**Thành Phần Của Trình Biên Dịch**

**1. Mã Nguồn (Source Code)**

Mã nguồn là chương trình được viết bằng ngôn ngữ lập trình của bạn. Đây là điểm bắt đầu của quá trình biên dịch, chứa các lệnh và cấu trúc mà con người có thể đọc được nhưng máy tính chưa thể hiểu trực tiếp.

**2. Phân Tích Từ Vựng (Lexer/Tokenizer)**

Đây là giai đoạn đầu tiên của quá trình biên dịch. Lexer đọc mã nguồn và chuyển đổi nó thành chuỗi các token (thẻ) [citation:1]. Mỗi token đại diện cho một phần tử ngôn ngữ như từ khóa, định danh, toán tử, hoặc dấu ngoặc. Quá trình này loại bỏ khoảng trắng, comment và định dạng không cần thiết.

Ví dụ: Câu lệnh int x = 5; sẽ được chia thành các token:

* int (từ khóa)
* x (định danh)
* = (toán tử gán)
* 5 (hằng số)
* ; (dấu chấm phẩy)

**3. Phân Tích Cú Pháp (Parser)**

Parser nhận chuỗi các token và xác định cấu trúc ngữ pháp của chương trình theo quy tắc của ngôn ngữ [citation:1]. Nó kiểm tra xem các token có được sắp xếp theo cách hợp lệ hay không. Phân tích cú pháp dựa trên các quy tắc ngôn ngữ lập trình cụ thể bằng cách xây dựng cây phân tích cú pháp [citation:1].

Các phương pháp phân tích phổ biến:

* Phân tích từ trên xuống (Top-down parsing)
* Phân tích từ dưới lên (Bottom-up parsing)

**4. Cây Cú Pháp Trừu Tượng (AST - Abstract Syntax Tree)**

AST là biểu diễn cấu trúc phân cấp của mã nguồn, loại bỏ các chi tiết cú pháp không cần thiết. Nó cho thấy mối quan hệ giữa các phần tử trong chương trình và giúp biên dịch viên phân tích và biến đổi mã hiệu quả hơn.

Ví dụ: Biểu thức a = b + c \* d sẽ tạo ra một cây với nút gốc là toán tử gán, nhánh phải là biểu thức b + c \* d được biểu diễn dưới dạng cây con.

**5. Phân Tích Ngữ Nghĩa (Semantic Analysis)**

Giai đoạn này kiểm tra ý nghĩa của mã nguồn [citation:1]. Bao gồm:

* Kiểm tra kiểu dữ liệu
* Kiểm tra phạm vi biến
* Kiểm tra tính hợp lệ của các lệnh và biểu thức
* Kiểm tra tính nhất quán của chương trình

Ví dụ: Phát hiện lỗi khi gán giá trị chuỗi cho biến số nguyên.

Int a = ngu;

**6. Bảng Ký Hiệu (Symbol Table)**

Bảng ký hiệu là cấu trúc dữ liệu lưu trữ thông tin về biến, hàm, lớp và các định danh khác trong chương trình. Thông tin bao gồm:

* Tên định danh
* Kiểu dữ liệu
* Phạm vi
* Địa chỉ bộ nhớ (nếu có)

Trình biên dịch sử dụng bảng này trong quá trình phân tích ngữ nghĩa và tạo mã.

**7. Phương Pháp 1: Chuyển Đổi Mã (Transpiler)**

Transpiler chuyển đổi mã nguồn từ ngôn ngữ của bạn sang mã C++. Lợi ích:

* Tận dụng bộ biên dịch C++ đã tối ưu hóa (như GCC, Clang)
* Đơn giản hóa quá trình phát triển ngôn ngữ mới
* Khả năng tương thích với thư viện C++ hiện có

Sau khi chuyển đổi, mã C++ được biên dịch bằng GCC/Clang để tạo tệp thực thi.

**8. Phương Pháp 2: Biên Dịch Đầy Đủ với LLVM**

LLVM (Low Level Virtual Machine) là nền tảng biên dịch mạnh mẽ. Quá trình:

* Chuyển đổi AST thành LLVM IR (Intermediate Representation)
* LLVM IR là biểu diễn trung gian độc lập với ngôn ngữ và nền tảng
* Cho phép tối ưu hóa mã ở nhiều cấp độ

LLVM cung cấp nhiều công cụ để xây dựng trình biên dịch hiệu quả.

**9. Tối Ưu Hóa Mã (Optimize Code)**

Giai đoạn này cải thiện hiệu suất của mã trung gian mà không thay đổi hành vi của chương trình. Bao gồm:

* Loại bỏ mã không sử dụng
* Trải rộng vòng lặp
* Tối ưu hóa biểu thức
* Tối ưu hóa bộ nhớ
* Tối ưu hóa đăng ký

LLVM cung cấp nhiều mức tối ưu hóa từ -O1 đến -O3.

**10. Tạo Mã Máy (Machine Code Generation)**

Chuyển đổi mã tối ưu hóa thành mã máy dành riêng cho kiến trúc phần cứng mục tiêu (như x86, ARM, RISC-V). Quá trình này tạo ra mã nhị phân mà CPU có thể thực thi trực tiếp.

**11. Liên Kết (Linking)**

Liên kết kết hợp các tệp đối tượng (object files) và thư viện cần thiết để tạo thành tệp thực thi. Quá trình này:

* Giải quyết các tham chiếu bên ngoài
* Kết hợp mã từ nhiều tệp nguồn
* Kết hợp với thư viện hệ thống và thư viện người dùng

**12. Tệp Thực Thi (Executable File)**

Đây là kết quả cuối cùng của quá trình biên dịch. Tệp thực thi chứa mã máy có thể chạy trực tiếp trên hệ điều hành mục tiêu. Định dạng tệp thực thi phụ thuộc vào hệ điều hành:

* Windows: .exe
* Linux/Unix: ELF (không có phần mở rộng)
* macOS: Mach-O

