩm là mang lập trình đến gần với các bạn học sinh cấp 1 cấp 2 bằng ngôn ngữ tiếng Việt, bên cạnh đem lại kiến thức trong quá trình nghiên cứu.

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA CÔNG NGHỆ THÔNG TIN

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

Họ tên sinh viên: Huỳnh Đinh Hoàng Viên

Số thẻ sinh viên: 102200160

Lớp: 20TCLC_DT3

Khoa: Công nghệ thông tin

1. *Tên đề tài đồ án: Ngôn ngữ lập trình tiếng Việt - VieLang*
2. *Đề tài thuộc diện: ☐ Có ký kết thỏa thuận sở hữu trí tuệ đối với kết quả thực hiện*
3. *Nội dung các phần thuyết minh và tính toán:*
   - Chương 1: Giới thiệu tổng quan về đề tài
   - Chương 2: Cơ sở lý thuyết của các công nghệ sử dụng
   - Chương 3: Phân tích và thiết kế hệ thống tổng quan
   - Chương 4: Phát triển các sản phẩm ứng dụng
   - Chương 5: Thử nghiệm và đánh giá hiệu suất
   - Chương 6: Kết quả đạt được, hạn chế và hướng phát triển
4. *Họ tên người hướng dẫn:* PGS TS. Nguyễn Thanh Bình
5. *Ngày giao nhiệm vụ đồ án: 4/04/2024*
6. *Ngày hoàn thành đồ án: 03/07/2023*

Đà Nẵng, ngày 03 tháng 07 năm 2023

Trưởng Bộ môn Công nghệ phần mềm **Người hướng dẫn**

LỜI NÓI ĐẦU

Trước tiên, em xin gửi lời cảm ơn chân thành nhất đến thầy PGS TS. Nguyễn Thanh Bình vì sự hướng dẫn, đồng hành và hỗ trợ em trong suốt quá trình thực hiện đồ án. Em xin cảm ơn sự chỉ dạy và định hướng từ thầy, giúp em hiểu rõ hơn về phương pháp nghiên cứu, công nghệ sử dụng và quy trình triển khai dự án. Nhờ có sự hỗ trợ nhiệt tình của thầy, em đã có cơ hội tiếp cận với một đề tài thú vị và mang tính ứng dụng cao.

Em cũng xin gửi lời cảm ơn đến các giảng viên trong khoa Công nghệ thông tin và trường Đại học Bách Khoa đã tạo điều kiện và cung cấp môi trường học tập chất lượng để em có thể hoàn thành dự án này. Sự đồng hành, động viên và kiến thức mà các thầy cô chia sẻ là nguồn động lực quan trọng để em không ngừng nỗ lực vươn đến mục tiêu đề ra.

Hy vọng rằng kết quả của đồ án này sẽ đáp ứng được mong đợi của thầy cô và mang lại đóng góp nhỏ cho cộng đồng lập trình, giúp em phát triển kỹ năng, mở rộng kiến thức và trở thành sinh viên có khả năng đáp ứng nhu cầu thực tế của ngành Công nghệ thông tin.

Xin chân thành cảm ơn và kính chúc thầy cô và toàn thể giảng viên trong khoa, trường mạnh khỏe và thành công trong công việc giảng dạy và nghiên cứu.

Sinh viên thực hiện

Huỳnh Đinh Hoàng Viên

CAM ĐOAN

Nội dung trong luận văn này là do em thực hiện dưới sự hướng dẫn trực tiếp của Thầy PGS TS Nguyễn Thanh Bình.

Tất cả các thông tin và dữ liệu được trình bày trong đồ án tốt nghiệp là đáng tin cậy, em cam đoan đã ghi rõ và chỉ ra nguồn gốc của mọi tài liệu, thông tin, và ý kiến được sử dụng trong đồ án tốt nghiệp.

Em cam kết tuân thủ tất cả các cam đoan và chấp hành các quy tắc đạo đức trong quá trình thực hiện đồ án tốt nghiệp. Nếu có những vi phạm quy chế đào tạo, em xin chịu trách nhiệm.

Sinh viên thực hiện

Huỳnh Đinh Hoàng Viên

MỤC LỤC

DANH SÁCH CÁC HÌNH VẼ

DANH SÁCH CÁC BÀNG

DANH SÁCH CÁC CHỮ VIẾT TẮT

AST	Abstract Syntax Tree
VS Code	Visual Studio Code
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JSON	JavaScript Object Notation
TDD	Test Driven Development

MỞ ĐẦU

Trong bối cảnh chuyển đổi số, công nghệ dần phổ biến với tất các mọi người. Hiện nay việc dạy và học lập trình đang dần phổ biến ở khối trung học cơ sở, kể cả tiểu học. Tuy nhiên, hiện tai đa số các bạn học sinh tiểu học và cơ sở vẫn còn khá yếu về Tiếng Anh, đặc biệt là các bạn ở vùng sâu vùng xa không có điều kiện đi học thêm, tiếp xúc với tiếng anh nhiều, điều này cũng cản trở một phần tới việc học lập trình. Việc tạo ra một ngôn ngữ lập trình với cú pháp bằng Tiếng Việt, điều này giúp các em học sinh dễ dàng tiếp cận với lập trình hơn, tăng hiệu quả dạy và học. Ngôn ngữ lập trình Tiếng Việt - VieLang được xây dựng nhằm cung cấp cho các em học sinh một mã nguồn bằng ngôn ngữ tự nhiên Tiếng Việt, dễ dàng tiếp cận.

Dự án đang tập trung chủ yếu hỗ trợ ngôn ngữ lập trình Javascript. Mặc dù hệ thống có tiềm năng sử dụng các ngôn ngữ bậc thấp như Rust hoặc C++ để tăng hiệu suất, nhưng trong phạm vi đồ án tốt nghiệp, em tập trung vào việc triển khai và kiểm thử với Javascript. Phương pháp nghiên cứu bao gồm việc nghiên cứu các công nghệ liên quan đến phân tích cú pháp và biên dịch mã. Em sẽ tiến hành phân tích, thiết kế và triển khai hệ thống, sau đó kiểm thử và đánh giá hiệu quả của hệ thống thông qua các ví dụ và thực nghiệm.

Hệ thống sử dụng các kỹ thuật phân tích cú pháp và chuyển đổi cú pháp thành định dạng để Javascript đọc và biên dịch được. Đầu tiên, phân tích cú pháp để hiểu cấu trúc và các thành phần của đoạn mã, tạo ra cây cú pháp trừu tượng. Tiếp theo, hệ thống sẽ trích xuất thông tin cần thiết từ cây cú pháp trừu tượng rồi đưa đến Javascript thông dịch và thực thi. Dự án bao gồm hai sản phẩm chính: trang web mã sandbox để mã và thực thi ngay trên website, mã nguồn mở VieLang để các nhà phát triển khác có thể tích hợp vào hệ thống riêng của mình. Các sản phẩm này sẽ giúp học sinh cấp tiểu học và trung học cơ sở tiếp cận sớm với lập trình dễ dàng hơn nhờ ngôn ngữ tự nhiên là Tiếng Việt, bên cạnh đó còn giúp lập trình viên có thể phát triển thêm hệ thống riêng.

CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI

1.1. Giới thiệu đề tài

Đề tài "Ngôn ngữ lập trình Tiếng Việt - VieLang" là một ngôn ngữ lập trình giúp các bạn học sinh cấp tiểu học và trung học cơ sở dễ dàng tiếp cận với lập trình hơn nhờ sử dụng ngôn ngữ tự nhiên Tiếng Việt, thay vì các cú pháp tiếng anh khó nhớ. Đây là một ngôn ngữ hữu ích trong quá trình giảng dạy và học lập trình.

Đầu vào bằng ngôn ngữ Tiếng Việt sẽ được chuyển đổi thành cây cú pháp trừu tượng (AST). AST giúp xác định các thành phần của đoạn mã, như là tên hàm, tên biến, biểu thức, câu lệnh,.. từ đó giúp Javascript hiểu, thông dịch và thực thi.

Trong đề tài này, em sẽ trình bày về cơ sở lý thuyết, phân tích và thiết kế hệ thống, cách phân tích cú pháp, chuyển đổi từ đoạn mã tiếng Việt sang cây cú pháp trừu tượng để máy tính hiểu, thông dịch và thực thi. Với những tầm quan trọng và lợi ích mà ngôn ngữ mang lại, đề tài này có ý nghĩa quan trọng trong việc hỗ trợ học sinh tiếp cận sớm với lập trình một cách dễ dàng. Quá trình giảng dạy dễ dàng hơn.

1.2. Mục đích thực hiện

Mục đích chính của đề tài là phát triển một công cụ hỗ trợ học sinh tiếp cận với lập trình cũng như quá trình giảng dạy của giáo viên.

Hệ thống có tầm quan trọng đáng kể trong việc hiểu, phân tích và giảng dạy ngôn ngữ lập trình, cũng như trong quá trình phân tích và thiết kế phần mềm. Nó cung cấp cho người Việt một ngôn ngữ lập trình thuần Việt, dễ dàng tiếp xúc và học tập.

Tăng cường hiểu biết về lập trình từ sớm: Một trong những mục tiêu chính của việc phát triển ngôn ngữ lập trình tiếng Việt là để giúp trẻ em tiểu học hiểu và tiếp cận với lập trình một cách dễ dàng hơn. Bằng cách sử dụng ngôn ngữ gần gũi và dễ hiểu, chúng ta có thể giúp trẻ em phát triển tư duy logic và kỹ năng giải quyết vấn đề từ khi còn nhỏ.

Tạo điều kiện thuận lợi cho việc học tập: Việc sử dụng ngôn ngữ lập trình tiếng Việt sẽ giúp loại bỏ rào cản ngôn ngữ cho các em học sinh tiểu học. Thay vì phải đối mặt với khó khăn trong việc hiểu các thuật ngữ và cú pháp lập trình bằng tiếng Anh, các em có thể dễ dàng tiếp cận và học tập thông qua ngôn ngữ mẹ đẻ của mình.

Giảng viên có thể sử dụng ngôn ngữ lập trình Tiếng Việt để giảng dạy cho học sinh với một cách truyền đạt dễ dàng hơn.

1.3. Phạm vi nghiên cứu

Đề tài tập trung vào việc phát triển ngôn ngữ lập trình bằng tiếng mẹ đẻ để các bạn học sinh dễ dàng học tập hơn, vì đối tượng là học sinh nên những bài toán áp dụng bằng ngôn ngữ lập trình này chưa đặt nặng vấn đề hiệu suất, hiệu năng. Ngôn ngữ được xây dựng bằng Javascript. Javascript là một ngôn ngữ lập trình phổ biến và được sử dụng rộng rãi trong nhiều lĩnh vực như là lập trình web, phần mềm desktop, game, … Hệ thống sẽ được thiết kế và phát triển dựa trên cú pháp và cấu trúc của ngôn ngữ tiếng việt và chuyển đổi nó sang dạng từ tố của Javascript để thông dịch và thực thi.

Đối tượng nghiên cứu bao gồm các đoạn mã bằng tiếng việt có độ phức tạp từ đơn giản đến trung bình. Đề tài tập trung vào việc xây dựng một gói thư viện phân tích cú pháp và ngữ nghĩa của các đoạn mã tiếng Việt, rồi dựa trên đó thực thi đoạn mã.

Phạm vi nghiên cứu của đề tài giới hạn trong việc phân tích và tạo ra ngôn ngữ lập trình giải quyết các bài toán cấp tiểu học và trung học cơ sở, mục đích chính để các bạn học sinh tiếp cận sớm với lập trình một cách dễ dàng từ đó rèn luyện tư duy lập trình từ sớm.

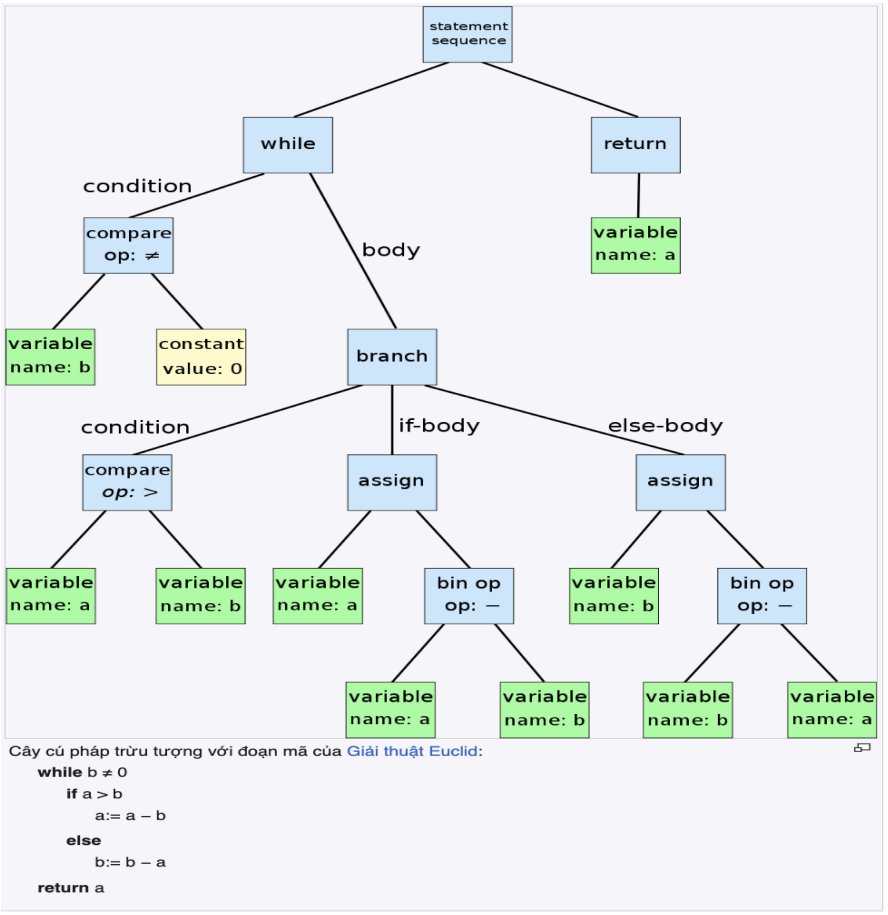
Phạm vi nghiên cứu của em tập trung nghiên cứu và phát triển ngôn ngữ lập trình giải quyết các bài toán quy mô nhỏ đến trung bình. Không bao gồm các bài toán phức tạp như đa luồng, … Vì phạm vi thời gian làm đồ án tốt nghiệp. Việc mở rộng phạm vi nghiên cứu để bao gồm nhiều hàm liên quan và xây dựng các kiến trúc phức tạp là một hướng phát triển tiềm năng trong tương lai.

CHƯƠNG 2: XÂY DỰNG TRÌNH BIÊN DỊCH VIELANG

2.1. Các khái niệm liên quan

2.1.1. Cây cú pháp trừu tượng - Abstract syntax tree (AST)

Cây cú pháp trừu tượng (AST) là một cây có giới hạn, có nhãn và có định hướng. Đây là cấu trúc cây mà các nút gốc của cây được gán nhãn bằng các toán tử và các nút của cây là các toán hạng. Tuy vậy, các lá cũng có khi là các giá trị NULL hoặc là các biến hoặc các hàng. Trong các công đoạn của chương trình dịch, cây AST này được dùng trong bộ phân tích cú pháp như là một trung gian giữa cây phân tích cú pháp (concrete syntax tree) và cấu trúc dữ liệu. Cây cú pháp trừu tượng khác với cây phân tích cú pháp là ở chỗ nó không chỉ quan tâm đến cú pháp mà còn quan tâm đến ngữ nghĩa của chương trình.

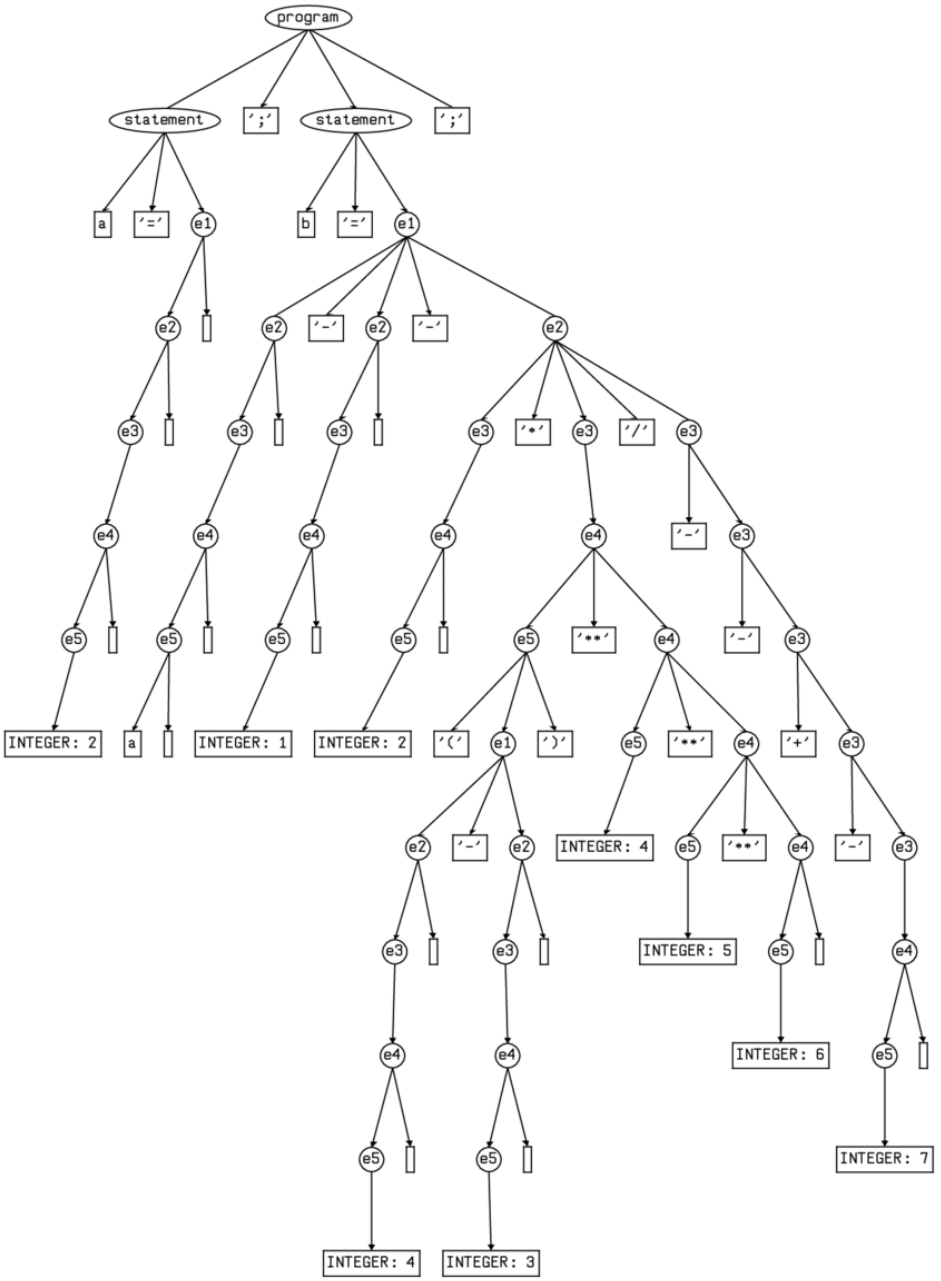


Hình 2.1. Cây cấu pháp trừu tượng

2.1.2. Kỹ thuật phân tích cú pháp bằng truy xuất đệ quy (Recursive Descent Parsing)

Truy xuất đệ quy là cách đơn giản nhất để thực hiện trình phân tích cú pháp. Tuy nhiên, không vì đơn giản mà nó yếu, ngược lại, phương pháp này nhanh chóng, mạnh mẽ và có thể xử lý các lỗi phức tạp. Trên thực tế, nhiều công cụ mạnh như GCC (trình biên dịch C/C++), V8 (máy ảo JavaScript trong Chrome), Roslyn (trình biên dịch C# viết bằng C#) và nhiều ngôn ngữ khác cũng sử dụng truy xuất đệ quy. Điều này chứng minh rằng truy xuất đệ quy là một kỹ thuật hiệu quả.

Kỹ thuật phân tích cú pháp truy xuất đệ quy còn được gọi là trình phân tích cú pháp từ trên xuống, vì chúng xây dựng cây phân tích từ trên xuống.



Hình 2.2. Cây phân tích cú pháp bằng truy xuất đệ quy

2.1.3. Cú pháp của ngôn ngữ lập trình

Cú pháp (syntax) là các quy tắc và nguyên tắc về viết mã và đặt cú pháp trong một ngôn ngữ lập trình cụ thể. Mỗi ngôn ngữ lập trình có một cú pháp riêng, định nghĩa các từ khóa, cú pháp, quy tắc về viết chú thích và biểu diễn dữ liệu. Hiểu và thực hiện đúng cú pháp của ngôn ngữ là rất quan trọng để viết mã nguồn chính xác và có thể chạy được.

2.1.4. Phân tích cú pháp

Phân tích cú pháp (syntax analysis) là quá trình phân tích và kiểm tra tính hợp lệ của một đoạn mã nguồn theo quy tắc cú pháp của ngôn ngữ lập trình. Quá trình này diễn ra trước khi mã được thực thi và thường là bước đầu tiên trong quá trình biên dịch hoặc thông dịch mã nguồn.

Trong quá trình phân tích cú pháp, đoạn mã nguồn sẽ được phân tách thành các thành phần cú pháp nhỏ hơn (tokens) dựa trên quy tắc ngôn ngữ lập trình. Các thành phần này bao gồm từ khóa (keywords), biểu thức (expressions), toán tử (operators), các hằng số (constants), và các dấu ngoặc, dấu câu, dấu phẩy, v.v.

Quá trình phân tích cú pháp sử dụng các quy tắc ngữ pháp (grammar rules) để kiểm tra xem các thành phần cú pháp có tuân theo cú pháp ngôn ngữ không. Một ngữ pháp ngôn ngữ lập trình định nghĩa cách các thành phần cú pháp phối hợp với nhau để tạo thành câu lệnh và khối mã.

Kết quả của quá trình phân tích cú pháp thường là một cấu trúc dữ liệu cây gọi là cây cú pháp (parse tree) hoặc cây cú pháp trừu tượng (abstract syntax tree). Cây cú pháp biểu diễn cấu trúc cú pháp của đoạn mã và cho phép các bước xử lý tiếp theo như phân tích ngữ nghĩa và tạo mã máy.

2.2. Các công nghệ được sử dụng

2.2.1. Ngôn ngữ lập trình JavaScript

JavaScript là một ngôn ngữ lập trình phía máy khách (client-side) được sử dụng chủ yếu trong việc phát triển ứng dụng web. JavaScript là một ngôn ngữ dựa trên sự kiện (event-driven) và có khả năng tương tác với người dùng thông qua các sự kiện, thao tác trên trang web và giao diện người dùng. JavaScript là một ngôn ngữ thông dịch, có thể thực thi trực tiếp trên trình duyệt mà không cần biên dịch trước.

Hình 2.3. Logo ngôn ngữ lập trình JavaScript

JavaScript có cú pháp đơn giản và rõ ràng, tương đồng với nhiều ngôn ngữ lập trình khác, giúp người mới học dễ dàng tiếp cận và phát triển ứng dụng nhanh chóng. JavaScript có thể chạy trên nhiều trình duyệt web khác nhau và được hỗ trợ trên nhiều hệ điều hành khác nhau. JavaScript có thể tương tác trực tiếp với các phần tử HTML và CSS trên trang web, cho phép thay đổi nội dung, hiệu ứng và tương tác trực tiếp với người dùng.

JavaScript là một công cụ quan trọng trong việc phát triển ứng dụng web tương tác và tạo ra trải nghiệm người dùng tốt hơn trên trình duyệt. Với tính linh hoạt, khả năng tương tác và tích hợp tốt với HTML và CSS, JavaScript đã trở thành một trong những ngôn ngữ lập trình phổ biến nhất trong lĩnh vực phát triển web.

2.2.3. Môi trường Node.js

Node.js là một nền tảng phát triển ứng dụng web mã nguồn mở được xây dựng trên JavaScript runtime của Chrome V8 Engine.

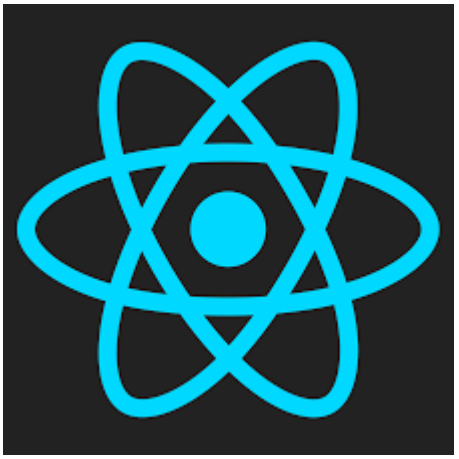
Hình 2.4. Logo môi trường Node.js

Với mô hình không chặn I/O và lập trình bất đồng bộ, Node.js tối ưu hóa hiệu suất và khả năng đáp ứng của ứng dụng. Sự linh hoạt và tích hợp tốt với các công nghệ mới như ES6/ES7, TypeScript, và GraphQL làm cho Node.js trở thành lựa chọn phổ biến cho việc phát triển ứng dụng web hiện đại.

Nhờ vào cộng đồng lập trình viên lớn mạnh và NPM (Node Package Manager), việc mở rộng chức năng của ứng dụng trở nên dễ dàng. Node.js thích hợp cho các ứng dụng web, API, máy chủ WebSocket, ứng dụng IoT, và nhiều hơn nữa.

2.2.4. Framework React

ReactJS là một thư viện JavaScript mã nguồn mở, được phát triển bởi Facebook, chủ yếu được sử dụng để xây dựng giao diện người dùng cho các ứng dụng web hiện đại. Điểm mạnh của ReactJS là cách nó quản lý và tái sử dụng các thành phần giao diện, giúp người phát triển xây dựng ứng dụng web phức tạp một cách dễ dàng và hiệu quả.



Hình 2.5. Logo framework ReactJS.

Với ReactJS, bạn có thể tạo ra các thành phần giao diện động, linh hoạt và dễ dàng tái sử dụng, giúp tăng tính tương tác và trải nghiệm người dùng. Nó cung cấp một cách tiếp cận dựa trên các "component" (thành phần), mỗi thành phần đại diện cho một phần nhỏ của giao diện người dùng, giúp tạo ra mã nguồn dễ đọc và bảo trì.

Bằng cách sử dụng ReactJS cùng với các công nghệ khác như Redux hoặc Context API, người phát triển có thể quản lý trạng thái ứng dụng một cách hiệu quả, giúp tạo ra các ứng dụng web có hiệu suất cao và dễ bảo trì.

Với cộng đồng lớn và sự phát triển liên tục, ReactJS đã trở thành một trong những công cụ phổ biến nhất cho việc phát triển ứng dụng web, được ứng dụng rộng rãi trong các dự án từ nhỏ đến lớn.

2.2.5. Build Tool - Vite

Vite là một công cụ phát triển web tốc độ cao, được tạo ra bởi Evan You, nhà đồng sáng lập Vue.js. Được thiết kế để thay thế quy trình build truyền thống của các dự án JavaScript, Vite nhằm mục tiêu cải thiện thời gian phản hồi và hiệu suất của quá trình phát triển.

Điểm nổi bật của Vite là việc sử dụng ES Module (ESM) như một cơ sở, cho phép tải các module theo yêu cầu (on-demand) trong quá trình phát triển, thay vì phải build toàn bộ ứng dụng từ đầu. Điều này mang lại trải nghiệm phát triển mượt mà và nhanh chóng, với thời gian khởi động ứng dụng và tái tạo trang cực kỳ nhanh.

.

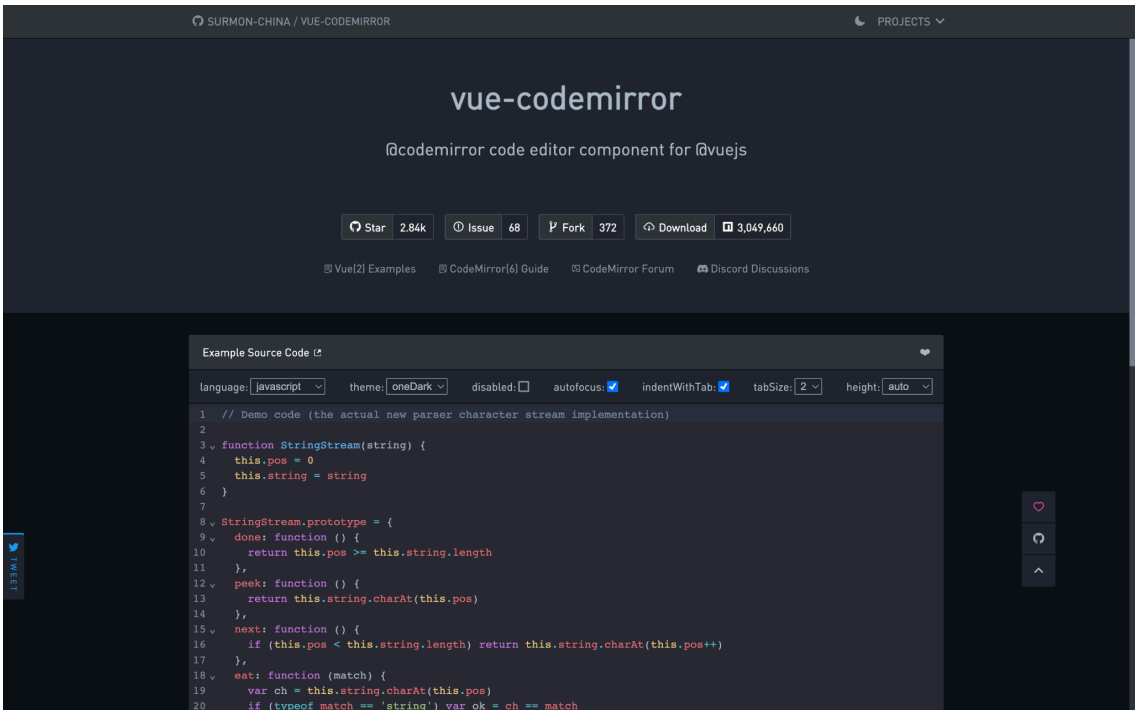
Hình 2.6. Logo Vite.

Với Vite, chúng ta có thể sử dụng các framework như Vue.js, React, hoặc Svelte, cũng như các công nghệ như TypeScript hay CSS Pre-processors mà không cần cấu hình phức tạp. Bên cạnh đó, Vite còn hỗ trợ hot module replacement (HMR), giúp cập nhật nhanh chóng các thay đổi trong mã nguồn mà không cần làm mới trang.

Với sự tích hợp sẵn các tính năng hiện đại và hiệu suất ấn tượng, Vite đã thu hút sự quan tâm của cộng đồng phát triển web và trở thành một lựa chọn phổ biến cho các dự án web hiện đại.

2.2.6. Thư viện @babel/generator

Thư viện **@babel/generator** là một phần của hệ thống Babel - một công cụ biên dịch mã nguồn JavaScript. @babel/generator có nhiệm vụ chuyển đổi cây cú pháp (AST - Abstract Syntax Tree) của mã nguồn JavaScript từ định dạng AST này sang chuỗi ký tự JavaScript, thường được gọi là "generate mã".



Hình 2.7. Thư viện @babel/generator.

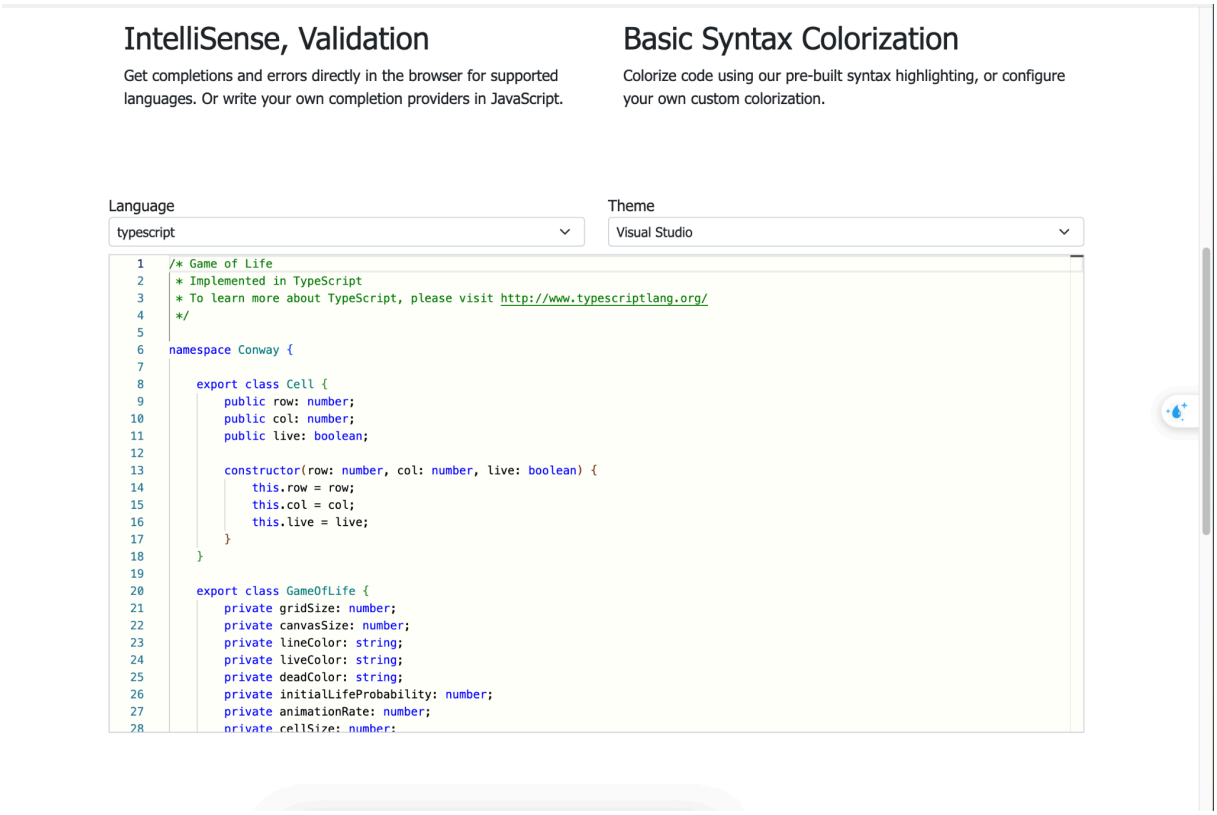
Với @babel/generator, bạn có thể truy cập vào cây cú pháp của mã nguồn JavaScript và tạo ra mã nguồn mới dựa trên cây cú pháp này. Điều này rất hữu ích khi bạn muốn thực hiện các biến đổi trên mã nguồn JavaScript, như việc tạo ra phiên bản mã nguồn tối ưu hóa hoặc tạo ra mã nguồn tương thích với các phiên bản JavaScript cũ hơn.

Ví dụ, bạn có thể sử dụng @babel/generator để thực hiện việc biến đổi mã nguồn từ JavaScript mới (ví dụ: ES6+) sang JavaScript cũ hơn (ví dụ: ES5) để hỗ trợ các trình duyệt cũ hơn hoặc các môi trường chạy không hỗ trợ các tính năng mới.

Tóm lại, @babel/generator là một công cụ quan trọng trong hệ sinh thái Babel, giúp biên dịch mã nguồn JavaScript từ dạng AST sang chuỗi ký tự JavaScript và cung cấp các cơ chế để thực hiện các biến đổi mã nguồn.

2.2.7. Thư viện monaco editor

Monaco Editor là một thư viện mã nguồn mở của Microsoft, cung cấp một trình soạn thảo mã nguồn dựa trên trình duyệt web. Được xây dựng trên nền tảng của Visual Studio Code, Monaco Editor mang lại trải nghiệm soạn thảo mã nguồn mạnh mẽ và linh hoạt, với các tính năng tiên tiến và hiệu suất tốt.



Hình 2.8. Thư viện monaco editor.

Một số tính năng nổi bật của Monaco Editor bao gồm:

1. Hỗ trợ nhiều ngôn ngữ: Monaco Editor hỗ trợ nhiều ngôn ngữ lập trình phổ biến, bao gồm JavaScript, TypeScript, HTML, CSS và nhiều ngôn ngữ khác.
2. Cú pháp làm nổi bật và kiểm tra lỗi: Trình soạn thảo này cung cấp tính năng làm nổi bật cú pháp (syntax highlighting) cho các ngôn ngữ lập trình, giúp làm nổi bật cú pháp của các từ khóa, biến và cấu trúc ngôn ngữ. Nó cũng có thể kiểm tra lỗi cú pháp và cung cấp gợi ý khi bạn gõ mã.
3. Tích hợp đa dạng: Monaco Editor có thể tích hợp dễ dàng vào các ứng dụng web và ứng dụng desktop, cho phép bạn tạo ra các trình soạn thảo mã nguồn mạnh mẽ trực tuyến.
4. Tùy chỉnh linh hoạt: Bạn có thể tùy chỉnh Monaco Editor để phù hợp với nhu cầu cụ thể của ứng dụng của mình, từ cấu hình cú pháp làm nổi bật cho đến tính năng gợi ý và kiểm tra lỗi.

Monaco Editor đã trở thành một trong những lựa chọn phổ biến cho các nhà phát triển muốn tích hợp một trình soạn thảo mã nguồn mạnh mẽ vào ứng dụng web của họ, nhờ vào tính linh hoạt, hiệu suất và tính năng tiên tiến mà nó mang lại.

2.2.8. Cơ sở dữ liệu Supabase

Supabase là một nền tảng cơ sở dữ liệu mã nguồn mở và hệ thống quản lý dữ liệu (Database Management System - DBMS) được xây dựng dựa trên PostgreSQL và có tích hợp các tính năng của Firebase. Nó cung cấp một loạt các dịch vụ và công cụ giúp phát triển ứng dụng web một cách nhanh chóng và hiệu quả.

Hình 2.9. Cơ sở dữ liệu supabase.

Dưới đây là một số điểm nổi bật của Supabase:

Cơ sở dữ liệu PostgreSQL: Supabase sử dụng PostgreSQL, một hệ quản trị cơ sở dữ liệu mạnh mẽ và linh hoạt, để lưu trữ dữ liệu. PostgreSQL được biết đến với tính bảo mật cao, tính linh hoạt và khả năng mở rộng.

Realtime: Supabase cung cấp tính năng realtime, cho phép các ứng dụng tương tác với dữ liệu một cách tức thời và đồng bộ giữa các thiết bị.

Authentication: Nền tảng này cung cấp các dịch vụ xác thực (authentication) cho phép người dùng đăng nhập và quản lý thông tin cá nhân một cách dễ dàng.

Storage: Supabase cung cấp dịch vụ lưu trữ tệp (file storage) để lưu trữ hình ảnh, video và các tệp tin khác.

RESTful API: Supabase tự động tạo ra các API RESTful cho cơ sở dữ liệu của bạn, giúp việc tương tác với dữ liệu trở nên đơn giản và tiện lợi.

Client Libraries: Supabase cung cấp các thư viện client cho nhiều ngôn ngữ lập trình khác nhau như JavaScript, Python, và Go, giúp việc phát triển ứng dụng trở nên dễ dàng hơn.

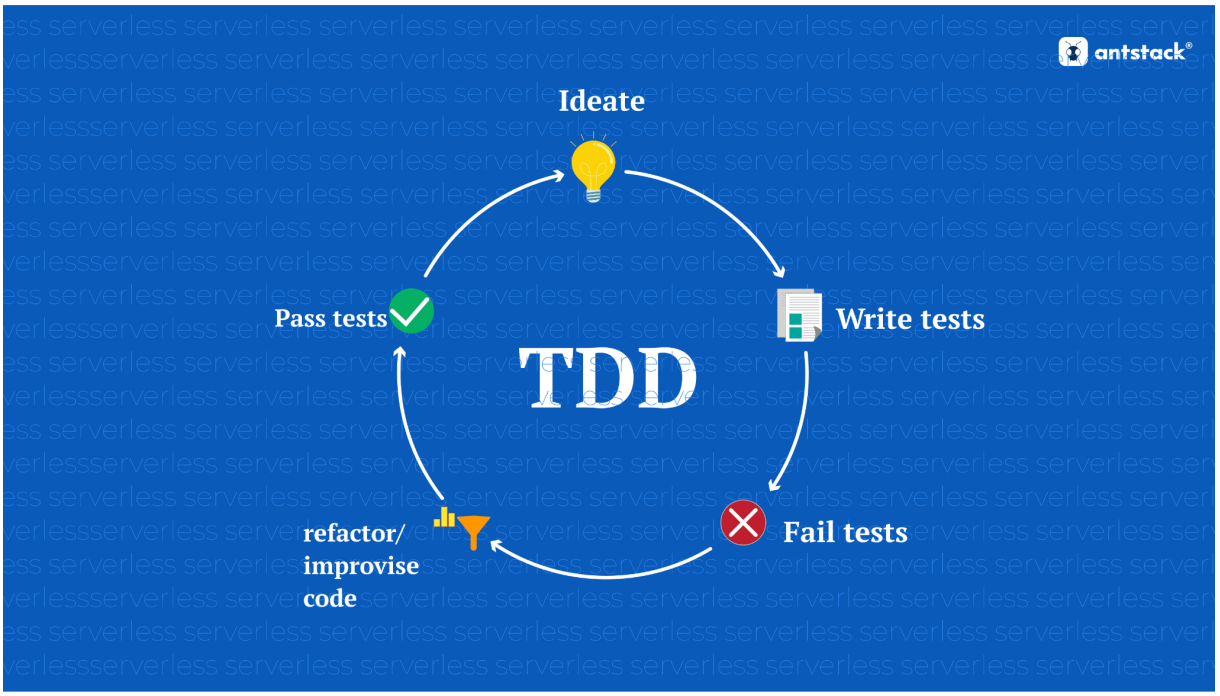
2.4. Phương pháp nghiên cứu

Phương pháp nghiên cứu kết hợp giữa việc tìm hiểu lý thuyết, xây dựng bộ chuyển đổi và phân tích cú pháp, sử dụng công cụ và thư viện có sẵn để xây dựng và phát triển hoàn thiện ngôn ngữ lập trình Tiếng Việt.

Đầu tiên, nghiên cứu sẽ được tiến hành để hiểu về cách mà máy tính đọc và biên dịch cú pháp và các công nghệ liên quan như ngôn ngữ lập trình, cú pháp, trình phân tích cú pháp, và các thư viện mã editor. Xác định các yêu cầu và phạm vi của ngôn ngữ lập trình tiếng Việt. Điều này bao gồm việc xác định các tính năng, đầu vào và đầu ra mong muốn của đề tài.

Test-driven development (TDD) là một phương pháp phát triển phần mềm mà trong đó các unit-test được viết trước khi việc triển khai mã. Quá trình này bắt đầu bằng việc viết unit-test tự động cho một tính năng hoặc chức năng cụ thể mà từng hàm/ hệ thống cần thực hiện. Sau đó, mã nguồn được triển khai để làm cho bài kiểm tra đó thành công. Quá trình lặp đi lặp lại này liên tục: viết một unit-test, triển khai mã nguồn để làm cho unit-test đó thành công và cải thiện mã nguồn nếu cần thiết.

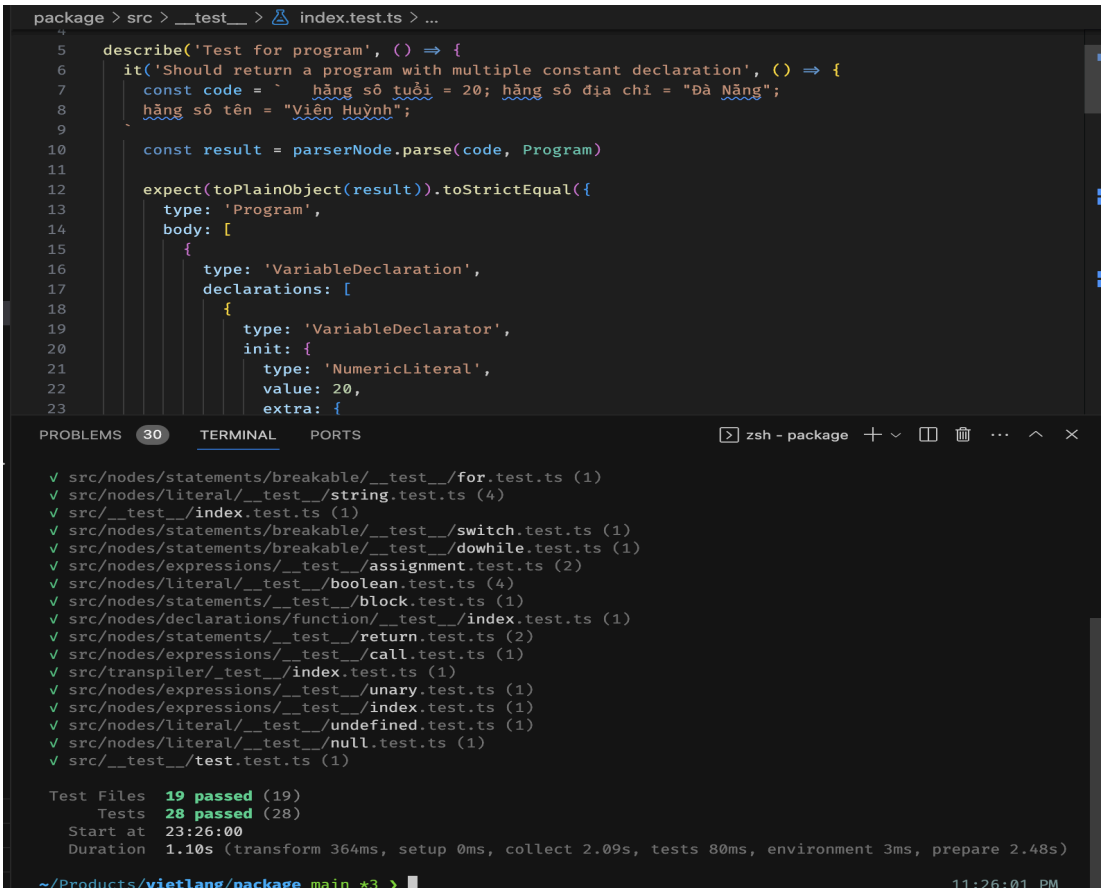
TDD không chỉ giúp đảm bảo rằng mã nguồn hoạt động đúng theo mong muốn mà còn tạo ra một tập hợp mạnh mẽ các unit-test tự động để đảm bảo tính ổn định của phần mềm trong quá trình phát triển và duy trì sau này. Nó cũng khuyến khích việc viết mã nguồn có thể kiểm tra được và dễ bảo trì, do đó làm tăng tính linh hoạt và khả năng mở rộng của dự án phần mềm.



Hình 2.10. Nguyên lý hoạt động của phương pháp Test-driven development

Các bước chính trong quy trình TDD bao gồm:

1. Xác định yêu cầu.
2. Viết test case: Viết unit test tập trung vào việc kiểm tra hành vi mong muốn của tính năng đó.
3. Chạy bài kiểm tra (run unit-test): Chạy unit-test để xác định xem nó có thất bại hay không. Lần chạy đầu tiên, unit-test sẽ thất bại vì chưa có mã nguồn triển khai cho tính năng đó.
4. Triển khai mã: Viết mã nguồn cần thiết để làm cho bài kiểm tra thành công.
5. Chạy lại unit-test: Chạy lại bài kiểm tra sau khi viết mã nguồn. Nếu bài kiểm tra vượt qua, điều này có nghĩa là mã nguồn đã được triển khai thành công và tính năng hoạt động theo mong muốn.
6. Tối ưu và làm sạch mã nguồn: Kiểm tra mã nguồn và làm sạch nó theo cách tốt nhất có thể mà không làm thay đổi hành vi của tính năng. Mục tiêu là giảm thiểu sự trùng lặp, tăng tính linh hoạt và dễ bảo trì của mã nguồn.
7. Lặp lại: Lặp lại các bước trên cho các tính năng tiếp theo hoặc mở rộng tính năng hiện tại.



Hình 2.11. Kết quả chạy thành công các bài kiểm thử đơn vị

Sử dụng phương pháp TDD trong quá trình phát triển giúp đảm bảo sự chính xác và độ tin cậy của từng hàm phân tích cú pháp. Việc viết unit test trước giúp xác định rõ yêu cầu và kết quả mong đợi, từ đó định hình cấu trúc và chức năng của hệ thống. Các unit test cũng đóng vai trò như một tài liệu thực thi, giúp kiểm tra và đánh giá hiệu quả của hệ thống theo các yêu cầu đã định sẵn.

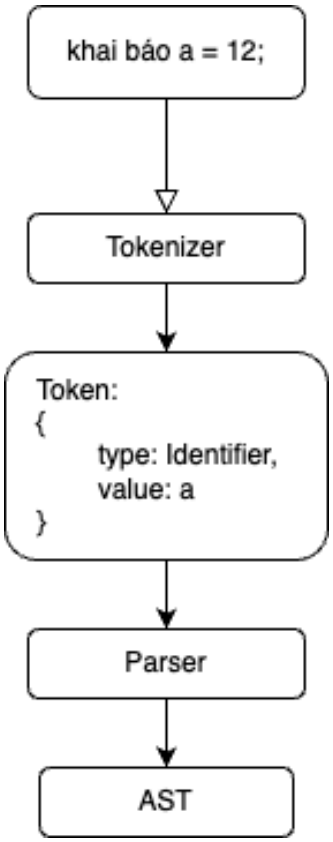
TDD cũng đảm bảo sự linh hoạt trong việc thay đổi và mở rộng hệ thống. Khi có thêm yêu cầu mới hoặc cải tiến, chỉ cần viết thêm unit test mới và triển khai mã tương ứng. Quá trình chạy unit test và test thành công sẽ đảm bảo tính ổn định và chất lượng của hệ thống khi được mở rộng.

Việc kết hợp phương pháp Test-driven development trong quá trình nghiên cứu và phát triển giúp đảm bảo tính chính xác, độ tin cậy và linh hoạt của hệ thống tự tạo sơ đồ khối.

CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

3.1. Thiết kế hệ thống tổng quan

Tổng quan, ngôn ngữ lập trình tiếng Việt sẽ nhận đầu vào là những đoạn mã bằng tiếng Việt. Sau đó hệ thống sẽ phân tích, chuyển đổi các đoạn mã sang dạng từ tố tương ứng với từng loại (biến, hàm, biểu thức, …). Tiếp theo sử dụng bộ phân tích cú pháp để phân tích. Hệ thống cung cấp sản phẩm giao diện người dùng dễ sử dụng thực thi các đoạn mã trực tiếp ở trên browser và có khả năng mở rộng cho các ngôn ngữ và môi trường phát triển khác.



Hình 3.1. Quá trình chuyển đổi cú pháp tiếng Việt sang AST

1. Quá trình bắt đầu bằng việc phân tích cú pháp của đoạn mã tiếng Việt đầu vào. Công đoạn này sử dụng các công cụ phân tích cú pháp để nhận diện các phần tử cú pháp trong mã, như biến, hàm, điều kiện, vòng lặp và các lệnh khác.

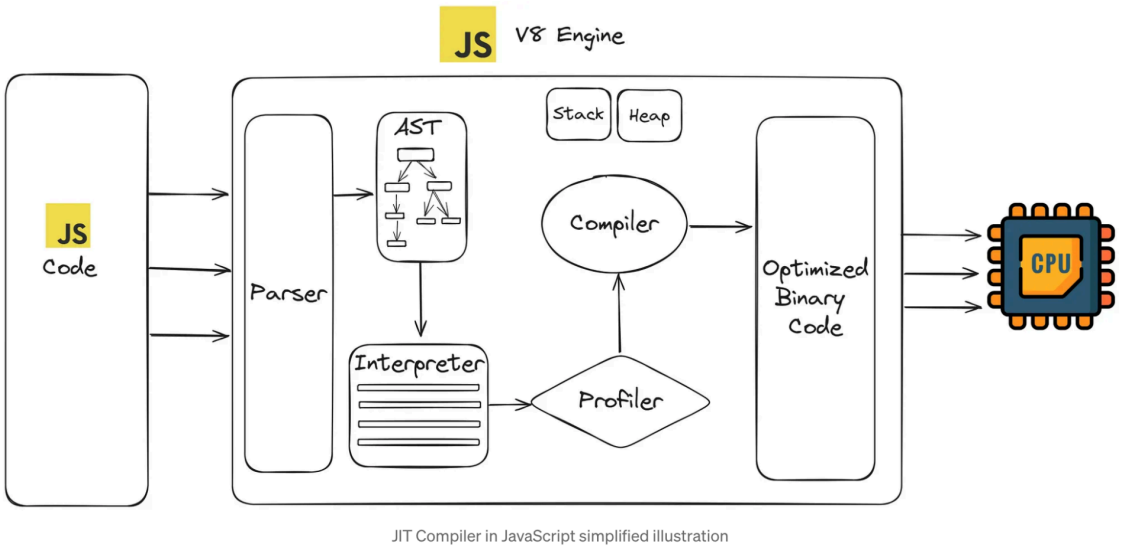
   Ví dụ:

   khai báo tên = “Viên Huỳnh”

Đoạn mã trên sẽ được phân tích thành cây cú pháp sau:

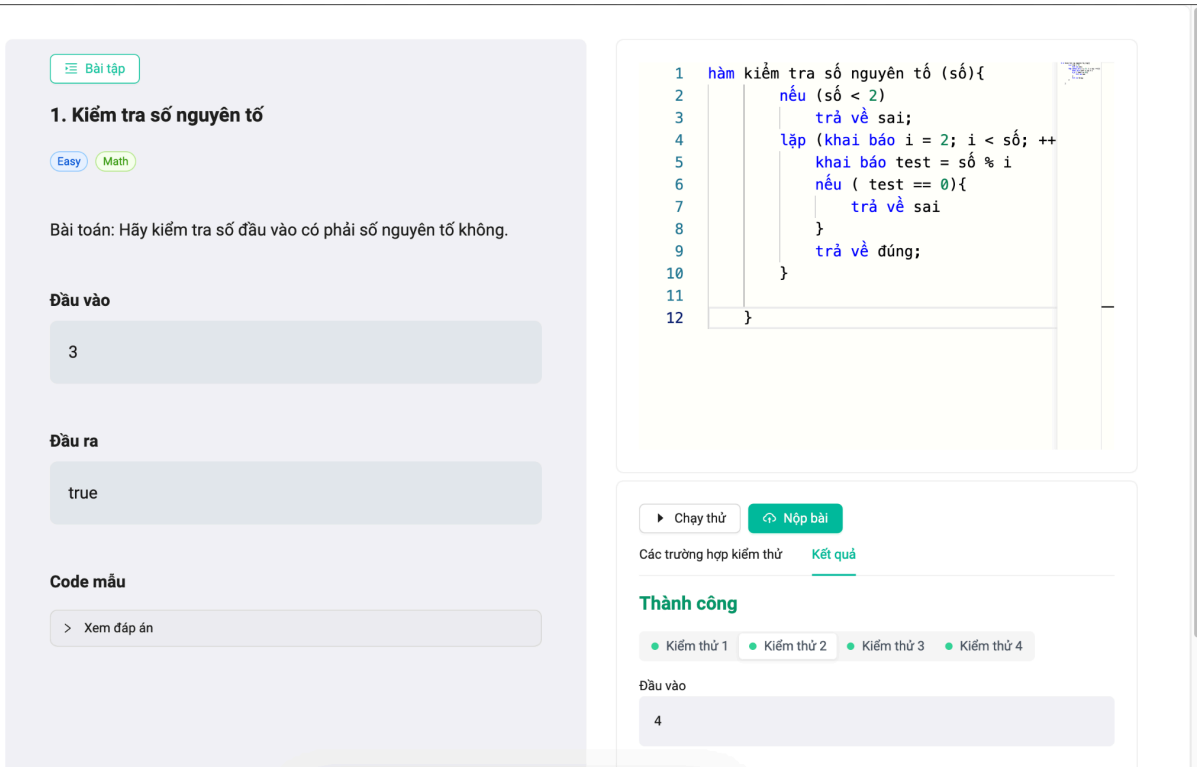
Hình 3.2. Cây cú pháp trừu tượng của ngôn ngữ VieLang

2. Dựa trên kết quả của quá trình phân tích cú pháp, hệ thống bắt đầu chuyển đổi thông dịch sang đoạn mã javascript. Sau khi mã nguồn được chuyển đổi sang JavaScript, ta sử dụng môi trường Node.js để biên dịch mã



Hình 3.3. Quy trình thực thi code của V8 Engine

3. Đồng thời, cần phát triển các ứng dụng người dùng để tạo giao diện đồ họa, cho phép người dùng trực tiếp tương tác trên trình duyệt web.



Hình 3.4. Giao diện trình duyệt của VieLang

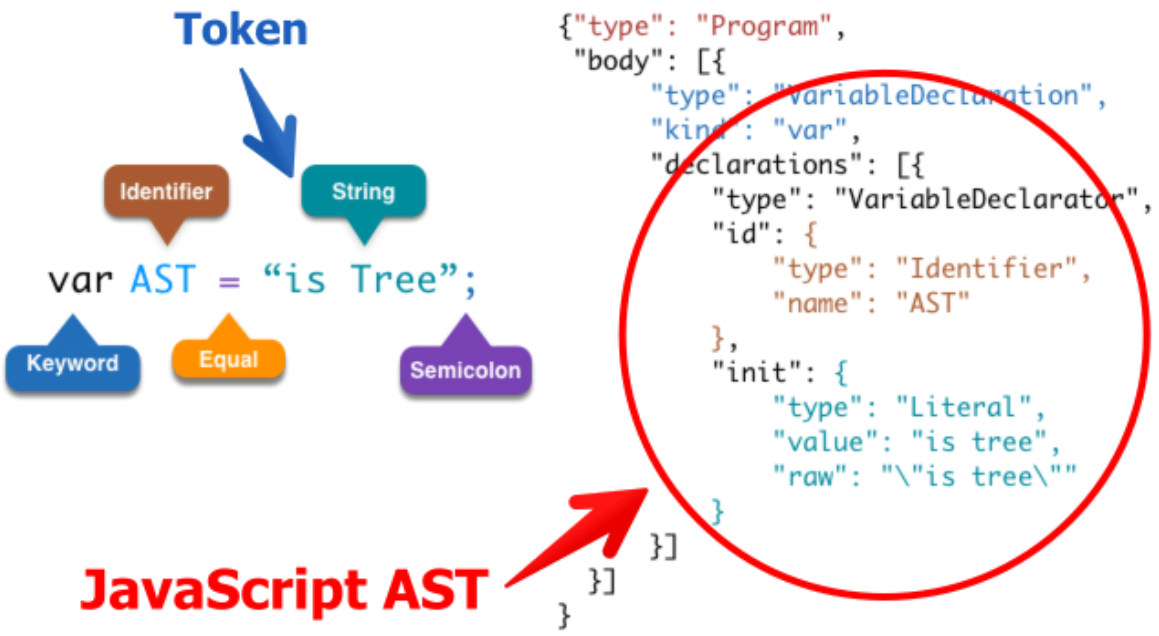
3.2. Xây dựng thuật toán phân tích cú pháp

3.2.1. Xây dựng bộ phân tách từ (Tokenizer)

Tokenizer là một công cụ hoặc hàm dùng để phân chia một chuỗi ký tự thành các đơn vị nhỏ hơn gọi là "token". Các từ tố này thường là các từ, số, dấu câu, hoặc các phần tử cú pháp khác trong ngôn ngữ lập trình hoặc văn bản. Mỗi từ tố thường kèm theo thông tin về loại của nó (ví dụ: từ khóa, biến, toán tử, số, dấu câu).

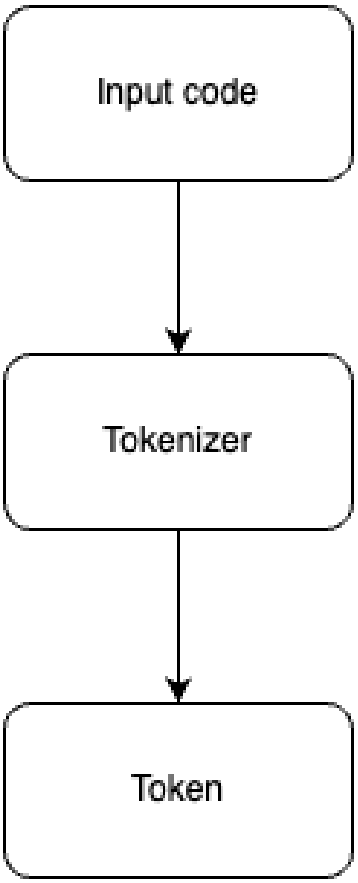
Để xây dựng cây cú pháp (syntax tree). Tokenizer là bước đầu tiên trong quá trình này, và chức năng cụ thể của nó bao gồm:

- Phân Tách Chuỗi Ký Tự: Chia chuỗi ký tự dài thành các từ tố đơn lẻ.
- Loại Bỏ Ký Tự Thừa: Loại bỏ các ký tự không cần thiết như khoảng trắng, dấu xuống dòng, hoặc chú thích.
- Nhận Dạng Loại Token: Gắn nhãn cho từng từ tố với loại cụ thể của nó (ví dụ: từ khóa, biến, số).
- Cung Cấp Token Cho Parser: Truyền các từ tố này cho parser để tiếp tục phân tích cú pháp.



Hình 3.5. Ví dụ về cú pháp của một đoạn mã

Dưới đây là quy trình hoạt động của bộ phân tách từ (Tokenizer):



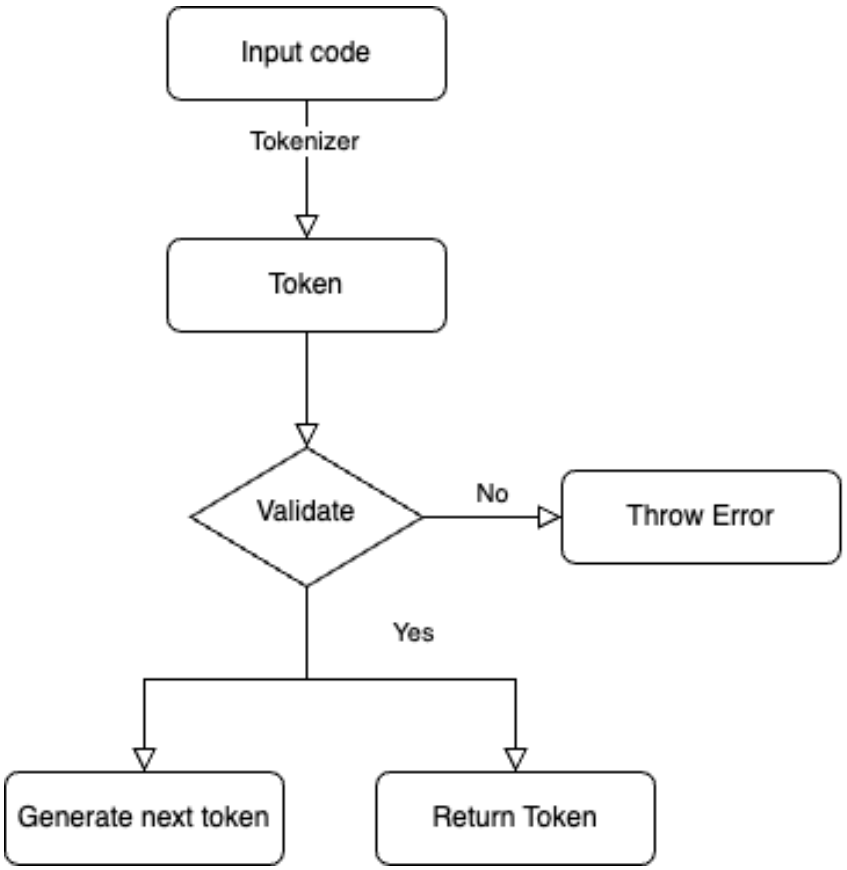
Hình 3.6. Quá trình phân tách từ thành dạng từ tố,

3.2.2. Xây dựng bộ phân tích từ vựng

Vì ngôn ngữ lập trình tiếng Việt - VieLang dựa trên cây cú pháp của ngôn ngữ lập trình Javascript, do đó quá trình xây dựng cây cú pháp trừu tượng yêu cầu kiến thức về ngữ pháp và quy tắc cú pháp của Javascript. Quá trình này phức tạp và đòi hỏi nhiều công sức và kiến thức sâu về ngôn ngữ lập trình. Ta phải phân tích, chuyển đổi ngôn ngữ tiếng Việt thành các node như từ khoá, biến, câu lệnh, vòng lặp, biểu thức, … Các tên hàm tên biết hay biểu thức đều được hỗ trợ dưới dạng tiếng Việt có dấu cách

Bước đầu tiên để xây dựng bộ phân tích cú pháp là định nghĩa các từ khoá. Tại đây, sử dụng unicode cho các từ khoá tiếng Việt, để đảm bảo tiếp nhận thông tin từ khoá đúng, chuẩn bất kể từ môi trường nào (window, macos, …)

Tiếp theo sử dụng từ tố được trả ra từ bộ phân tách từ (Tokenizer) để kiểm tra và xây dựng cây cú pháp trừu tượng.



Hình 3.7. Luồng hoạt động của bộ phân tách từ.

Từ bộ từ tố trên chuyển đổi thành cây cú pháp. Ví dụ:


```
{
  "type": "VariableDeclaration",
  "declarations": [
    {
      "type": "VariableDeclarator",
      "init": {
        "type": "StringLiteral",
        "start": 15,
        "end": 27,
        "value": "Viên Huỳnh",
        "extra": {
          "rawValue": "Viên Huỳnh",
          "raw": "\"Viên Huỳnh\""
        }
      },
      "id": {
        "type": "Identifier",
        "name": "t_234n"
      }
    }
  ],
  "kind": "let"
}
```


Hình 3.8. Kết quả của quá trình phân tách từ

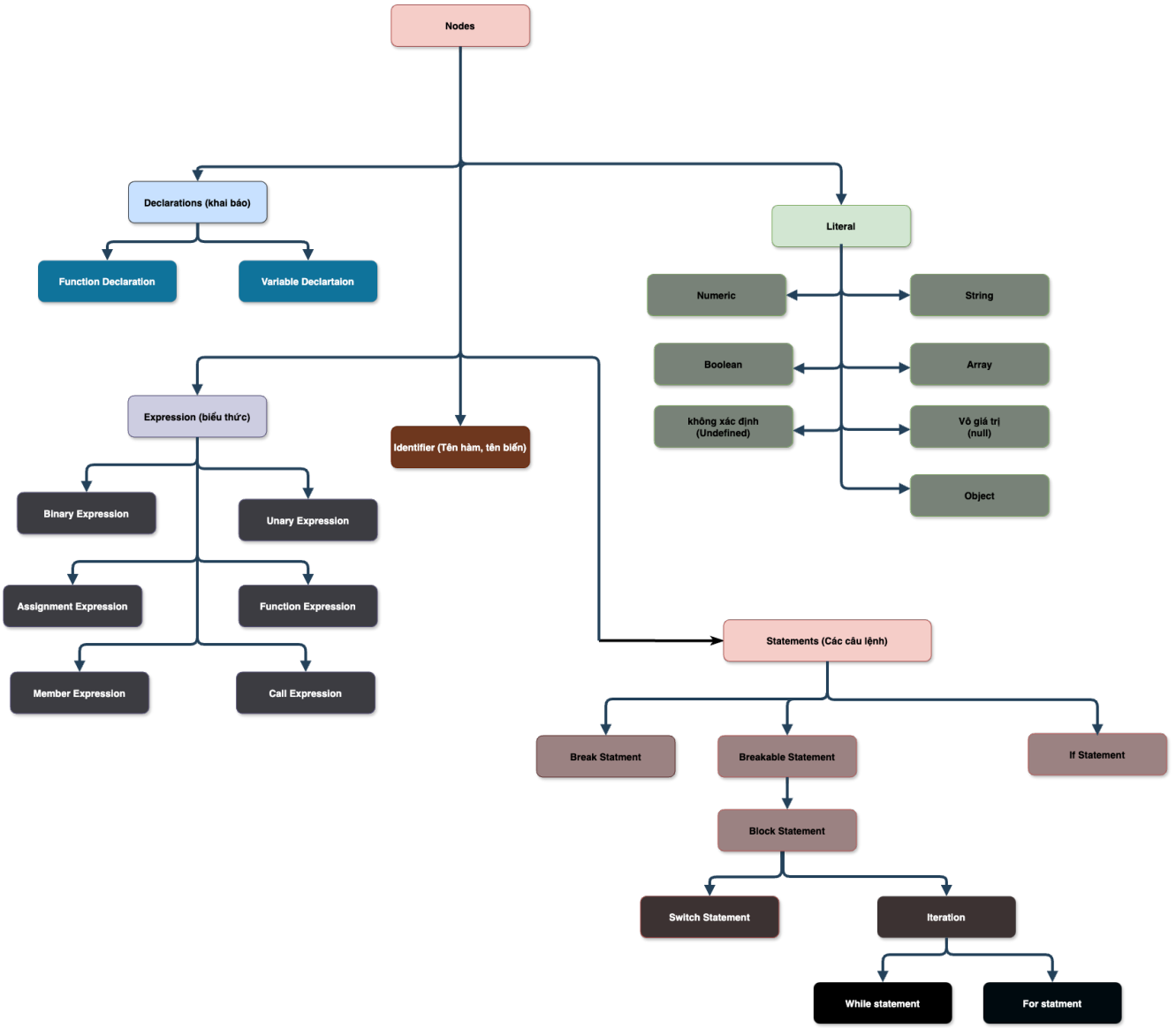
Regex	Loại
`/^[A-Za-z\u00C0-\u1EF9]+(\s[A-Za-z\u00C0-\u1EF9]+)*/`	Tên hàm, biến
`/^\s+/`	Dấu cách
`^\+\+/`	Toán từ “++”
`/^[+\-]/`	Toán tử: +, -
`/^[*\/\%]/`	Toán tử : * , /, %
`/^(<=|>=|<|>|==)/`	Toán tử quan hệ: >, >= , <, <=, ==
`/^\b(let|khai b\u00E1o)\b/`	Tử khoá “khai báo” (let)
`[/^const|h\u1EB1ng s\u1ED1/`	Từ khoá “hằng số” (const)
`[/^(\d+(\.|)(\d+|)([Ee]([+-]|)\d+|))/`	Số
`/^"[^"]*"/`	Chuỗi
`/^'[^']*'/`	Chuỗi
`/(true|false|\u0111\u00FAng|sai)\b/`	Từ khoá boolean: “đúng” hoặc “sai”
`/^\b(break|ph\u00E1v\u00F2ng l\u1EB7p)\b/,`	Tử khoá “phát vòng lặp” (break)
`/\u0074\u0068\u1EF1\u0063\u0020\u0068\u0069\u1EC7\u006E/g`	Từ khoá “thực hiện” (do)

`/^\b(switch|duy\u1EC7t)\b/`	Từ khoá “duyệt” (switch)
`/^\b(case|tr\u01B0\u1EDDng h\u1EE3p)\b/`	Từ khoá “trường hợp” (case)
`/^\b(if|n\u1EBFu)\b/`	Từ khoá “nếu” (if)
`/^\b(else|kh\u00F4ng th\u00EC)/`	Từ khoá “ngược lại” (else)
`[/^\b(return|tr\u1EA3 v\u1EC1)/`	Từ khoá “trả về” (return)
`/^\b(continue|ti\u1EBFp t\u1EE5c)\b/`	Từ khoá “tiếp tục” (continue)
`/^\b(for|\u006C\u1EB7\u0070)\b/`	Từ khoá “lặp” (for)
`/\u006B\u0068\u0069\u0020\u006D\u00E0/g`	Từ khoá “khi mà” (while)
`/^\b(function|h\u00E0m)\b/`	Từ khoá “hàm” (function)
`/^\b(default|m\u1EB7c \u0111\u1ECBnh)\b/`	Từ khoá “mặc định” (default)

Bàng 3.1. Bàng định nghĩa từ khóa bằng biểu thức chính quy để nhận diện cú pháp.

3.3. Phân tích bộ từ tố hệ thống (Common Token)

Dưới đây là tổng quan sơ đồ hệ thống:



Hình 3.9. Tổng quan sơ đồ cú pháp hệ thống

3.3.1. Nhận diện tên hàm

Tên hàm và tên biến là các định danh trong mã nguồn của ngôn ngữ lập trình. Chúng thường tuân theo các quy tắc cụ thể của ngôn ngữ lập trình đó. Đối với ngôn ngữ lập trình VieLang, tên hàm tên biến được chấp nhận các kí tự có dấu và dấu cách

a) Biểu thức chính quy nhận diện tên biến, tên hàm (Identifier)

Sử dụng cú pháp Regex: /^[A-Za-z\u00C0-\u1EF9]+(\s[A-Za-z\u00C0-\u1EF9]+)*/ để nhận diện tên hàm, tên biến. Cú pháp regex trên giúp nhận diện các kí tự từ A-Z bao gồm ký tự có dấu và dấu cách.

```
: SpecIdentifier = [/^[A-Za-z\u00C0-\u1EF9]+(\s[A-Za-z\u00C0-\u1EF9]+)*/, Keyword.IDENTIFIER] as Spec
```

Hình 3.10. Biểu thức chính quy nhận diện tên hàm, tên biến

Giải thích biểu thức chính quy:

[A-Za-z\u00C0-\u1EF9]: Đây là một tập hợp các ký tự (character class) bao gồm:

A-Z: Các chữ cái viết hoa từ A đến Z.

a-z: Các chữ cái viết thường từ a đến z.

\u00C0-\u1EF9: Các ký tự Unimã từ À (U+00C0) đến ỹ (U+1EF9), bao gồm các ký tự có dấu và một số ký tự đặc biệt trong tiếng Việt.

+: Dấu cộng biểu thị rằng tập hợp ký tự trước đó phải xuất hiện ít nhất một lần (1 hoặc nhiều lần).

\s: Dấu cách.

b) Nhận diện tên biến

Tên biến thường xuất hiện sau các từ khóa khai báo kiểu dữ liệu (“khai báo”, “hằng số”).

Ví dụ:

```
1    khai báo tên = "Viên Huỳnh"
2    khai báo tuổi = 22
```

Hình 3.11. Ví dụ về khai báo biến bằng ngôn ngữ VieLang

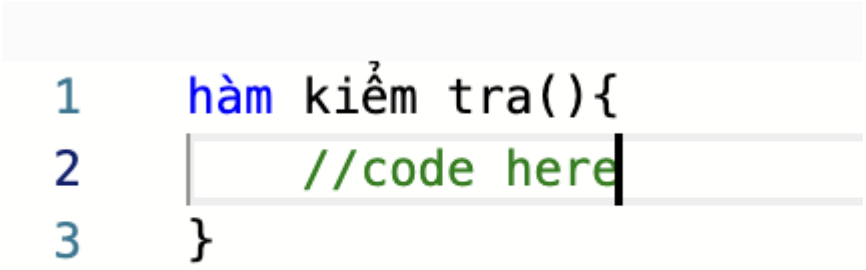
Thuật toán nhận diện;

- Quét từ tố từ tokenizer.
- Nếu gặp từ tố là từ khóa khai báo biến (“khai báo”, “hằng số”.), kiểm tra từ tố tiếp theo.
- Token tiếp theo nếu là định danh (identifier), gán nhãn là tên biến (variable).

c) Nhận diện tên hàm

Tên biến thường xuất hiện sau các từ khóa “hàm”.

Ví dụ:



Hình 3.12. Ví dụ về khai báo hàm bằng ngôn ngữ VieLang

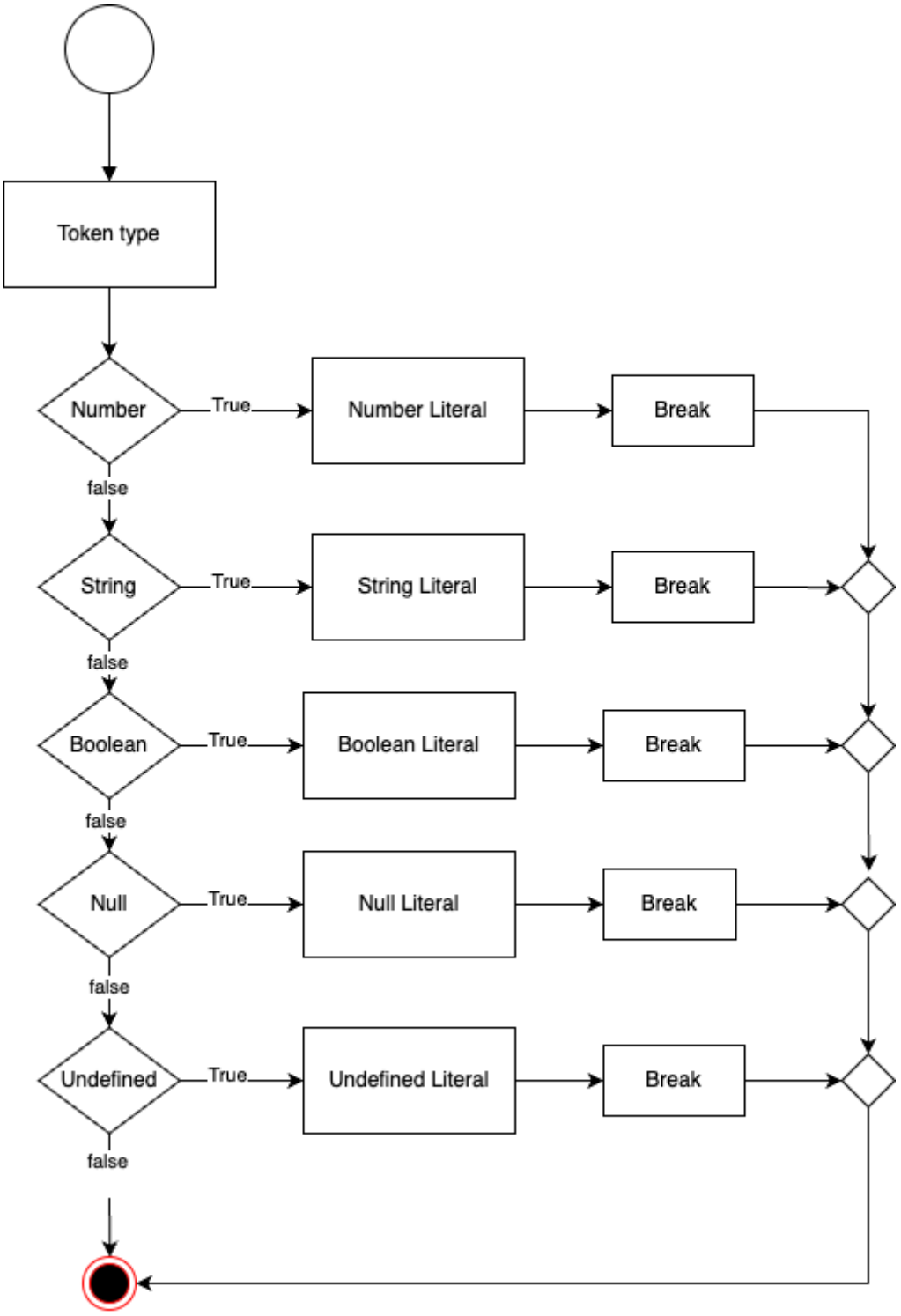
Thuật toán nhận diện:

- Quét từ tố từ tokenizer.
- Nếu gặp từ tố là từ khóa “hàm”, kiểm tra từ tố tiếp theo.
- Token tiếp theo nếu là định danh (identifier), gán nhãn là hàm (function).

3.3.2. Hằng (Literal)

Literal là các giá trị cố định xuất hiện trực tiếp trong mã nguồn. Ngôn ngữ lập trình **VieLang** có các kiểu dữ liệu sau:

- Numeric Literal (Số).
- String Literal (Chuỗi).
- Null Literal (Vô giá trị).
- Undefined Literal (Không xác định).
- Boolean Litera (Đúng, sai).



Hình 3.13. Luồng hoạt động phân tích loại hằng.

Việc xây dựng parser literal là một bước quan trọng trong quá trình phân tích từ vựng và cú pháp của mã nguồn. Parser literal giúp xác định và phân loại các giá trị cố định trong mã nguồn, từ đó hỗ trợ các bước xử lý tiếp theo như xây dựng cây cú pháp và tối ưu hóa mã.

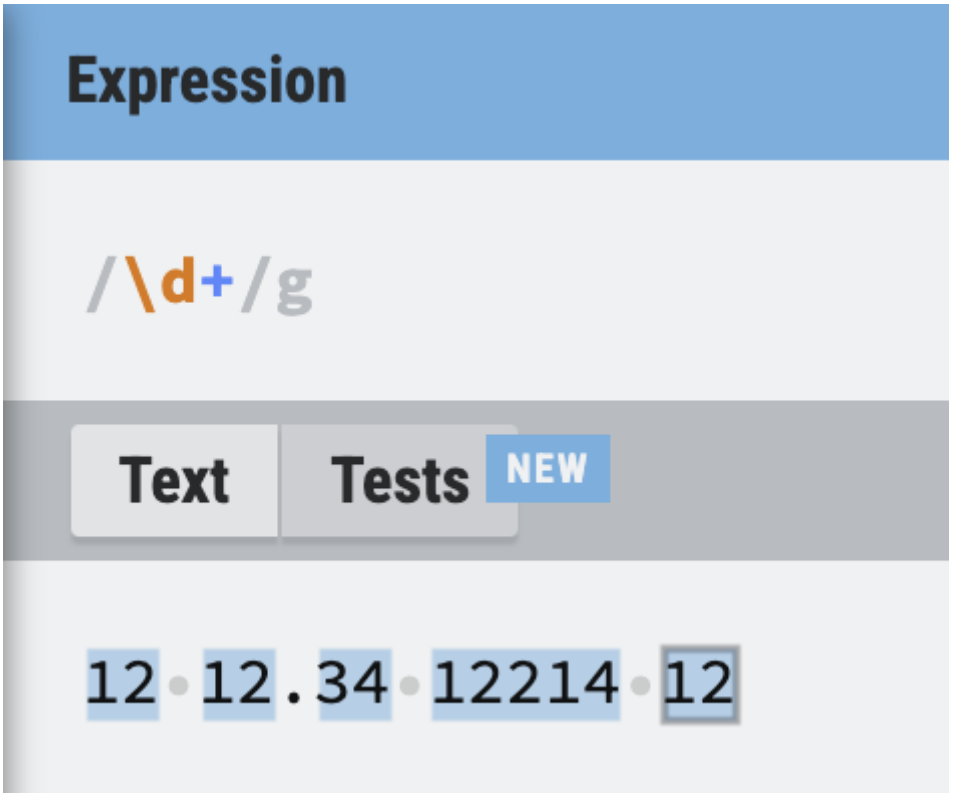
a) Phân tích từ tố số (Number Literal)

Đối với số, sử dụng cú pháp regex `[/^(\d+(\.|)(\d+|)([Ee]([+-]|)\d+|))/` để tìm, tách các cú pháp của số.

(\d+(\.|)(\d+|)([Ee]([+-]|)\d+|)): Đây là một nhóm con được bao bọc bởi dấu mở ngoặc đơn "(" và dấu đọc ngoặc đơn ")". Chúng ta sẽ phân tích chi tiết bên trong nhóm này.

\d+

- \d: khớp với bất kỳ chữ số nào (từ 0 đến 9).
- + : chỉ định rằng có một hoặc nhiều chữ số.

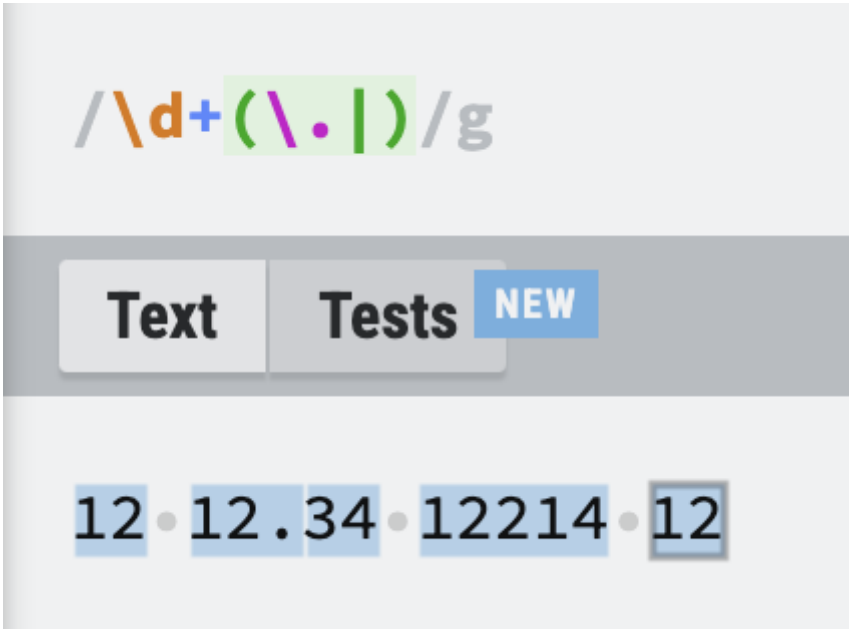


Hình 3.14. Kết quả của biểu thức chính quy \d

(\.|):

- . khớp với ký tự dấu chấm (dấu thập phân).
- | là toán tử OR, nghĩa là dấu chấm có thể xuất hiện hoặc không.

Như ta thấy dưới đây, biểu thức chính quy đã phát hiện được số thập phân “12.34”



Hình 3.15. Kết quả của biểu thức chính quy nhận diện số nguyên, số thực

([Ee]([+-]|)\d+|):

[Ee] khớp với chữ E hoặc e, ký hiệu cho số mũ trong biểu diễn khoa học.

([+-]|) là một nhóm con khác:

[+-] khớp với dấu + hoặc dấu - (cho biết số mũ là dương hay âm).

| là toán tử OR, nghĩa là có thể có hoặc không có dấu + hoặc -.

\d+ khớp với một hoặc nhiều chữ số (phần số mũ).

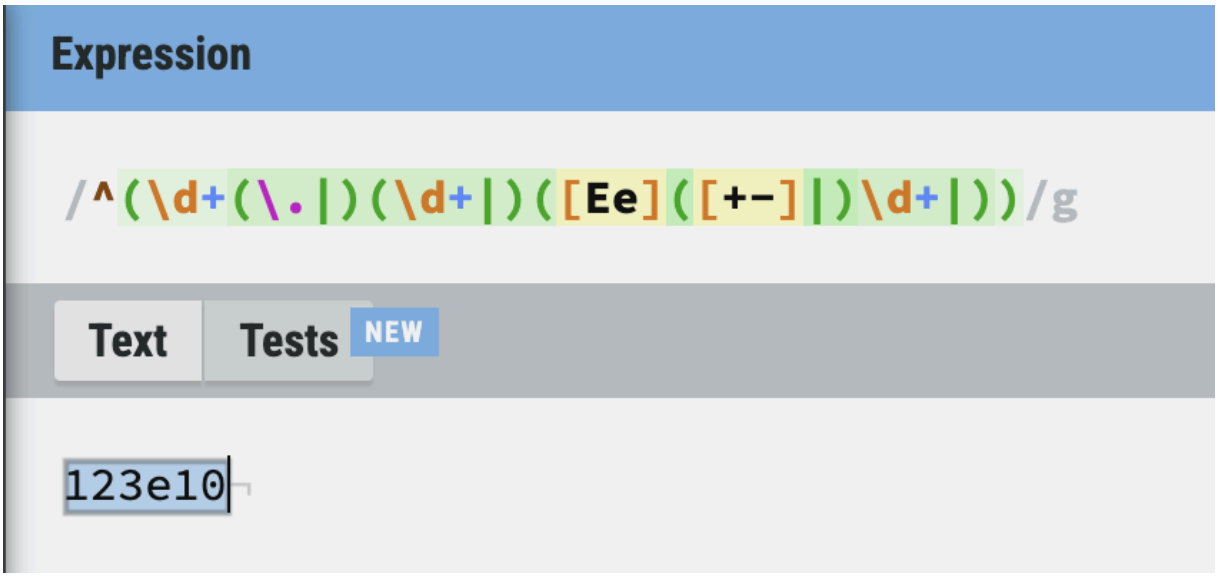
| là toán tử OR, nghĩa là phần số mũ có thể xuất hiện hoặc không.

Ví dụ:

- 123e10

- 123E+10

- 123.456e-10



Hình 3.16. Kết quả của biểu thức chính quy nhận diện số đối với số mũ

b) Phân tích từ tố chuỗi (String literal)

Chuỗi ký tự và ký tự đơn được nhận diện dựa trên dấu ngoặc kép và dấu ngoặc đơn bằng regex /^"[^"]*"/ (đối với ngoặc kép) và /^'[^']*'/ (đối với ngoặc đơn)
Trong đó (Ví dụ đối với ngoặc kép, tương tự đối với ngoặc đơn):

- ^ : Bắt đầu từ đầu chuỗi.
- ": Khớp với dấu ngoặc kép mở
- [^"]*: Khớp với bất kỳ ký tự nào ngoại trừ dấu ngoặc kép, lặp lại không hoặc nhiều lần
- ": Khớp với dấu ngoặc kép đóng.

Hình 3.17. Kết quả của biểu thức chính quy nhận diện chuỗi.

Thuật toán nhận diện:

- Quét từ tố từ tokenizer.
- Nếu từ tố được bao quanh bởi dấu ngoặc kép ", dấu ngoặc đơn ', gán nhãn là chuỗi ký tự (String Literal).

c) Nhận diện từ tố boolean (Boolean Literal)

Boolean được nhận diện dựa trên từ khóa "đúng" hoặc "sai".

Thuật toán nhận diện:

- Quét từ tố từ tokenizer.
- Nếu từ tố là "đúng" hoặc "sai", gán nhãn là boolean (Boolean Literal).

d) Nhận diện từ tố vô giá trị (Null literal)

Null literal được nhận diện dựa trên từ khóa "vô giá trị".

Thuật toán nhận diện:

- Quét từ tố từ tokenizer.
- Nếu từ tố là "vô giá trị", gán nhãn là vô giá trị (Null Literal).

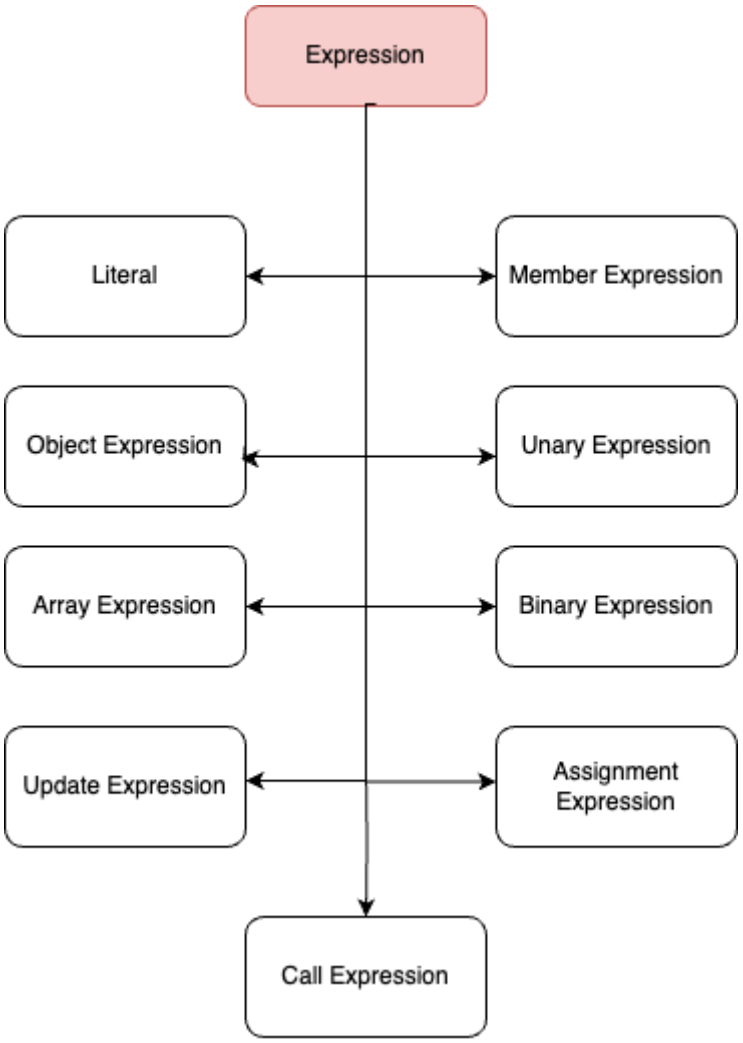
e) Nhận diện từ tố không xác định (Undefined Literal)

Undefined Literal được nhận diện dựa trên từ khóa "không xác định"

Thuật toán nhận diện:

- Quét từ tố từ tokenizer.
- Nếu từ tố là "không xác định", gán nhãn là không xác định (Undefined Literal).

3.3.3. Biểu thức (Expression)



Hình 3.18. Sơ đồ các loại biểu thức trong hệ thống ngôn ngữ VieLang

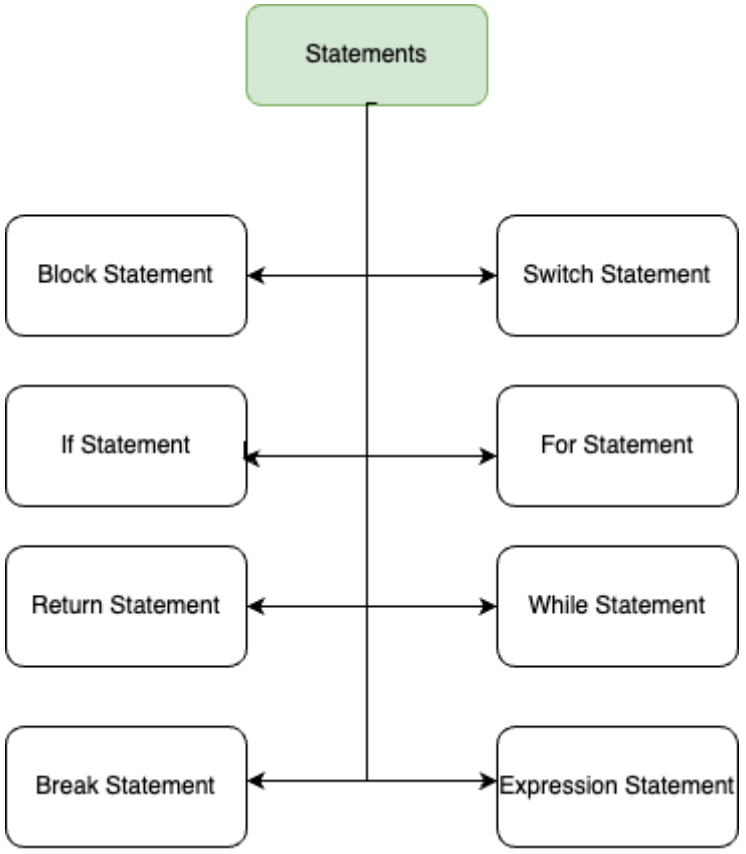
Hệ thống có các loại biểu thức (expression) sau:

- Literal: Tương ứng với number, string, undefined, null, boolean
- Object Expression: (hay còn gọi là object literal) là cách định nghĩa một đối tượng bằng cách sử dụng một tập hợp các cặp khóa-giá trị
- Array Expression: (hay còn gọi là array literal) là một cách để định nghĩa một mảng bằng cách sử dụng một tập hợp các giá trị được bao bọc bởi cặp dấu ngoặc vuông ([]). Array expressions là cách cơ bản và phổ biến để tạo ra các mảng trong VieLang, cho phép bạn dễ dàng định nghĩa và làm việc với các tập hợp dữ liệu.
- Update Expression: Các toán tử update: (++, --)
- Member Expression: là cách truy cập vào các thành viên (thành phần) của đối tượng, bao gồm cả thuộc tính và phương thức. Member expression sử dụng cú pháp dấu chấm (.) hoặc dấu ngoặc vuông ([]).
- Unary Expression: (biểu thức đơn ngôi) là một biểu thức chỉ có một toán hạng và một toán tử. Unary expression thực hiện một hoạt động duy nhất trên một

giá trị hoặc biến. Các toán tử đơn ngôi thường gặp bao gồm các toán tử dấu cộng và trừ (+, -) và phủ định (!)

- Binary Expression: (biểu thức nhị phân) là một biểu thức có hai toán hạng và một toán tử. Biểu thức nhị phân thực hiện một phép toán giữa hai giá trị hoặc biến. Các toán tử nhị phân bao gồm các phép toán số học, phép toán so sánh, phép toán logic, và các phép toán khác.
- Assignment Expression: (biểu thức gán) là một biểu thức sử dụng toán tử gán (=) để gán giá trị cho một biến. Assignment expression có thể sử dụng các toán tử gán khác nhau để thực hiện các phép toán khác nhau trong quá trình gán giá trị.

3.3.4. Các câu lệnh (Statements)



Hình 3.19. Sơ đồ các câu lệnh có trong ngôn ngữ lập trình VieLang.

a) Khối lệnh (Block Statement)

Được bao gồm trong cặp dấu ngoặc nhọn {}. Khối lệnh là cách để nhóm các lệnh lại với nhau, kiểm soát phạm vi biến và tạo các khối mã có cấu trúc.

Ví dụ:

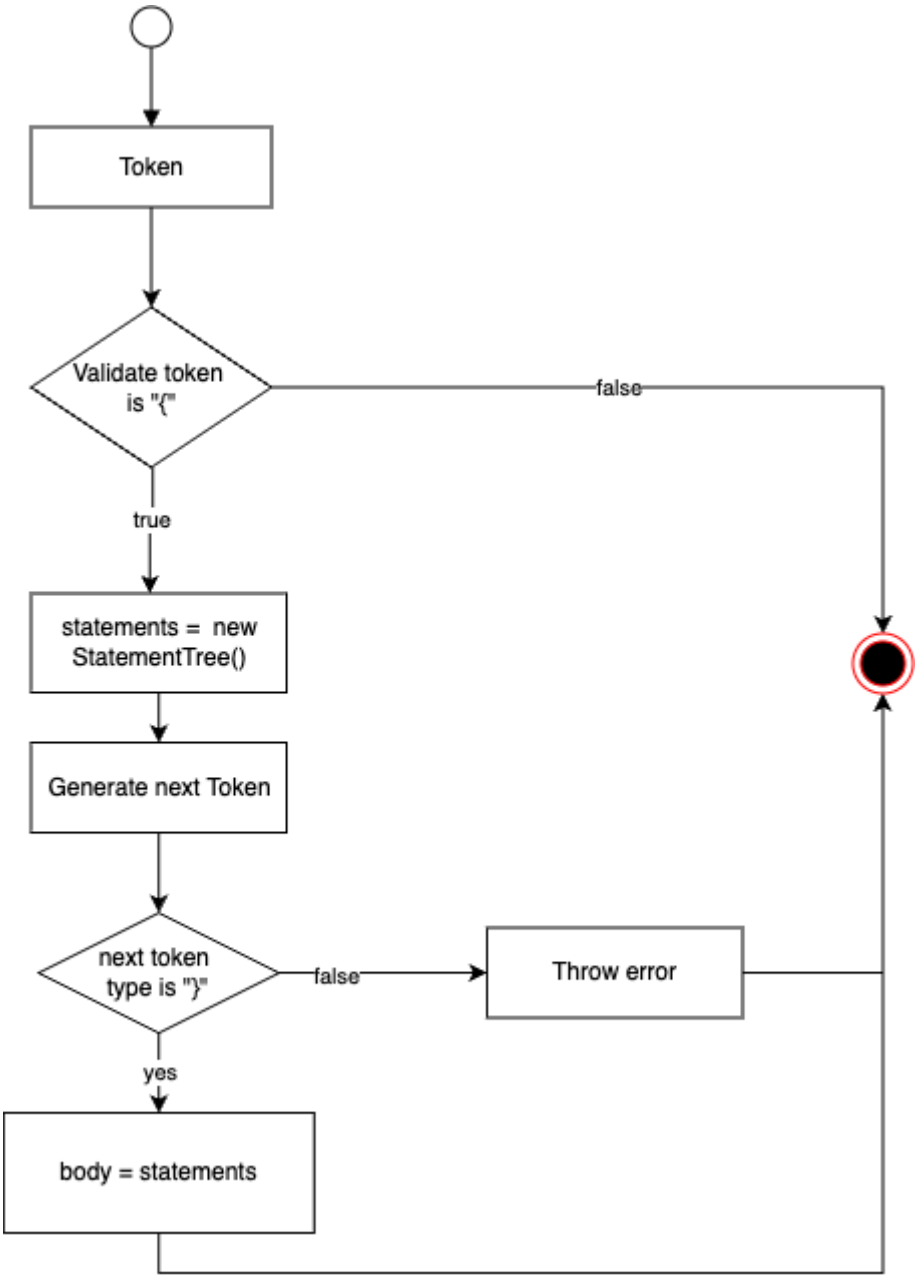
```
{
    khai báo tên = "viên"
    in ra (tên) // viên
}

in ra (tên) // Lỗi: "tên" chưa được khai báo
```

Hình 3.18. Ví dụ về khối lệnh

Trong ví dụ trên, các lệnh khai báo tên =”viên” và in ra (tên) được nhóm lại với nhau trong một khối. Khối này không có tên và được giới hạn phạm vi biến bên trong dấu ngoặc nhọn. Nếu truy vấn biến “tên” ở ngoài khối lệnh sẽ báo lỗi.

Khối lệnh được sử dụng trong các câu lệnh điều kiện như là if, esle, for, do-while, switch, while. Hay try-catch và function



Hình 3.20. Luồng hoạt động của quá trình phân tích cú pháp khối lệnh.

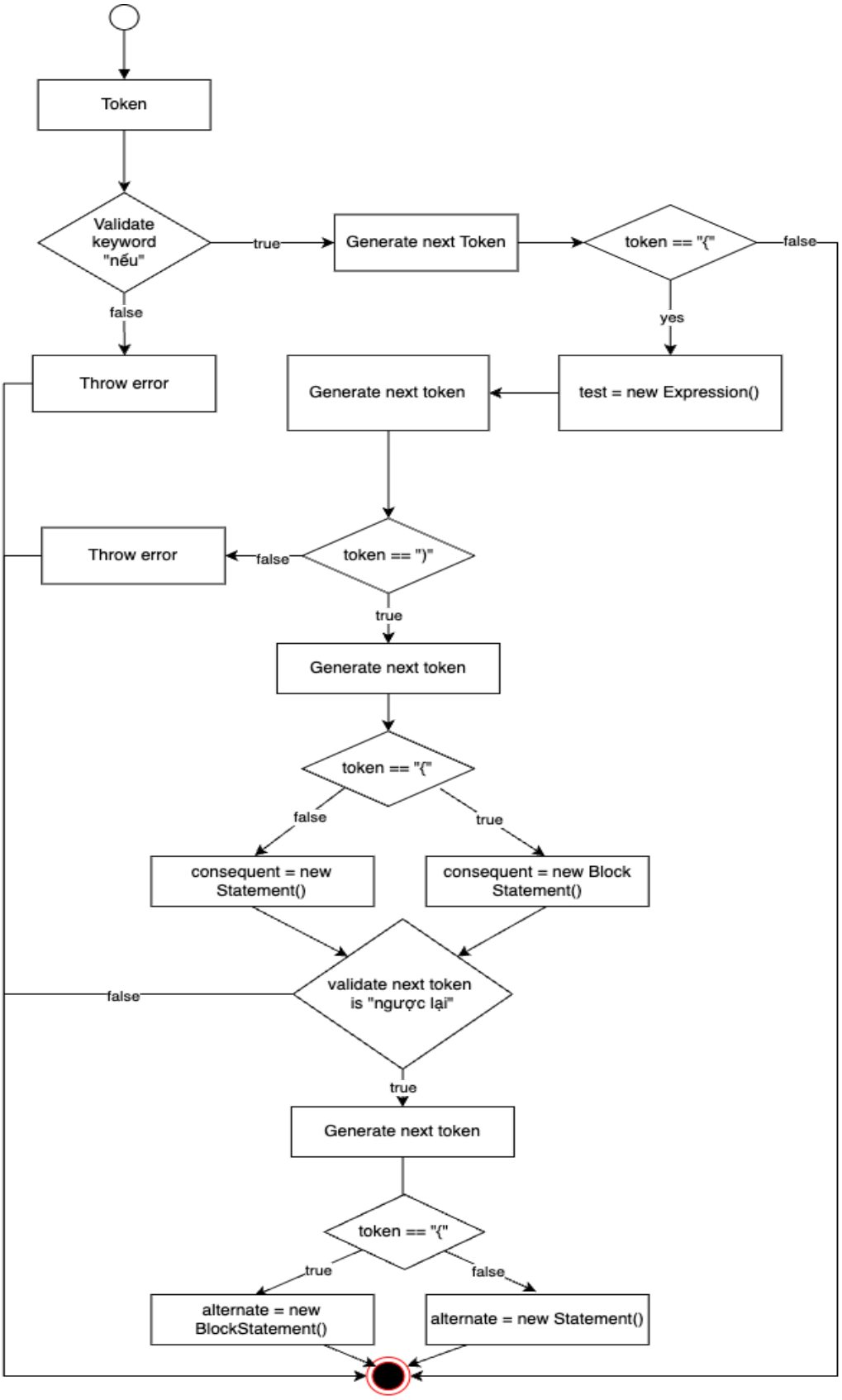
b) Câu lệnh điều kiện if (if statement)

Câu lệnh if (if statement) trong VieLang được sử dụng để thực thi một đoạn mã chỉ khi một điều kiện nhất định được thỏa mãn (đúng).

Ví dụ:

```
1    nếu(a % 2 == 0){
2        // Code
3    }
```

Hình 3.21. Ví dụ về câu lệnh điều kiện của ngôn ngữ lập trình VieLang.



Hình 3.22. Luồng hoạt động của quá trình phân tích cú pháp câu lệnh điều kiện.

Luồng phân tích cú pháp if-else:

- Kiểm tra từ tố đầu vào, là từ khoá "nếu" (if) thì phân tích câu lệnh điều kiện (if statement)
- Xác thực dấu mở ngoặc "(" và tạo biểu thức điều kiện. Sau từ khoá "nếu" phải có dấu "("
- Gán một biểu thức vào biến "test", đây là điều kiện của câu lệnh if-else.
- Xác thực dấu ngoặc ")", đảm bảo biểu thức điều kiện kết thúc bằng dấu ngoặc đóng
- Xác thực khối lệnh thực thi khi điều kiện đúng (consequent):
    + Kiểm tra từ tố tiếp theo có phải dấu mở ngoặc nhọn "{" không
    + Nếu đúng tạo 1 đối tượng Block Statement biểu diễn khối mã (một hoặc nhiều câu lệnh thực thi nằm trong cặp dấu ngoặc nhọn)
    + Nếu không tạo 1 đối tượng statement. Biểu diễn các câu lệnh đơn
- Kiểm tra từ khoá "ngược lại" (else). Nếu đúng, kiểm tra từ tố tiếp theo có phải dấu ngoặc nhọn không. Nếu là dấu ngoặc nhọn gán Block Statement cho biến "alternate", nếu sai sán Statement cho biến alternate

c) Câu lệnh trả về (Return Statement)

Câu lệnh trả về (return statement) trong VieLang được sử dụng để kết thúc một hàm và trả về một giá trị từ hàm đó. Khi return được thực thi, hàm sẽ dừng lại và giá trị được chỉ định sẽ được trả về cho nơi mà hàm được gọi

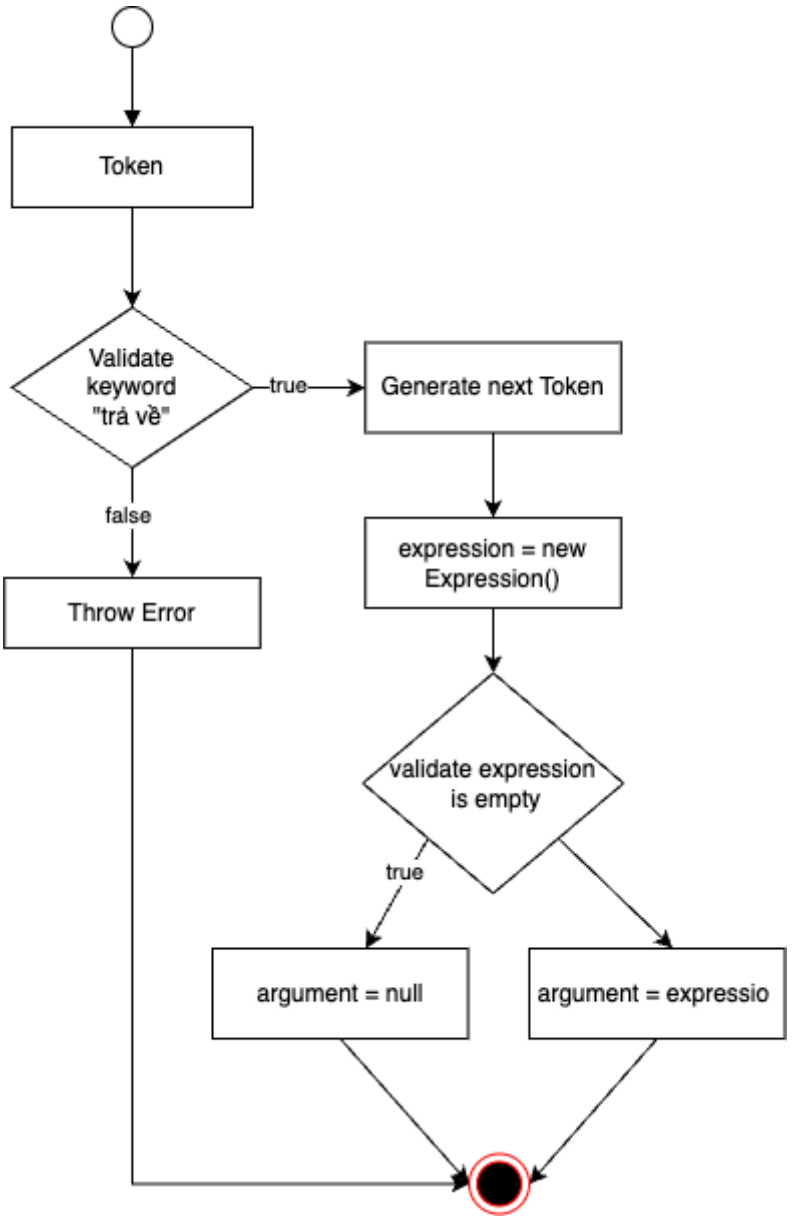
Ví dụ:

```
hàm kiểm tra số lẻ(){
    nếu(a % 2 != 0){
        trả về đúng
    }
    trả về sai
}
```

Hình 3.23. Ví dụ về câu lệnh trả về trong ngôn ngữ lập trình VieLang.

Luồng phân tích cú pháp trả về (return statement):

- Xác thực từ khoả "trả về". Kiểm tra xem từ khóa hiện tại có phải là "trả về" không. Nếu không, nó có thể báo lỗi cú pháp.
- Tạo 1 đối tượng Expression
- Kiểm tra expression. Nếu có, gán đối số cho return statement là "argument" là expression, nếu không gán "null"



Hình 3.24. Luồng hoạt động của quá trình phân tích cú pháp câu lệnh trả về.

d) Câu lệnh vòng lặp (For statement)

Được sử dụng để tạo ra một vòng lặp có điều kiện được kiểm tra trước mỗi lần lặp. Vòng lặp này thường được sử dụng để lặp qua một chuỗi hoặc một tập hợp các phần tử.

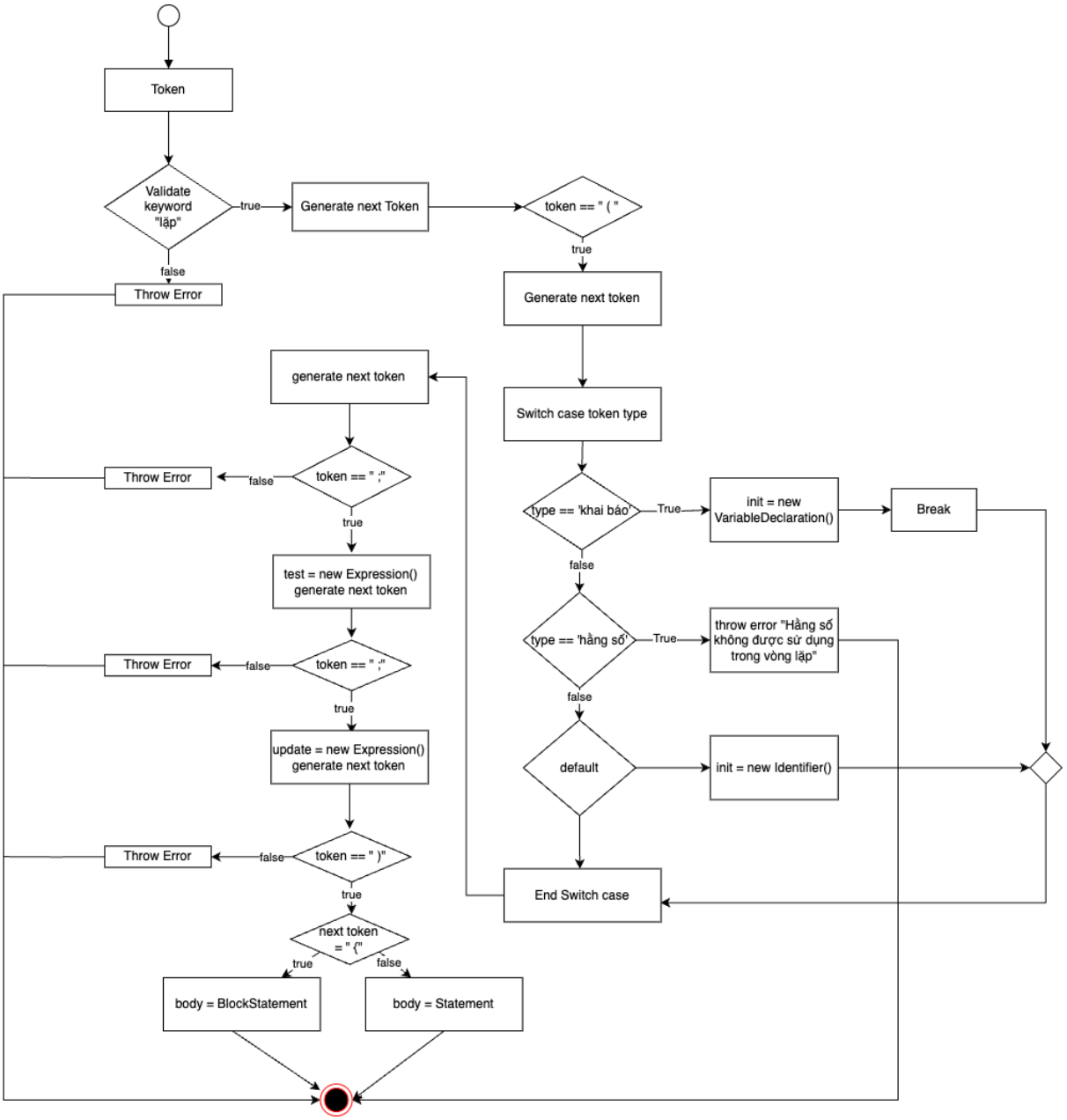
Ví dụ:

```
1    lặp (khai báo i = 0; i< 5; i ++){
2        in ra(i)
3    }
```

Hình 3.25. Ví dụ câu lệnh vòng lặp trong ngôn ngữ lập trình VieLang.

Luồng phân tích cú pháp vòng lặp for:

- Kiểm tra có pháp cú pháp "lặp" không. Nếu đúng bắt đầu phân tích cú pháp vòng lặp. Nếu sai báo lỗi và dừng chương trình
- Kiểm tra từ tố có phải dấu ngoặc đơn
- Xử lý các loại khai báo biến trong vòng lặp:
    - Nếu là từ khoá "khai báo" thì gán biến init là một VariableDeclaration (khai báo biến)
    - Nếu là từ khoá "hằng số" thì báo lỗi và dừng chương trình
    - Cuối cùng nếu không phải cả hai trường hợp trên thì gán init là một identifier
- Xác thực các phần còn lại, xác thực dấu ";" đầu tiên. nếu đúng gán biểu thức (Expression) vào biến test
- Xác thực dấu ";" thứ 2. Nếu đúng gán biến biểu thức (expression) vào biến update
- Kiểm tra từ tố tiếp theo có phải dấu đóng ngoặc đơn ")". Nếu đúng sinh ra từ tố tiếp theo. Nếu sai báo lỗi dừng chương trình
- Kiểm tra từ tố tiếp theo có phải dấu "{". Nếu đúng gán body là một BlockStatement, nếu sai gán body là một Statement.



Hình 3.26. Luồng hoạt động của quá trình phân tích cú pháp câu lệnh vòng lặp.

e) Câu lệnh duyệt (Switch Statement)

Được sử dụng để lựa chọn một trong nhiều khối mã để thực thi dựa trên giá trị của biểu thức được xác định.

Ví dụ:

```
duyệt(tuổi tác){
    trường hợp 20:
        in ra("bạn 20 tuổi")
    trường hợp 12:
        in ra("bạn 12 tuổi")
    mặc định:
        in ra("mặc định ")
}
```

Hình 3.24. Ví dụ về câu lệnh duyệt trong ngôn ngữ lập trình VieLang.

f) Câu lệnh khi mà (While statement)

Câu lệnh while trong Vielang được sử dụng để thực thi một khối mã nếu một điều kiện nhất định là "đúng", và tiếp tục thực thi khối mã đó cho đến khi điều kiện trở thành "sai".

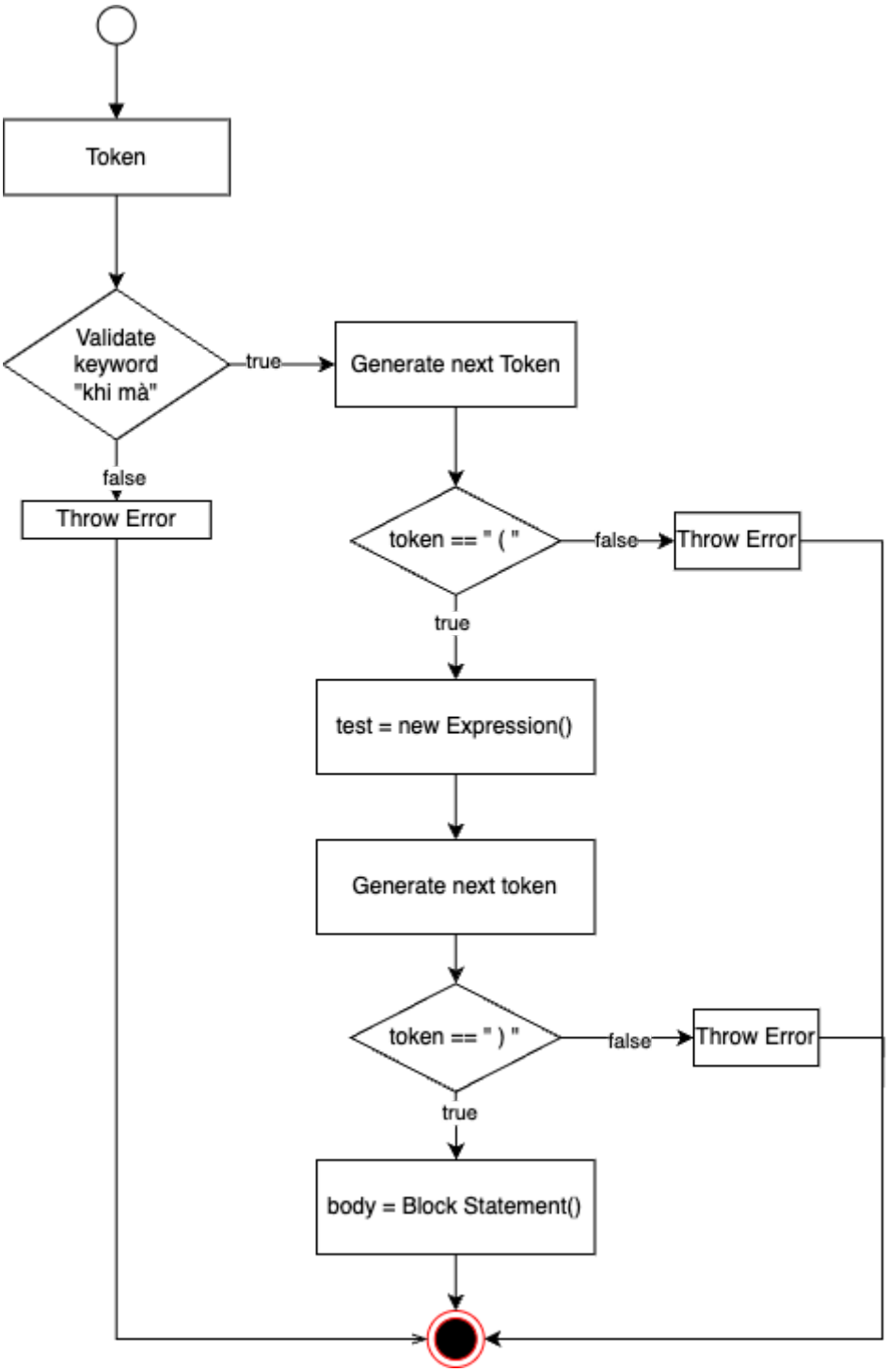
Ví dụ

```
1    khai báo a = 0;
2      khi mà (a < 4){
3          in ra (a)
4          a++
5    }
```

Hình 3.28. Ví dụ về câu lệnh vòng lặp khi mà trong ngôn ngữ lập trình VieLang.

Luồng phân tích cú pháp vòng lặp while:

- Xác thực từ khoá "khi mà". Nếu đúng bắt đầu phân tích cú pháp vòng lặp for
- Kiểm tra từ tố tiếp theo có phải dấu mở ngoặc đơn không "(". Nếu đúng gán test là một biểu thức (Expression). Nếu sai báo lỗi và dừng chương trình.
- Kiểm tra từ tố tiếp theo có phải đấu ngoặc đóng ")" không. Nếu đúng gán body là một Block Statement. Nếu sai báo lỗi và dừng chương trình.
- Kết thúc chương trình.



Hình 3.29. Luồng hoạt động của quá trình phân tích cú pháp câu lệnh vòng lặp khi mà.

g) Câu lệnh phá huỷ (Break Statement)

Được sử dụng để kết thúc một cấu trúc điều khiển vòng lặp hoặc một câu lệnh switch và chuyển quyền điều khiển tới câu lệnh ngay sau cấu trúc đó.

Ví dụ:

```
1   lặp (khai báo i = 0; i< 5; i ++){
2       nếu(i ==3){
3           phá huỷ
4       }
5   }
```

Hình 3.27. Ví dụ về câu lệnh phá huỷ

h) Câu lệnh biểu thức (Expression Statement)

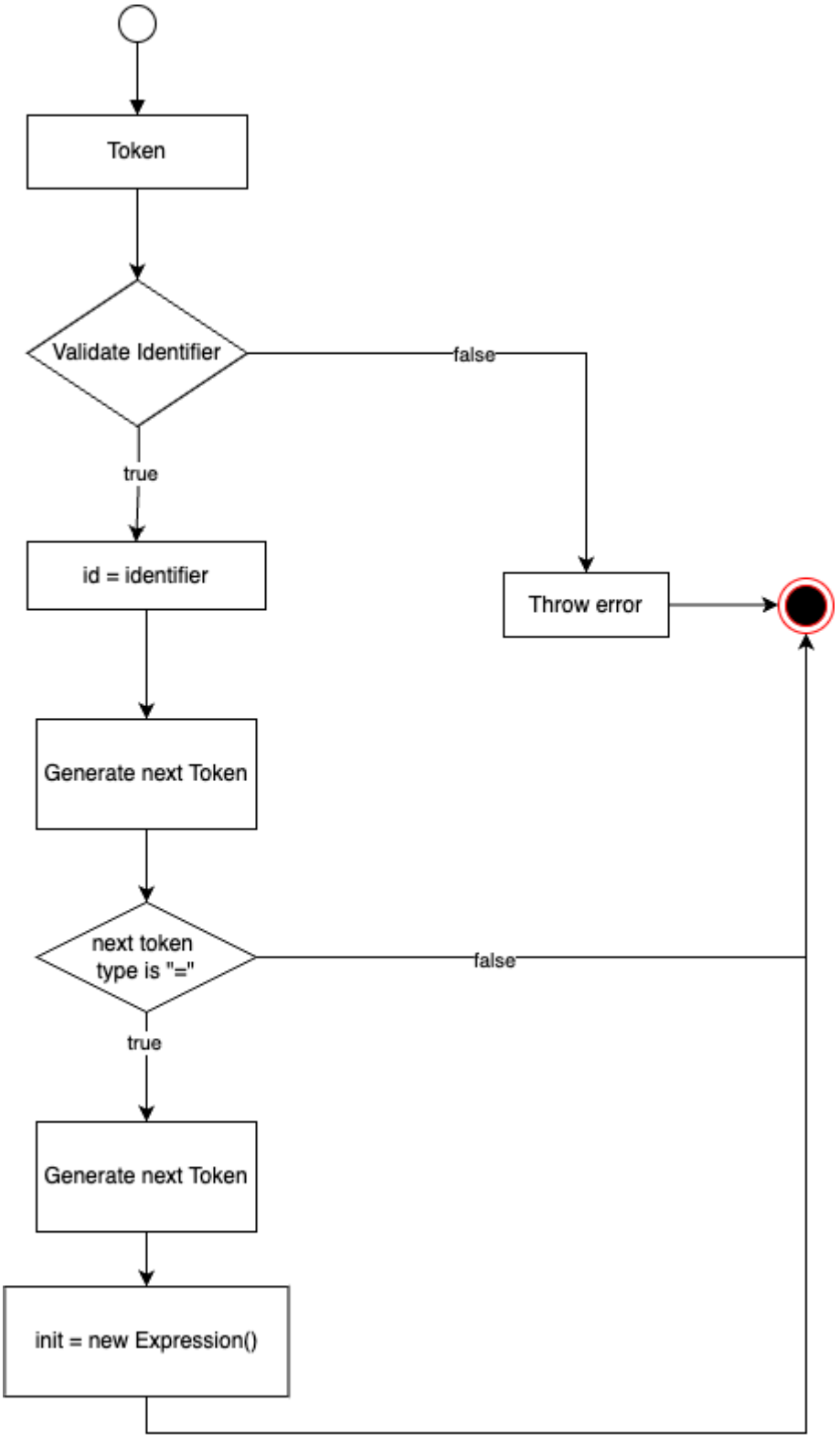
Gồm các biểu thức đã nêu ở phần **3.3.3**.

3.3.5. Khai báo biến (Variable Declaration)

a) Bộ khai báo (Variable Declarator)

Bước đầu tiên là xây dựng phân tích cú pháp cho bộ khai báo (Variable Declarator)

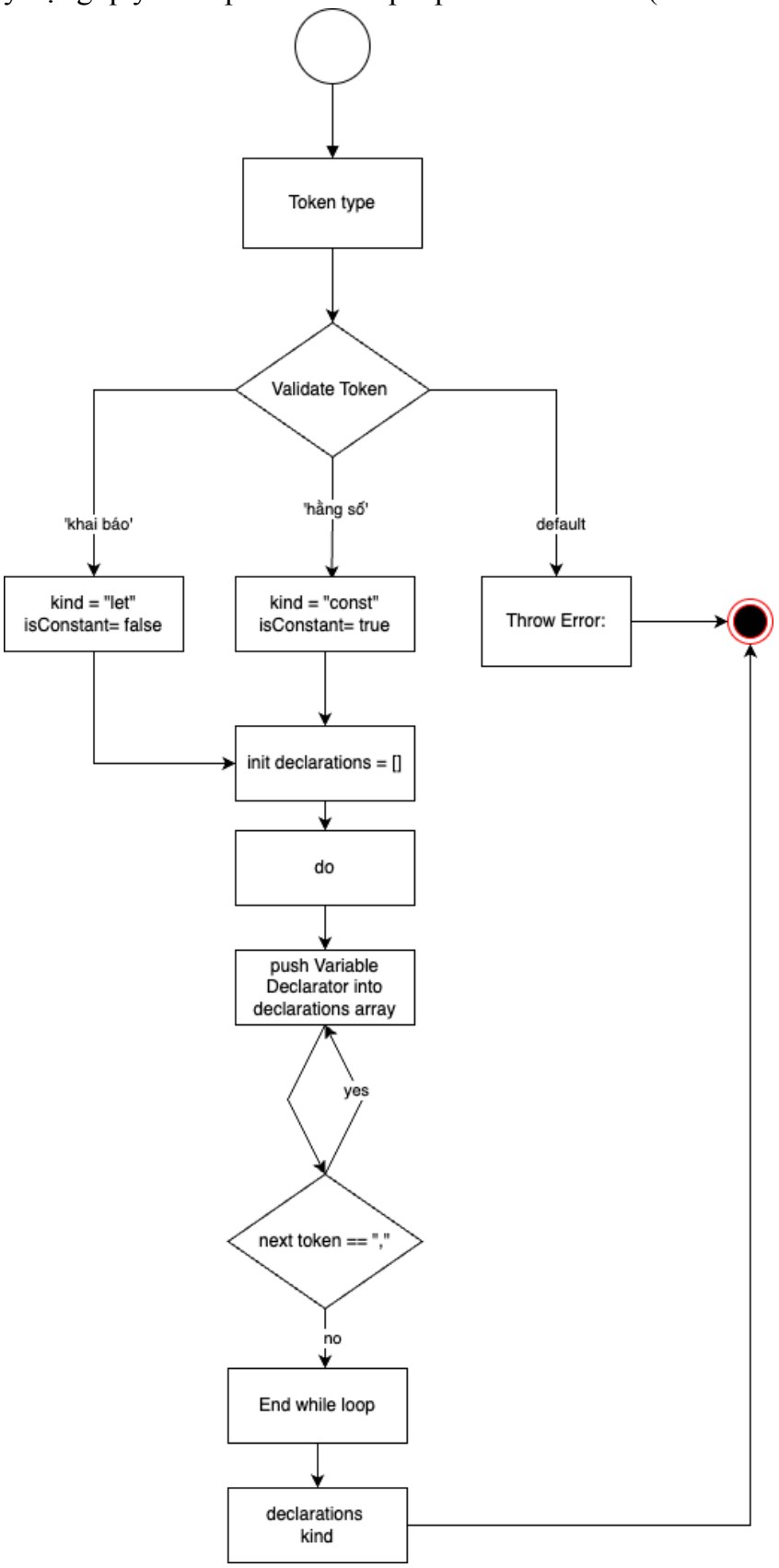
- Nhận vào đoạn mã VieLang, nếu phát hiện từ khoá “khai báo” thực hiện phân tích cú pháp khai báo biến
- Kiểm tra tên biến, nếu tên biến không hợp lệ báo lỗi, nếu hợp lệ tìm sinh từ tố tiếp theo đồng thời gán “ID” là tên biến vừa kiểm tra
- Kiểm tra từ tố vừa được sinh ra có phải kí tự “=”. Nếu sai thì kết thúc, nếu đúng sinh ra từ tố tiếp theo
- Gán init là 1 biểu thức (expression) và kết thúc



Hình 3.30. Luồng hoạt động của quá trình phân tích cú pháp bộ khai báo biến.

b) Khai báo biến (Variable Declaration)

Tiếp theo xây dựng quy trình phân tích cú pháp khai báo biến (Variable Declaration)



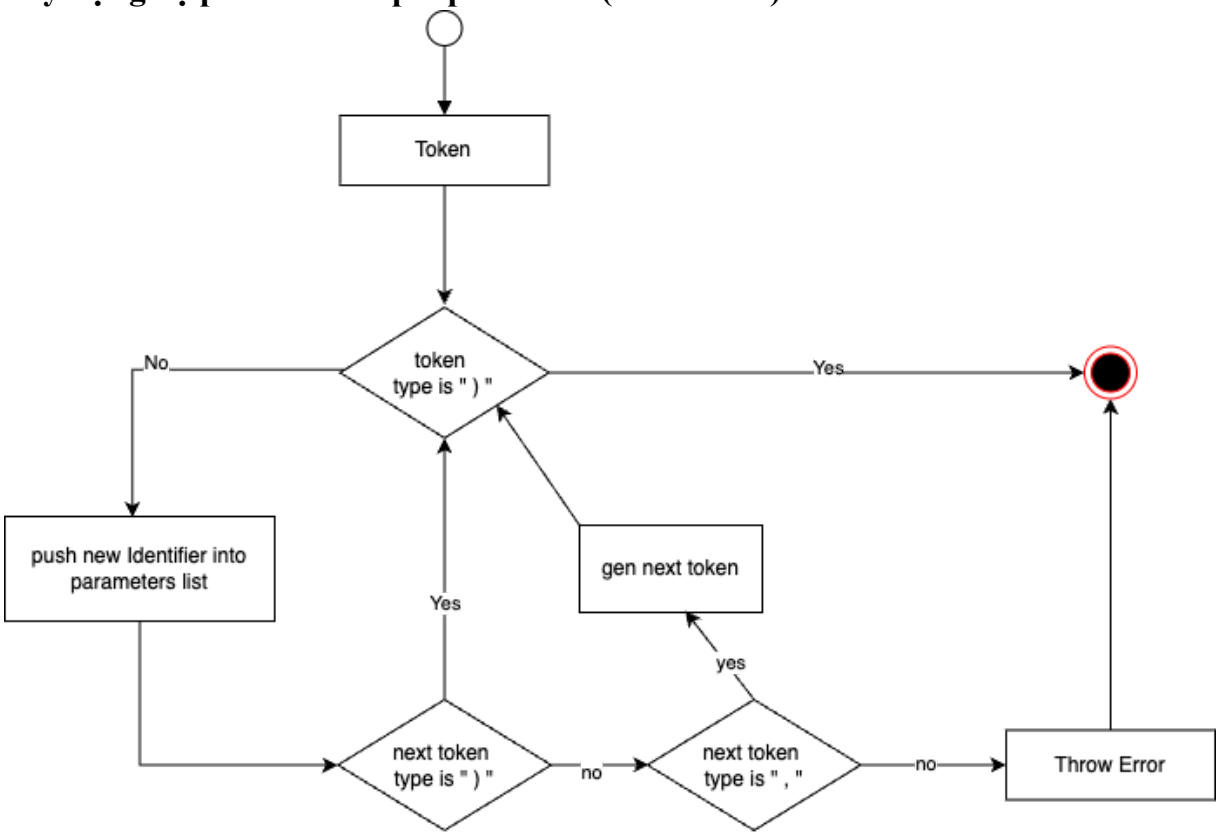
Hình 3.31. Luồng hoạt động của quá trình phân tích cú pháp khai báo biến

Luồng phân tích cú pháp khai báo biến

- Kiểm tra token, nếu từ tố là khai "khai báo", gán kind = 'let", hoặc nếu loại từ tố là "hằng số" gán kind = 'constant' và isContant = true. Nếu không: Ném lỗi SyntaxError
- Khởi tạo mảng declarations: Array<VariableDeclarator> = []
- Lặp lại (do-while)
  + Thêm đối tượng VariableDeclarator mới vào declarations
  + Kiểm tra điều kiện tiếp tục lặp: từ tố tiếp theo nếu là "," thì tiếp tục gán một VariableDeclarator vào mảng declarations. Nếu không kết thúc vòng lặp

3.3.6 Khai báo hàm (Function Declaration)

a) Xây dựng bộ phân tích cú pháp tham số (Parameter)



Hình 3.32. Luồng hoạt động của quá trình phân tích cú pháp tham số.

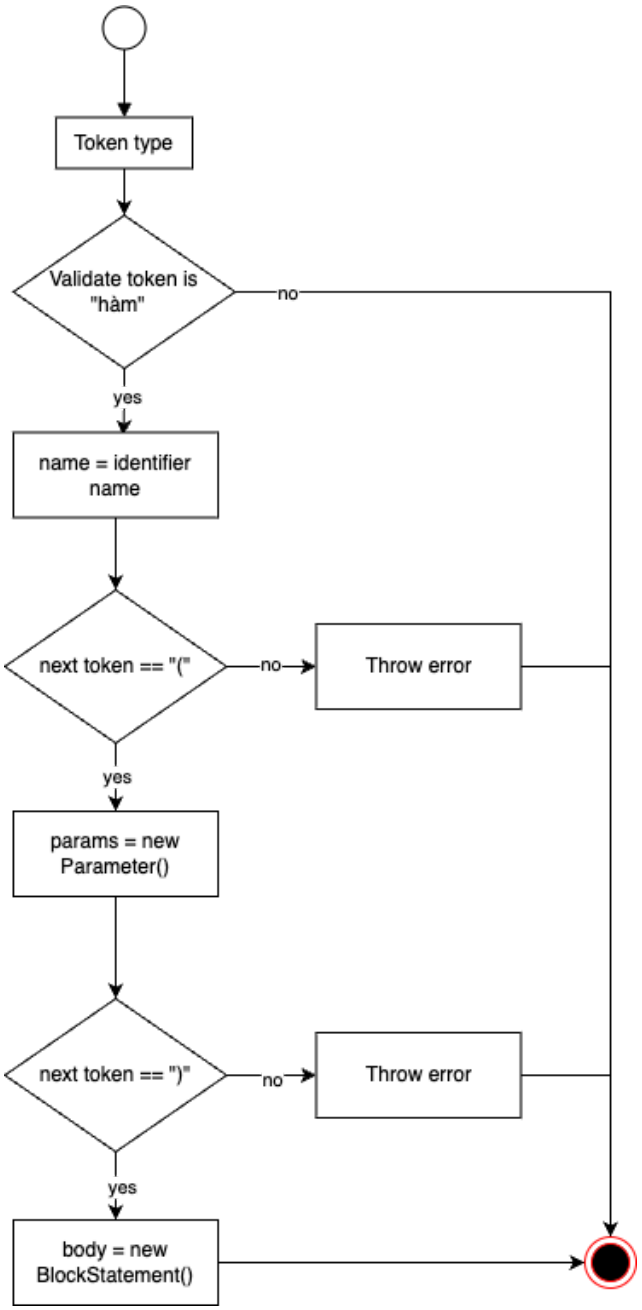
Luồng phân tích cú pháp tham số hàm:

- Khởi đầu vòng lặp while, kiểm tra từ tố có khác dấu ")" hay không Nếu đúng dừng vòng lặp. Nếu khác chuyển sang bước tiếp theo.
- Thêm các phần tử tham số vào mảng chứa các tham số
- Kiểm tra từ tố tiếp theo có phải ")" hay không, nếu không thì kiểm tra từ tố tiếp theo có phải dấu ",". Nếu đúng tiếp tục vòng lặp
- Kiểm tra từ tố có phải dấu phẩy "," không. Nếu đúng sinh ra từ tố tiếp theo và tiếp tục vòng lặp, nếu sai báo lỗi và dừng chương trình

b) Xây dựng bộ phân tích cú pháp khai báo hàm

Luồng phân tích cú pháp khai báo hàm:

- Xác thực từ khoá “hàm” (function). Đàm bảo rằng từ tố hiện tại là từ khoá “hàm” nếu không trình phân tích cú pháp ném ra lỗi
- Khởi tạo và gán tên hàm.
- Xác thực dấu mở ngoặc đơn “(“
- Khởi tạo và gán các tham số bằng trình phân tích cú pháp tham số (được mô tả ở trên)
- Xác thực dấu đóng ngoặc đơn “)”
- Khởi tạo và gán thân hàm
- Kết thúc quá trình phân tích cú pháp.



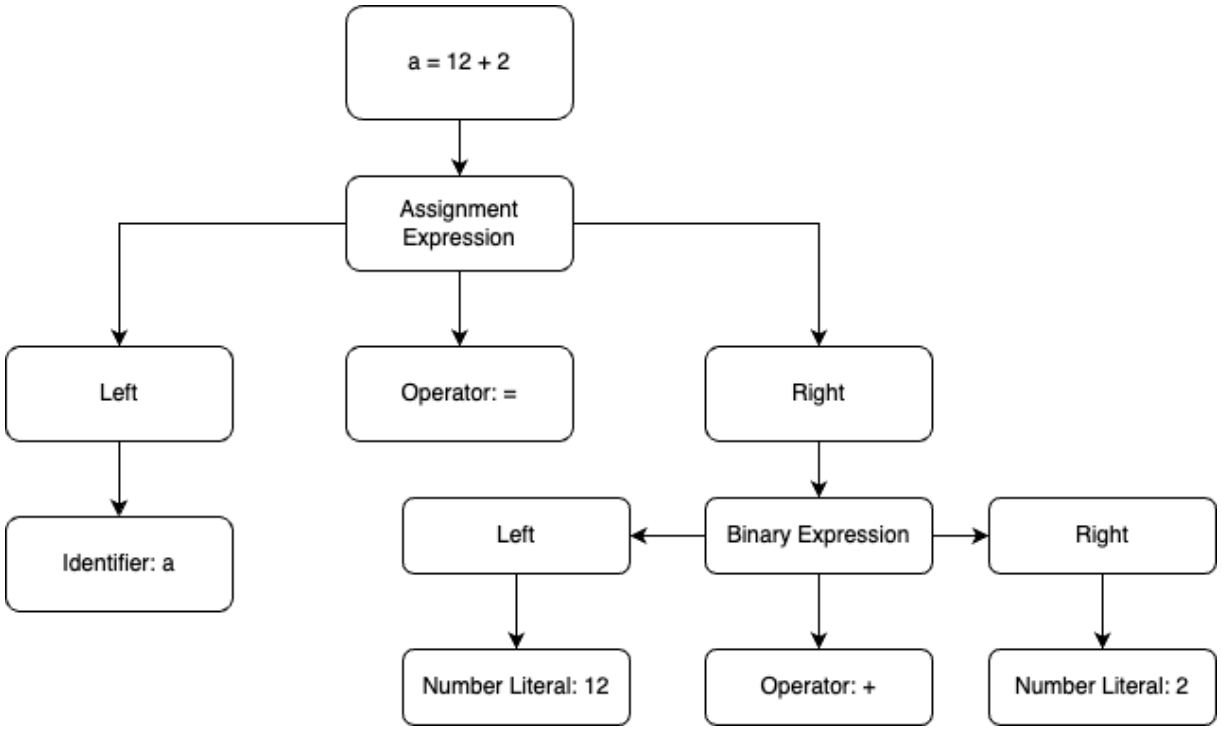
Hình 3.33. Luồng hoạt động của quá trình phân tích cú pháp khai báo hàm.

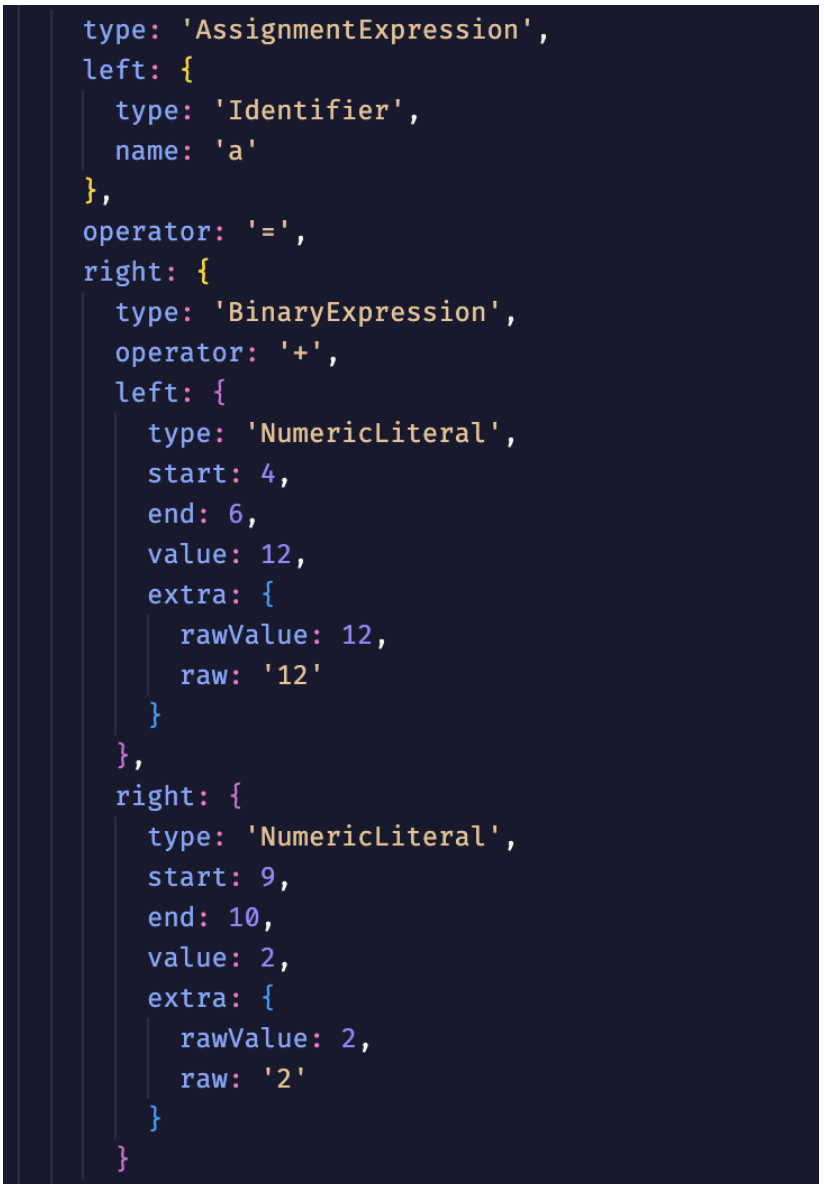
CHƯƠNG 4: HỆ SINH THÁI SẢN PHẨM CỦA NGÔN NGỮ LẬP TRÌNH VIELANG

4.1. Kết quả bộ phân tích cú pháp VieLang

4.1.1. Biểu thức gán (Assignment Expression)

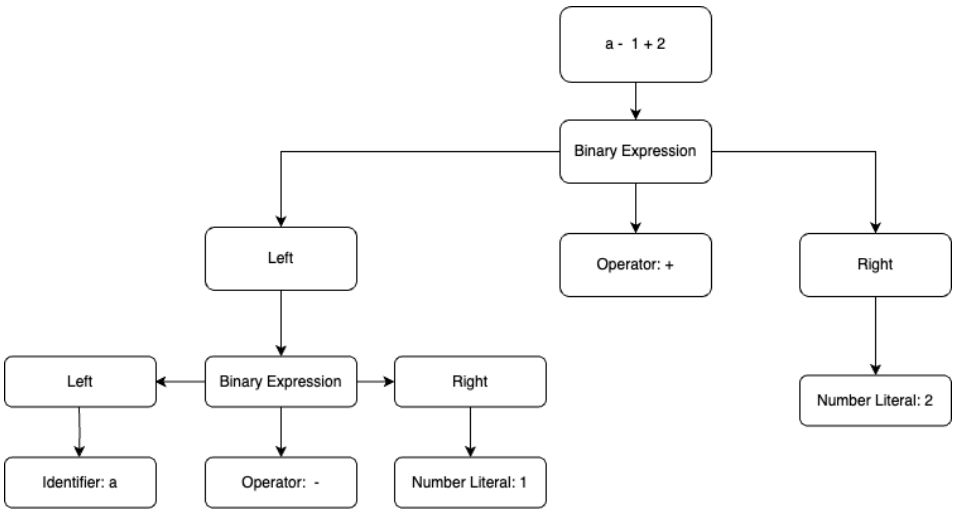
- Cây cú pháp trừu tượng của đoạn mã `a = 12 + 2` :




type: 'AssignmentExpression',
left: {
type: 'Identifier',
name: 'a'
},
operator: '=',
right: {
type: 'BinaryExpression',
operator: '+',
left: {
type: 'NumericLiteral',
start: 4,
end: 6,
value: 12,
extra: {
rawValue: 12,
raw: '12'
}
},
right: {
type: 'NumericLiteral',
start: 9,
end: 10,
value: 2,
extra: {
rawValue: 2,
raw: '2'
}
}

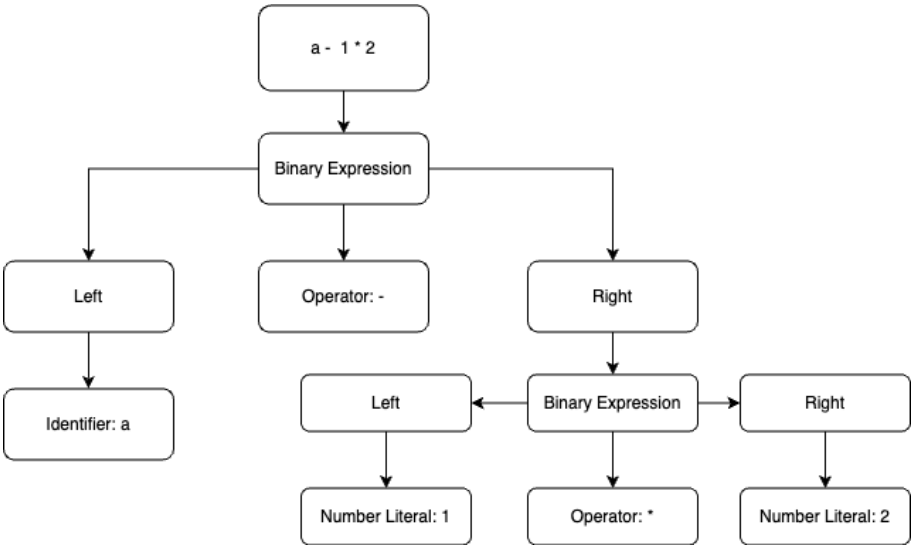
4.1.2. Biểu thức nhị phân (Binary Expressison)

Biểu thức: a - 1 + 2. Sẽ được phân tích từ trái sang phải như sau




```
type: 'BinaryExpression',
left: {
  type: 'BinaryExpression',
  operator: '-',
  left: {
    type: 'Identifier',
    name: 'a'
  },
  right: {
    type: 'NumericLiteral',
    start: 5,
    end: 6,
    value: 1,
    extra: {
      rawValue: 1,
      raw: '1'
    }
  }
},
operator: '+',
right: {
  type: 'NumericLiteral',
  start: 8,
  end: 9,
  value: 2,
  extra: {
    rawValue: 2,
    raw: '2'
  }
}
```

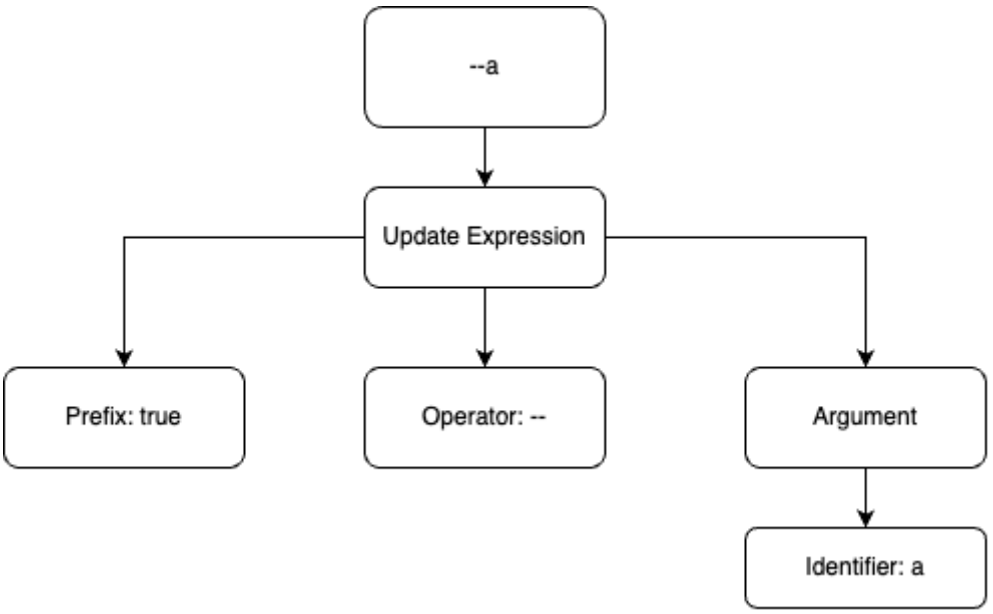
Biểu thức: a - 1 * 2. Sẽ được phân tích nhân chia trước cộng trừ sau. Từ đó có cây cú pháp như sau:




```
type: 'BinaryExpression',
left: {
  type: 'Identifier',
  name: 'a'
},
operator: '-',
right: {
  type: 'BinaryExpression',
  operator: '*',
  left: {
    type: 'NumericLiteral',
    start: 4,
    end: 5,
    value: 1,
    extra: {
      rawValue: 1,
      raw: '1'
    }
  },
  right: {
    type: 'NumericLiteral',
    start: 8,
    end: 9,
    value: 2,
    extra: {
      rawValue: 2,
      raw: '2'
    }
  }
}
```

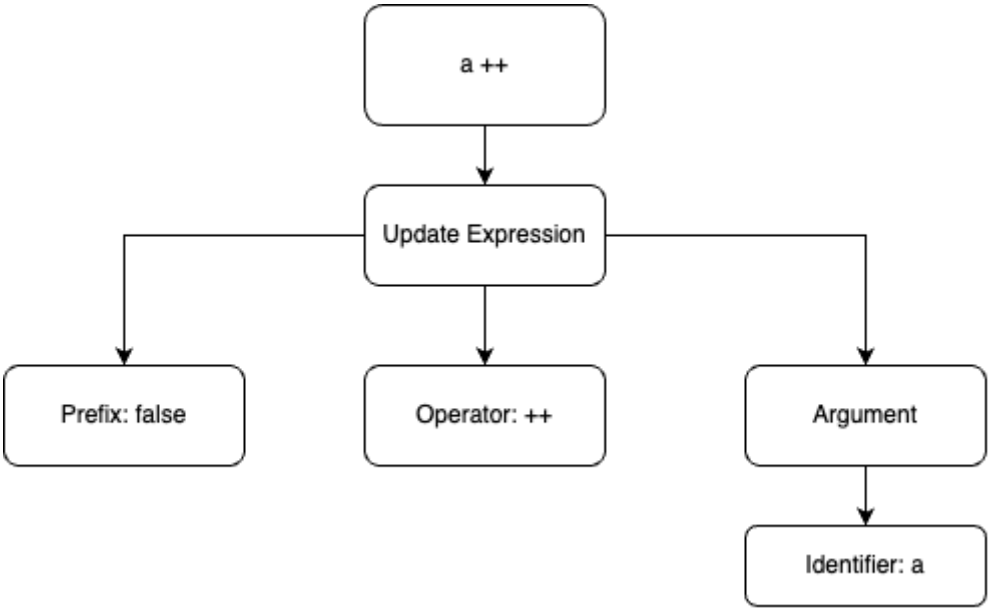
4.1.3 Biểu thức cập nhập (Update Expression)

- Prefix Update Expression



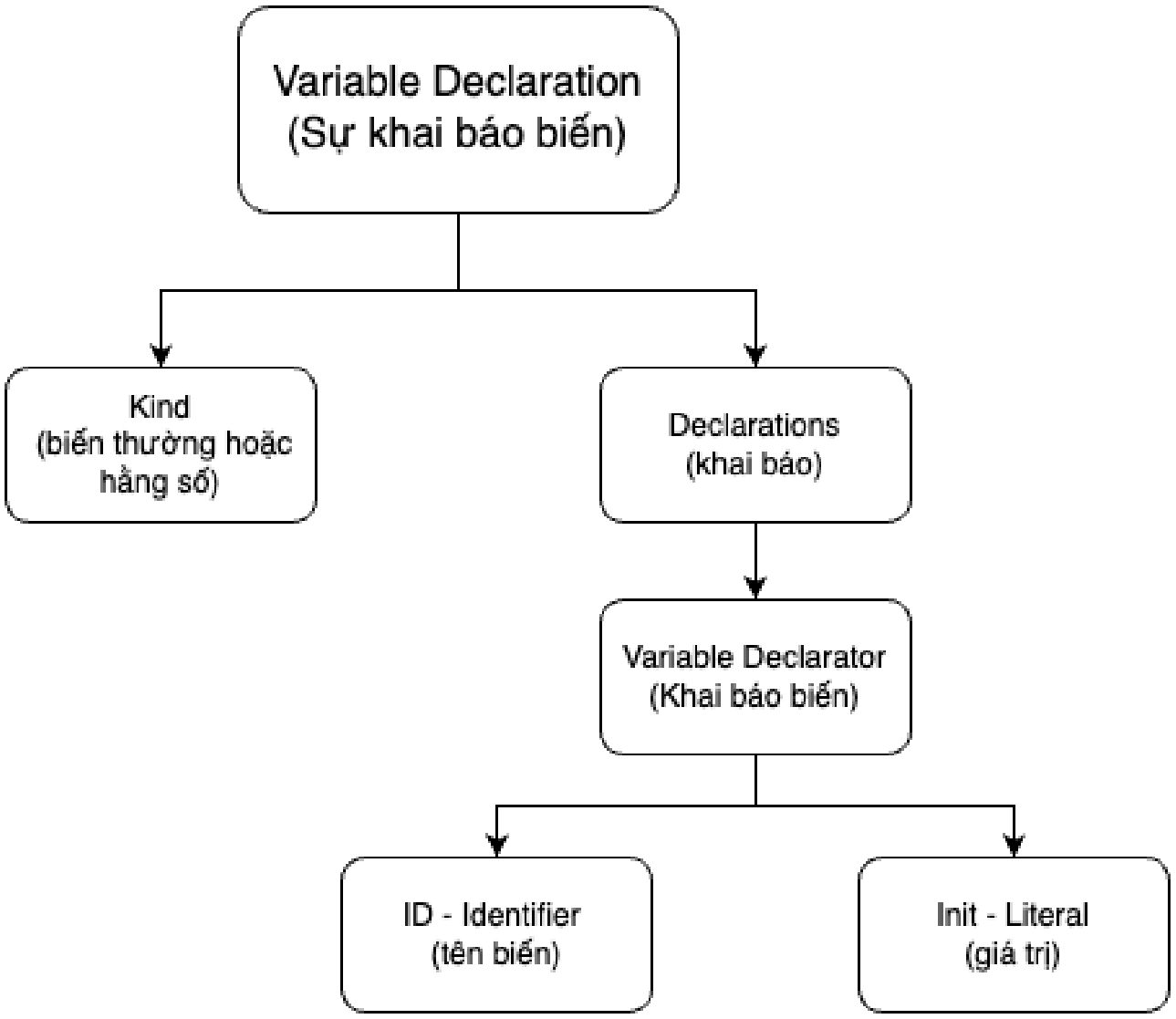

```
type: 'UpdateExpression',
argument: {
  type: 'Identifier',
  name: 'a'
},
prefix: true,
operator: '--'
```

- Suffix Update Expression

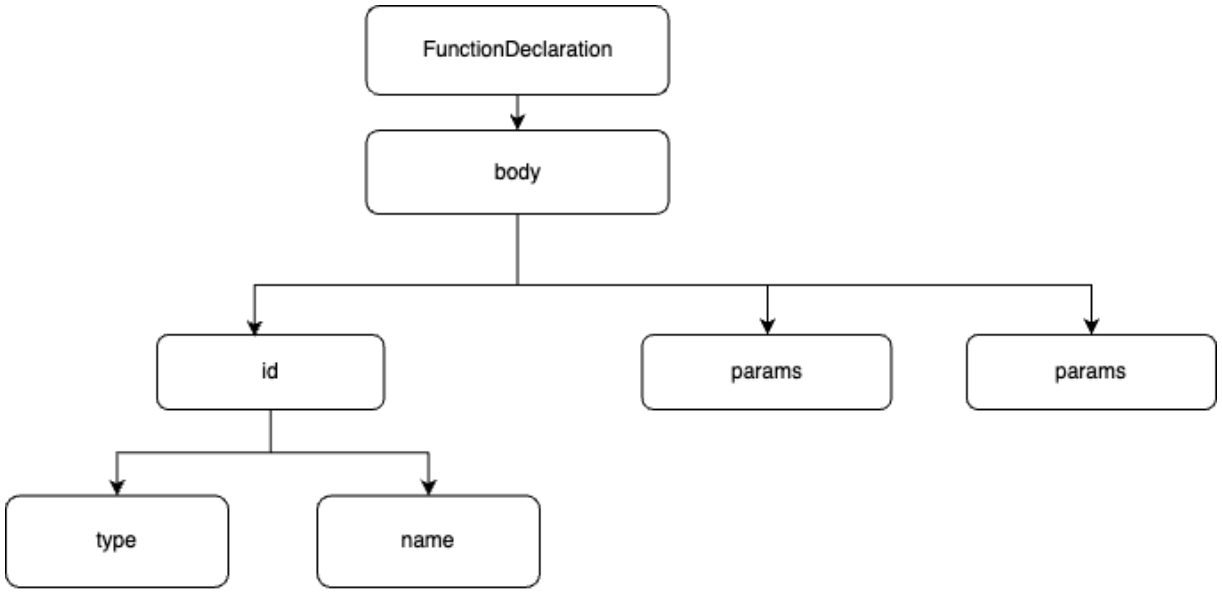



```
type: 'UpdateExpression',
argument: {
  type: 'Identifier',
  name: 'a'
},
prefix: false,
operator: '++'
```

4.1.4 Khai báo biến (Variable Declaration)



4.1.5 Khai báo hàm (Function Declaration)



Dưới đây là cây cú pháp của hàm sau:

```
1    hàm kiểm tra(a){
2        return a + 4
3    }
```


type: 'Program',
body: [
{
type: 'FunctionDeclaration',
async: false,
id: {
type: 'Identifier',
name: 'ki_7875m_tra'
},
params: [
{
type: 'Identifier',
name: 'a'
}
],
body: {
type: 'BlockStatement',
body: [
{
type: 'ReturnStatement',
argument: {
type: 'BinaryExpression',
left: {
type: 'Identifier',
name: 'a'
},
operator: '+',
right: {
type: 'NumericLiteral',
value: 4,
extra: {
rawValue: 4,
raw: '4'
}
}
}
}
]
}
}
]

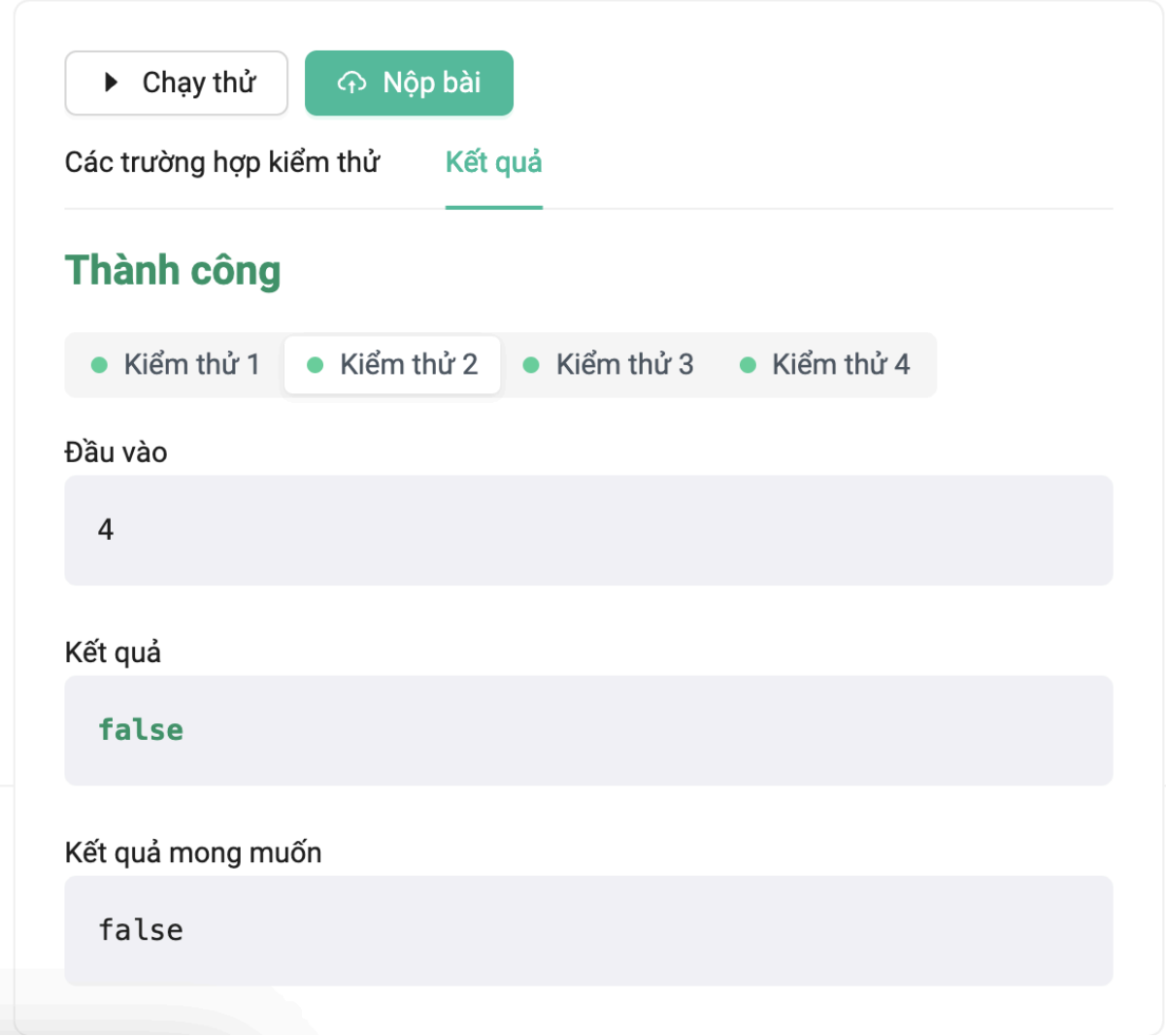
4.2. Trang web playground

Trang web playground được xây dựng bằng React.js để tạo ra một trình biên dịch mã ngay trên browser. Đây là một công cụ hữu ích đáp ứng yêu cầu của đề tài, trực quan hóa các đoạn mã để dễ dàng cho học sinh sử dụng. Giao diện của trang web được thiết kế đơn giản và trực quan. Học sinh có thể nhập đoạn mã Vielang của mình vào bên phải, sau đó nhấn nút "Thực thi" để thực thi đoạn mã Vielang

Bên cạnh đó, trang web đã tích hợp sẵn các ví dụ bài tập khác nhau, chỉ bằng cách click chuột vào nút danh sách bài tập để xem danh sách

Hình 4.8. Giao diện trang web playground

Làm bài tập trực tiếp trên giao diện với bộ kiểm thử cho từng bài



Hình 4.9. Giao diện bộ kiểm thử trang web playground

Trang web Vielang đã được deploy bằng Vercel (https://vielang.vienhuynh.dev). Vercel là một nền tảng phổ biến cho việc triển khai ứng dụng web và cung cấp một quy trình triển khai đơn giản và dễ dùng. Vercel cung cấp một hệ thống mạnh mẽ, đảm bảo trang web được tải nhanh và có hiệu suất cao trên các vị trí truy cập từ khắp nơi trên thế giới. Việc deploy trang web bằng Vercel giúp đảm bảo rằng trang web luôn hoạt động ổn định và có thể truy cập một cách thuận tiện cho người dùng, cung cấp trải nghiệm tốt nhất khi sử dụng ứng dụng này.

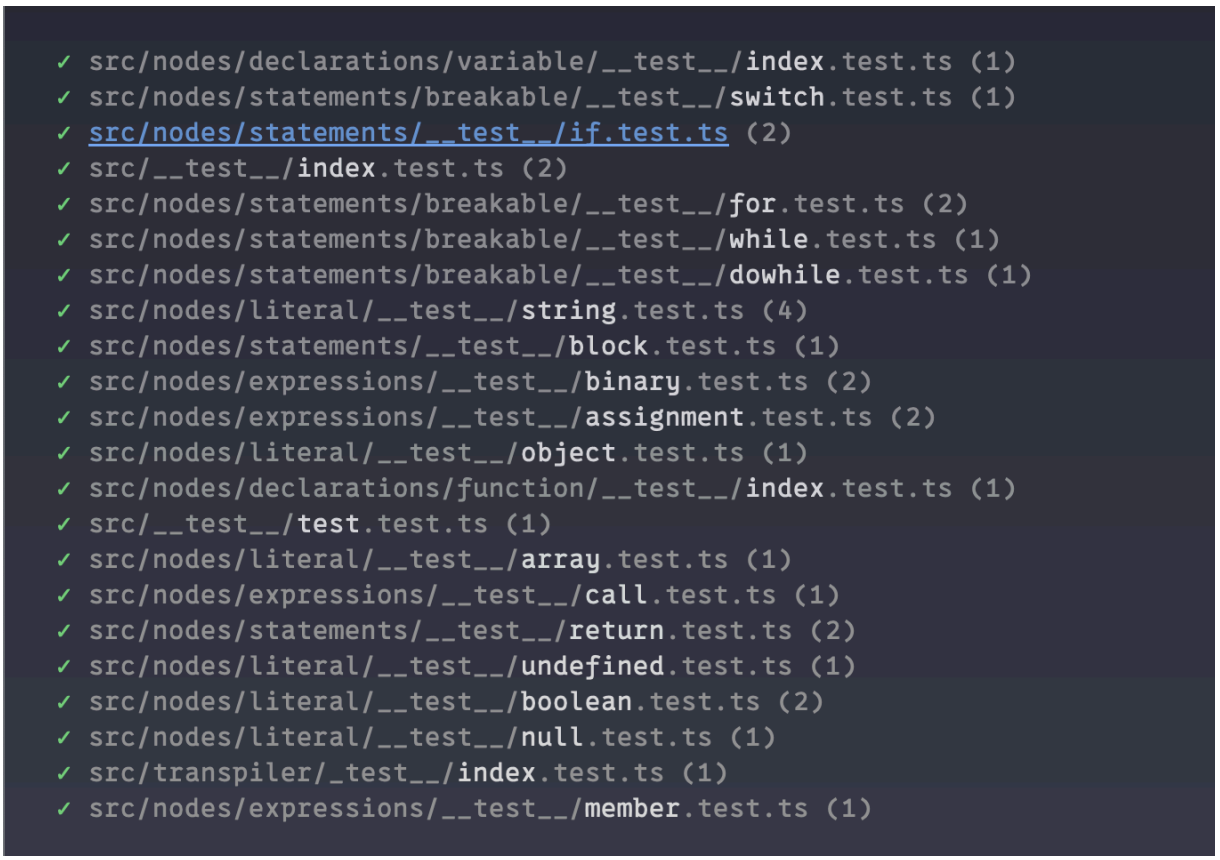
Tổng quan, Vielang Playground là một công cụ hữu ích học sinh làm bài tập bằng ngôn ngữ VieLang. Với giao diện đơn giản, những ví dụ demo trực tiếp và tính năng thực thi mã ngay trên trình duyệt, giúp học sinh dễ dàng tiếp cận.

Chương 5: THỬ NGHIỆM VÀ ĐÁNH GIÁ

5.1. Thử nghiệm

5.1.1. Bộ dữ liệu kiểm tra

Quá trình phát triển theo phương pháp TDD, do đó mỗi phần đều có test case. Trong quá trình phát triển tính năng mới, đảm bảo các test case cũ vẫn được chạy thành công



Hình 5.1. Bộ kiểm thử cho hệ thống phân tích cú pháp

5.1.2. Quá trình thử nghiệm

Thực hiện các bước sau để thực nghiệm hệ thống ngôn ngữ lập trình tiếng Việt - Vielang:

1. Định nghĩa bộ test-case cho từng phần. Vì viết mã theo phương pháp TDD. Do đó trước khi viết mã, sẽ định nghĩa bộ test- case mong muốn
2. Chạy bộ test case, đảm bảo tất cả các test-case đều thực thi đúng
3. Kiểm tra kết quả cây cú pháp được tạo ra và đánh giá tính chính xác.
4. Tiếp tục thực hiện các vòng lặp của quá trình thử nghiệm để cải thiện tính chính xác của bộ phân tích. Điều chỉnh các tham số để đạt được sự phù hợp và chính xác tốt hơn.

5.2. Đánh giá kết quả

5.2.1. Phương pháp đánh giá

Đánh giá tự động: Sử dụng bộ test-case đảm bảo tính ổn định và đúng đắn của hệ thống

Đánh giá tính khả thi: Xem xét tính khả thi của ngôn ngữ lập trình tiếng Việt trong môi trường thực tế. Điều này bao gồm việc kiểm tra ngôn ngữ có chứa các lỗi cú pháp hay lỗi logic không, có tuân thủ các quy tắc và ràng buộc của ngôn ngữ không, và có thể triển khai và chạy được hay không.

Kiểm tra trên tập dữ liệu: Kiểm tra bằng các bài toán cơ bản, ví dụ như tính giai thừa, sắp xếp, …

Đánh giá độ phù hợp: Đánh giá độ phù hợp của ngôn ngữ lập trình tiếng Việt trong việc giảng dạy. Xem xét tính rõ ràng, dễ hiểu và hợp lý của sơ đồ khối, có thể giúp học sinh dễ dàng tiếp cận, viết mã.

5.2.2. Kết quả đánh giá

a) Điểm mạnh:

Giúp người mới học lập trình, đặc biệt là học sinh và người không biết tiếng Anh, dễ dàng hơn trong việc tiếp cận các khái niệm cơ bản.

Tạo động lực cho việc học lập trình bằng cách giảm thiểu khó khăn về ngôn ngữ.

Tính khả thi: Ngôn ngữ lập trình tiếng Việt là khả thi để triển khai và chạy môi trường thực tế trong quá trình giảng dạy . Hệ thống kiểm tra và đảm bảo rằng ngôn ngữ tuân thủ các quy tắc ngôn ngữ và không chứa lỗi cú pháp hay lỗi logic.

b) Hạn chế:

- Quá trình phổ biến rộng rãi trong cộng đồng lập trình và áp dụng việc dạy học sẽ tốn nhiều thời gian, vì đây là môi trường giáo dục nên cần đảm bảo chất lượng
- Hiệu suất có thể không bằng các ngôn ngữ lập trình phổ biến hiện nay do sự tối ưu chưa cao.

KẾT LUẬN

Ngôn ngữ lập trình tiếng Việt là một sáng kiến đáng ghi nhận, giúp tăng cường khả năng tiếp cận lập trình cho người Việt Nam, đặc biệt là những người gặp khó khăn về ngôn ngữ. Tuy nhiên, để phát triển và mở rộng hơn nữa, cần có sự đầu tư về tài nguyên, tài liệu học tập, và sự hỗ trợ từ cộng đồng lập trình viên. Đồng thời, cần phải cải thiện hiệu suất và khả năng tương thích với các công cụ lập trình hiện đại.

Đề xuất cụ thể:

a) Phát triển thêm tài liệu học tập và hướng dẫn chi tiết:
- Biên soạn các giáo trình, video hướng dẫn, và các bài tập thực hành cụ thể.
- Xây dựng một trang web hoặc diễn đàn chính thức để người học có thể trao đổi và nhận hỗ trợ.

b) Tăng cường quảng bá và tổ chức các sự kiện lập trình:
- Tổ chức các cuộc thi lập trình sử dụng ngôn ngữ tiếng Việt để thu hút sự quan tâm của cộng đồng.
- Hợp tác với các trường học để đưa ngôn ngữ này vào chương trình giảng dạy thử nghiệm.

c) Cải thiện khả năng tương thích và tích hợp:
- Phát triển các thư viện và framework hỗ trợ.
- Tích hợp với các công cụ lập trình phổ biến như Visual Studio Code, GitHub.

Việc thực hiện những đề xuất trên có thể giúp ngôn ngữ lập trình tiếng Việt trở nên phổ biến và hữu ích hơn, đồng thời tạo điều kiện thuận lợi cho người Việt Nam tiếp cận và thành thạo lập trình.

TÀI LIỆU THAM KHẢO