ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG VIỆT - HÀN
Vietnam - Korea University of Information and Communication Technology

Chương 1

Nhập môn chương trình dịch

Ngôn ngữ lập trình (Programming Language)

- Con người giao tiếp với nhau thông qua ngôn ngữ.
- Máy tính điện tử ra đời xuất phát từ nhu cầu thực tế trong cuộc sống: ***cần có một công cụ nào đó để trợ giúp con người xử lý thông tin.***

- Con người yêu cầu máy tính điện tử hỗ trợ giải quyết công việc.
- Như vậy máy tính điện tử phải hiểu được yêu cầu của con người.

Ngôn ngữ lập trình (Programming Language)

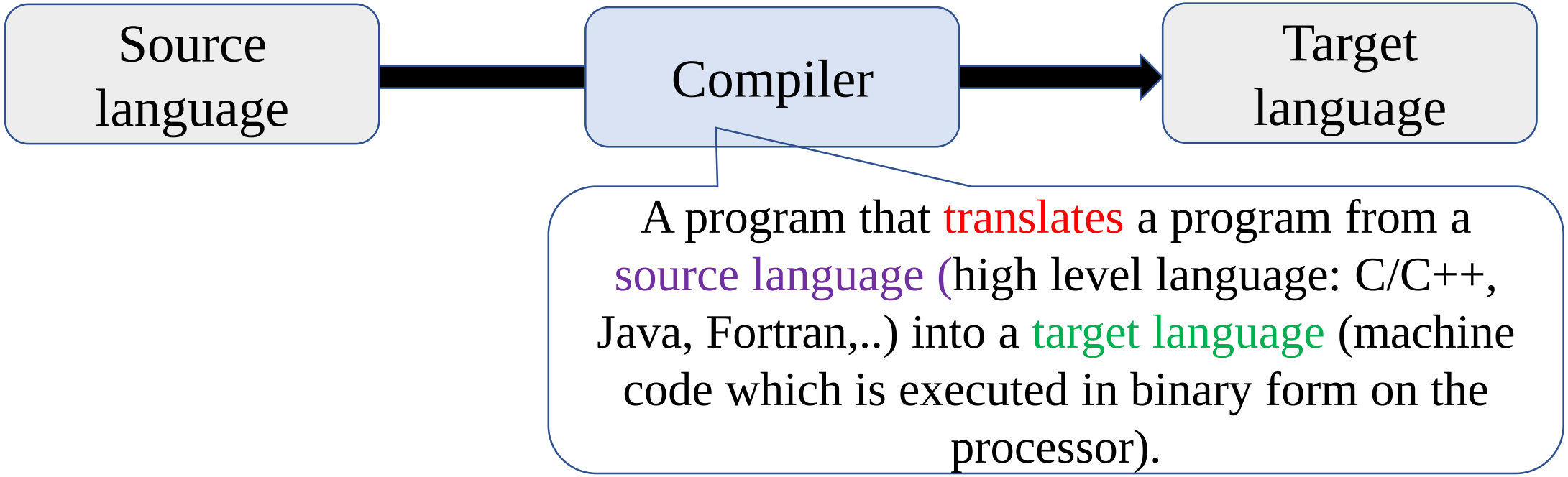
- Ngôn ngữ lập trình ra đời
- Con người sử dụng ngôn ngữ lập trình và thực hiện lập trình để tạo ra chương trình, cung cấp chương trình cho máy tính điện tử (thể hiện yêu cầu giải quyết công việc cho máy tính điện tử hiểu)
- Con người giao tiếp với máy tính điện tử cũng thông qua ngôn ngữ
- Phân loại ngôn ngữ lập trình:
  - **Ngôn ngữ máy (machine language)**: gắn chặt với phần cứng, xa lạ ngôn ngữ tự nhiên, xử lý trực tiếp bộ vi xư lý
  - **Hợp ngữ (assembly language)**: bớt gắn chặt phần cứng, sử dụng một số từ đẽ nhớ (bước đầu mô phỏng theo ngôn ngữ tự nhiên)
  - **Ngôn ngữ bậc cao (high-level language)**: gần gũi với ngôn ngữ tự nhiên

Ngôn ngữ lập trình (Programming Language)

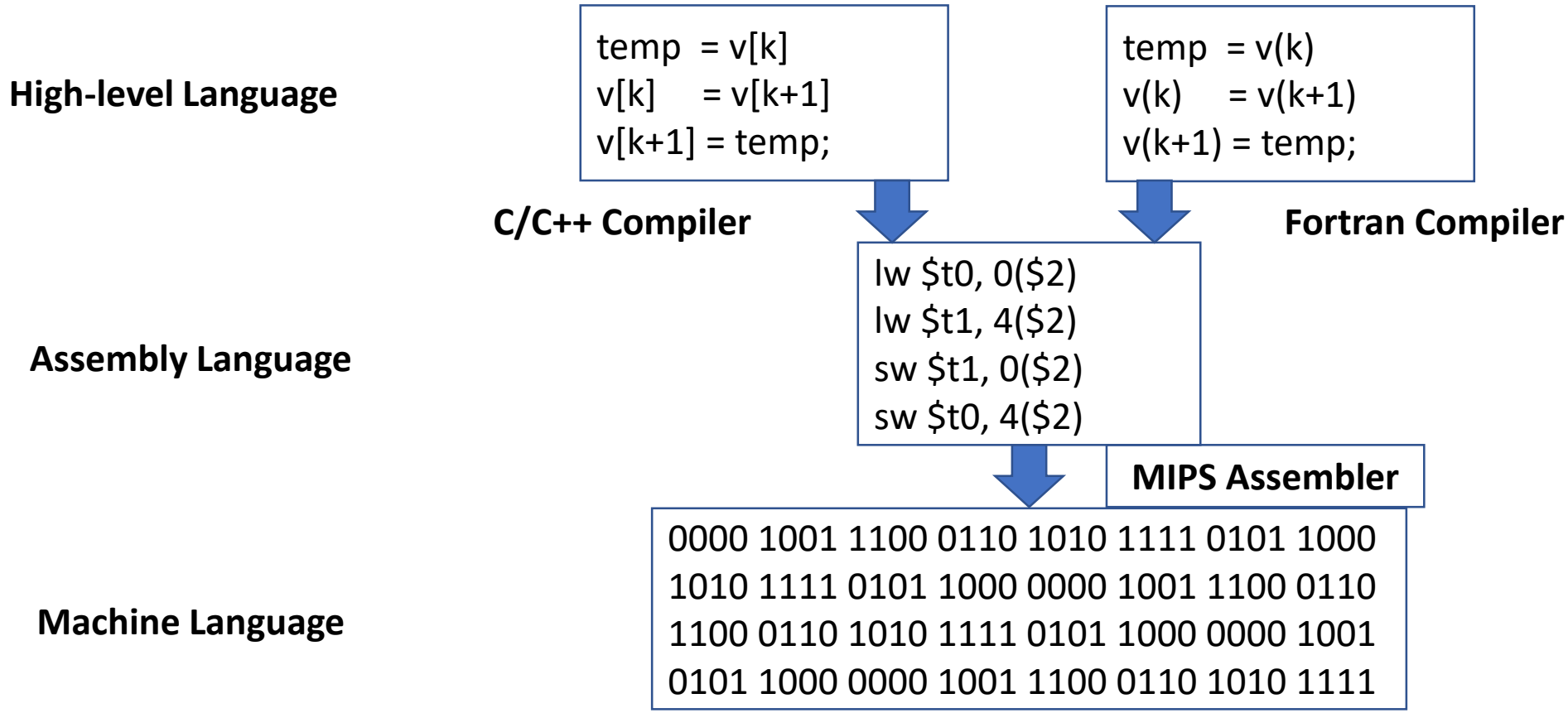
- Ngôn ngữ lập trình bậc cao:
  - Không yêu cầu người lập trình phải có hiểu biết sâu về cấu trúc của một máy tính cụ thể.
  - Độc lập với một máy tính cụ thể, dễ sử dụng để tạo ra chương trình.
  - Mã nguồn được viết bằng ngôn ngữ lập trình bậc cao có thể được dịch thành ngôn ngữ máy và chương trình có thể thực thi (chạy) trên nhiều loại máy tính khác nhau.
  - Gần gũi ngôn ngữ tự nhiên nên ngôn ngữ lập trình bậc cao có thể mô tả vấn đề một cách tự nhiên, dễ hiểu
- Tạo được nhiều chương trình nguồn và các công việc phức tạp, đa dạng của con người được giải quyết bằng máy tính điện tử ngày càng nhiều.

Ngôn ngữ lập trình (Programming Language)

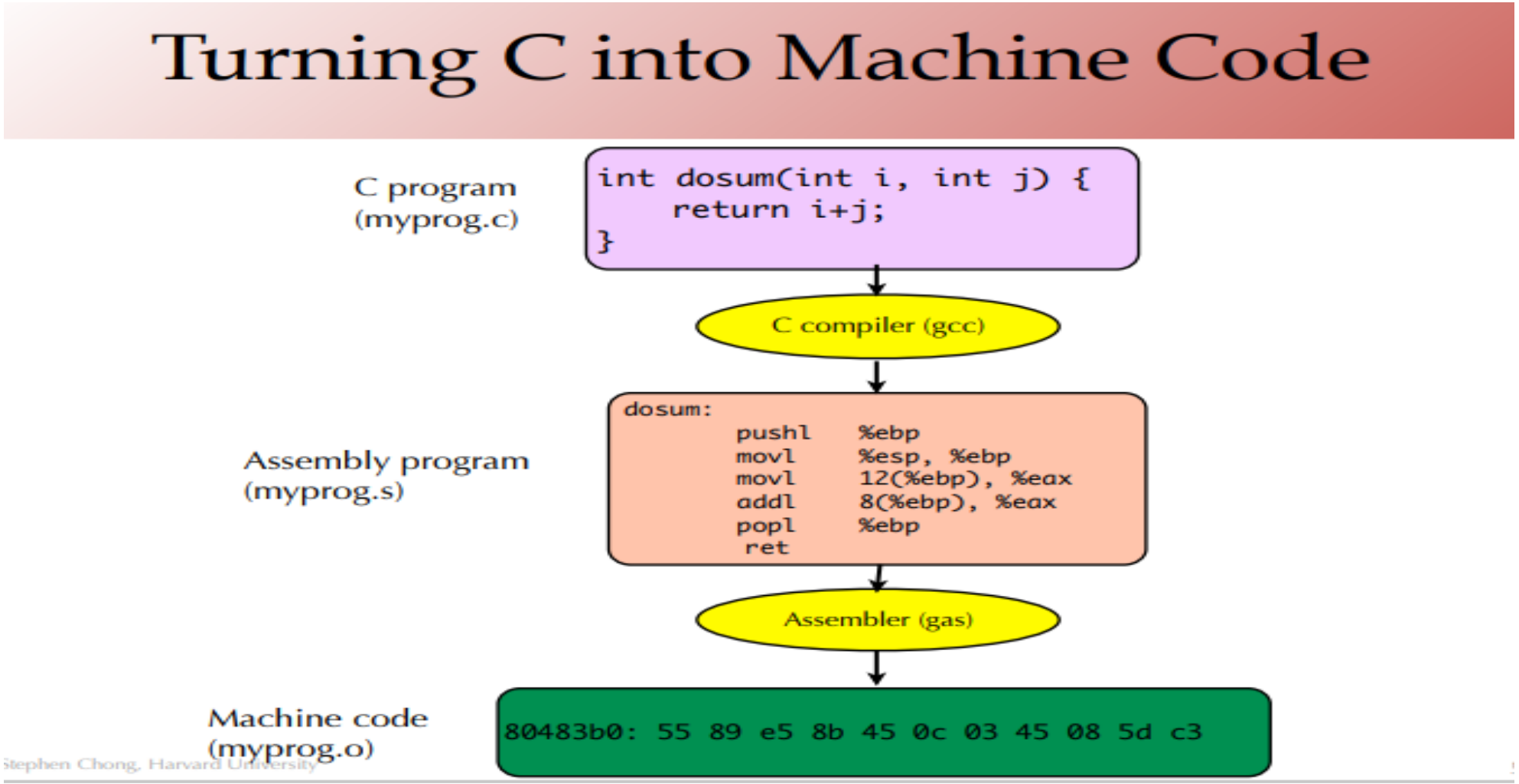
- Máy tính điện tử chỉ hiểu ngôn ngữ máy
- Chương trình được tạo ra từ các ngôn ngữ khác nhau (source language) phải được chuyển sang chương trình theo ngôn ngữ máy (target language)



Ngôn ngữ lập trình (Programming Language)

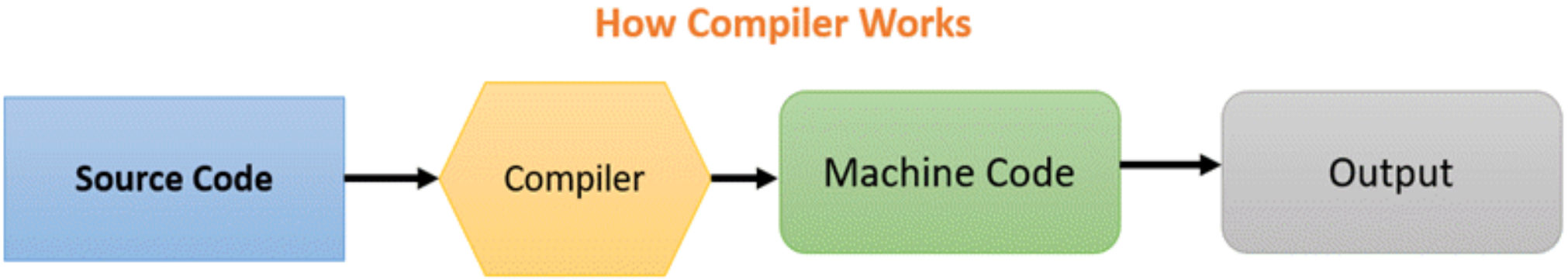


Ngôn ngữ lập trình (Programming Language)



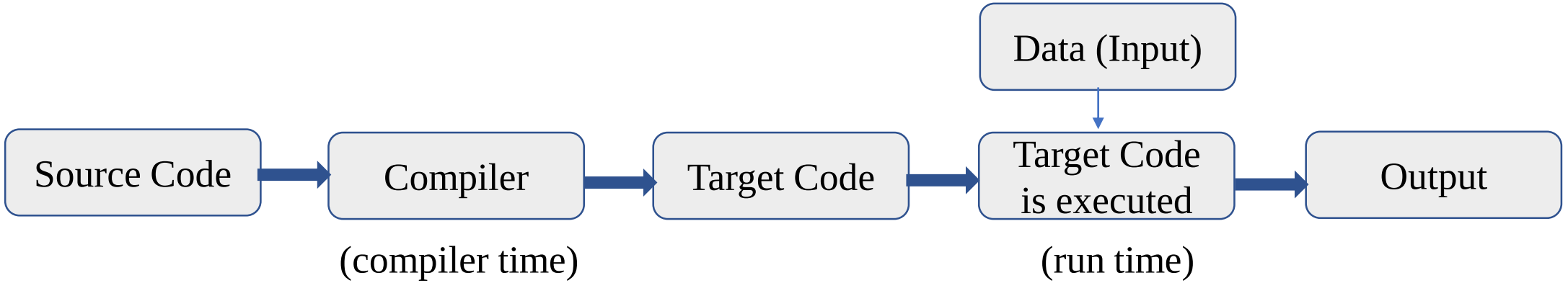
Compiler vs Interpreter

- Chương trình dịch được chia thành 2 loại: trình biên dịch (compiler) và trình thông dịch (interpreter)



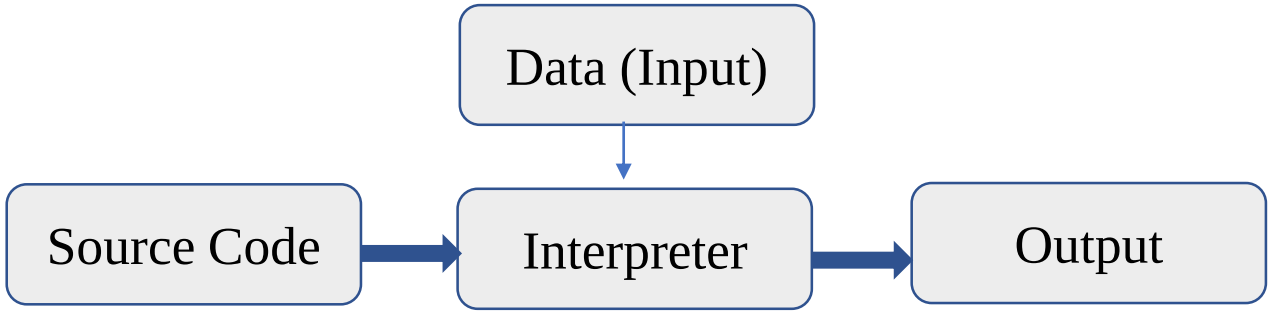
Compiler vs Interpreter

- Compiler: dịch toàn bộ chương trình nguồn sang chương trình đích rồi mới thực thi
  - Thời gian dịch (compiler time): Thời gian chuyển một chương trình nguồn sang chương trình đích
  - Thời gian thực thi (run time): Thời gian chương trình đích được thực thi



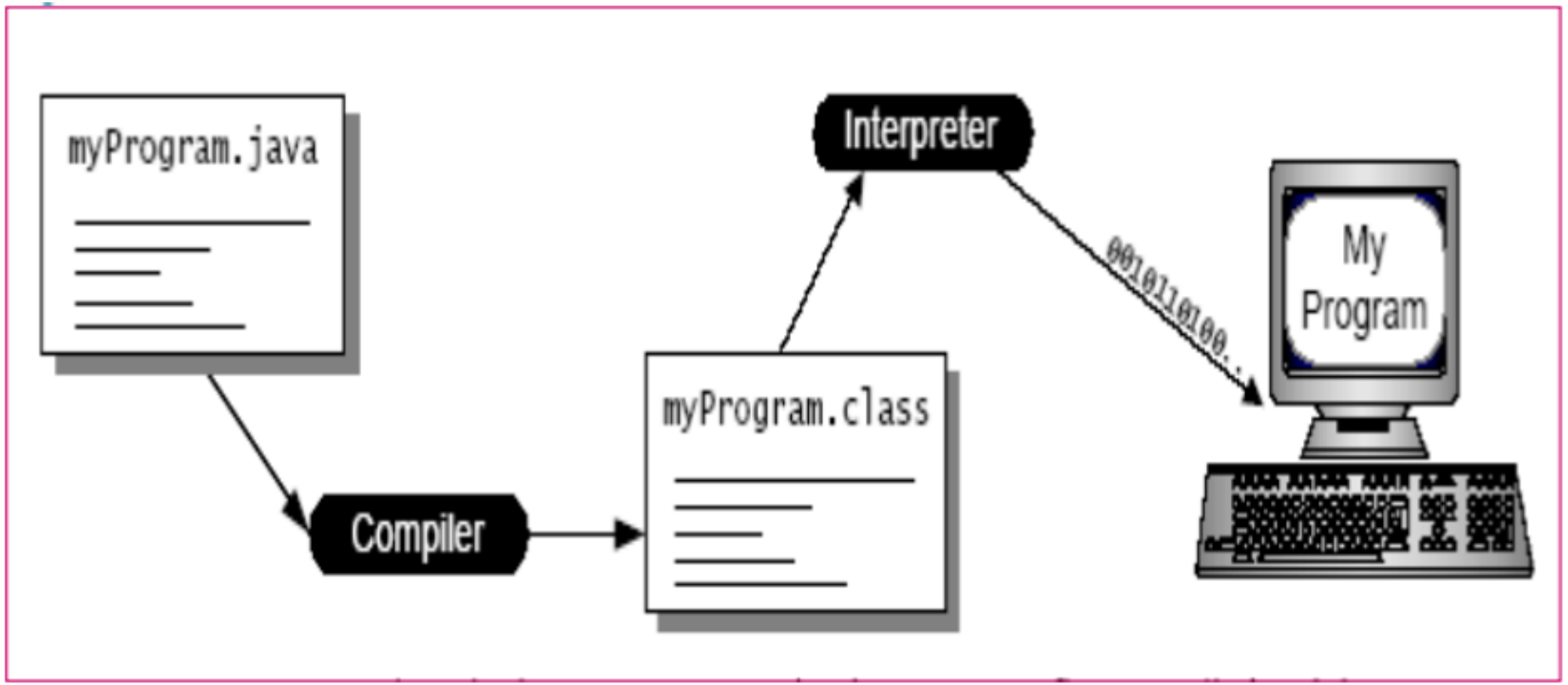
Compiler vs Interpreter

- Interpreter: phân tích (dịch) từng phát biểu (câu lệnh) theo thứ tự rồi thực thi luôn cho đến khi không còn câu lệnh nào
- Vận hành theo phương thức tương tác với người dùng



- Có thể áp dụng kỹ thuật của trình biên dịch: biên dịch chương trình nguồn sang dạng mã trung gian, sau đó từ mã trung gian sẽ được thông dịch bằng trình thông dịch

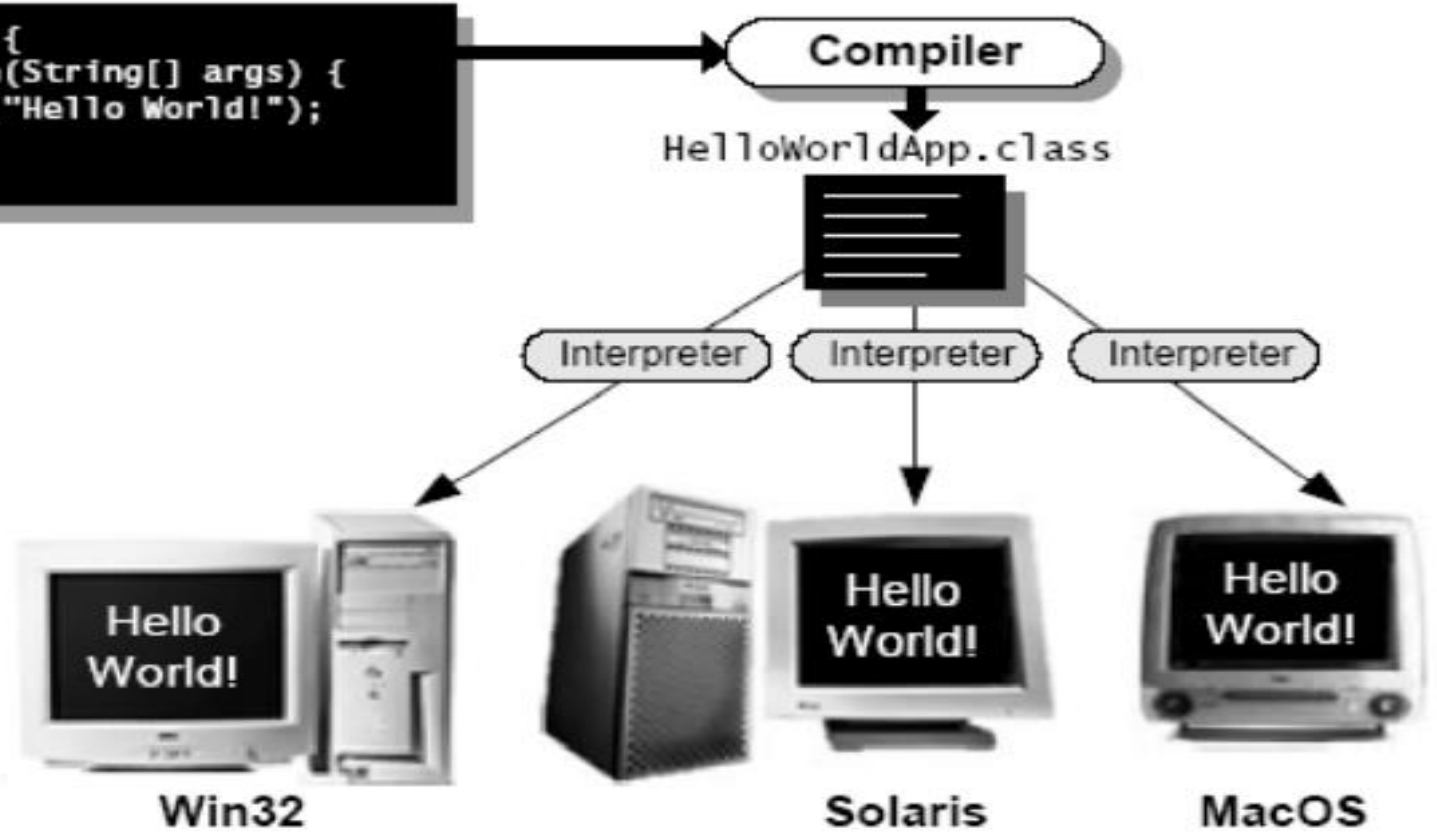
Compiler vs Interpreter



Compiler vs Interpreter

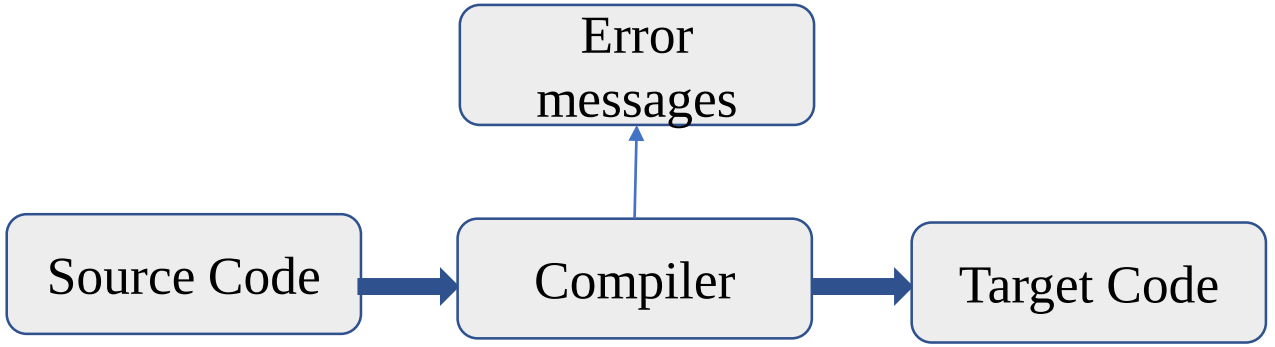
HelloWorldApp.java

```
public class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```



Compiler

- Trình biên dịch là chương trình dùng để đọc chương trình nguồn và dịch chương trình đó sang chương trình đích (chương trình tương ứng trong ngôn ngữ khác)
- Có nhiều trình biên dịch (do có nhiều ngôn ngữ nguồn và nhiều ngôn ngữ đích)
- Một phần quan trọng trong quá trình dịch là ghi nhận lỗi trong chương trình nguồn để thông báo lại

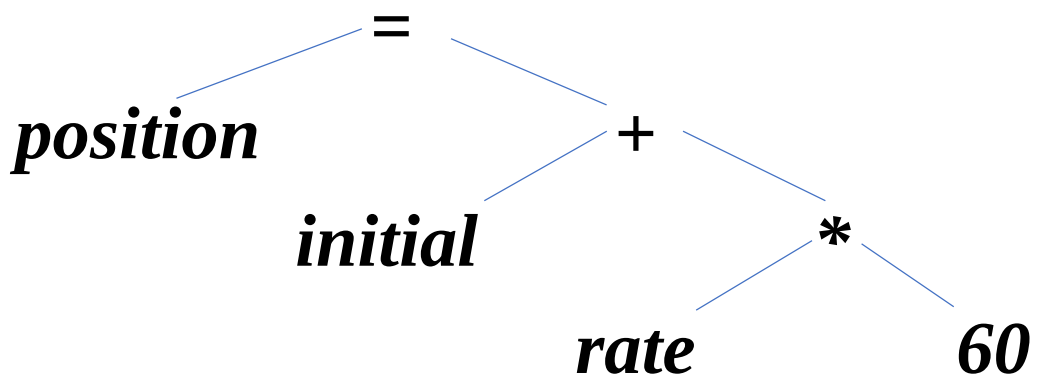


Compiler

- Chương trình nguồn là một chuỗi ký tự, do đó trình biên dịch có nhiệm vụ chuyển chuỗi ký tự này sang chuỗi ký tự khác (từ biểu diễn này sang biểu diễn khác)
- Công việc biên dịch có thể được chia thành 2 phần:
  - Phần phân tích (analysis): phân rã chương trình nguồn thành các phần cấu thành (theo cấu trúc phân cấp) và tạo ra một dạng biểu diễn (đặc tả) trung gian cho chương trình nguồn. Thường sử dụng phân cấp dạng cây, gọi là cây cú pháp (syntax tree) mà trong đó có mỗi nút là một toán tử và các nhánh con là các toán hạng
  - Phần tổng hợp (synthesis): xây dựng ngôn ngữ đích từ dạng biểu diễn (đặc tả) trung gian của chương trình nguồn. Phần tổng hợp đòi hỏi các kỹ thuật đặc biệt.

Compiler

- Ví dụ: Cây cú pháp cho câu lệnh gán: ***position = initial + rate * 60***



Compiler

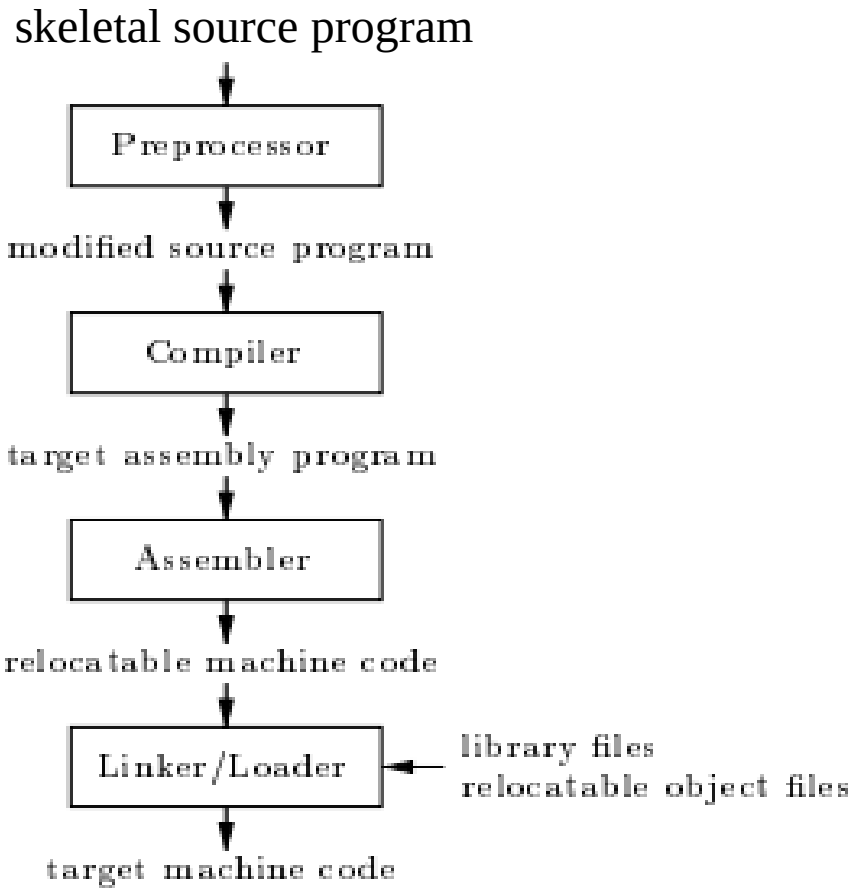
- Để tạo ra một chương trình đích có thể thực thi được (executable program), ngoài trình biên dịch có thể cần dùng nhiều chương trình khác nữa
- Các chương trình đó gồm:
  - Bộ tiền xử lý (preprocessor)
  - Trình dịch hợp ngữ (assembler)
  - Bộ tải/liên kết (loader/linker)
- Những chương trình này tạo thành hệ thống xử lý ngôn ngữ (language-processing system)

Compiler

- Preprocessor:
  - Một chương trình nguồn khung (skeletal source program) có thể được phân thành các module và được lưu trong các tập tin riêng rẻ.
  - Preprocessor (một chương trình riêng biệt) tập hợp lại các tập tin này và chuyển các macro thành các câu lệnh của chương trình nguồn (source program / modified source program).
- Chương trình đích được tạo ra bởi trình biên dịch có thể cần phải được xử lý thêm trước khi chúng có thể chạy được.
- Thông thường, trình biên dịch chỉ tạo ra mã lệnh hợp ngữ (assembly code) để trình dịch hợp ngữ (assembler) dịch thành dạng mã máy rồi được liên kết với một số thủ tục trong thư viện hệ thống thành các mã thực thi được trên máy tính.

Compiler

- Sơ đồ một hệ thống xử lý ngôn ngữ (language-processing system)



Đặc tả ngôn ngữ lập trình

- Để đặc tả ngôn ngữ lập lập trình, tối thiểu ta cần định nghĩa 3 thành phần:
  - Tập hợp các ký tự cần dùng trong các chương trình hợp lệ
  - Tập hợp các chương trình hợp lệ
  - Ngữ nghĩa (ý nghĩa) của từng chương trình hợp lệ
- Định nghĩa tập hợp các ký tự (gọi là bảng chữ cái) bằng cách liệt kê. Số lượng, ý nghĩa sử dụng của các ký tự trong bảng chữ cái của các ngôn ngữ lập trình thông thường khác nhau.
- Nhìn chung bảng chữ cái của các ngôn ngữ lập trình gồm:
  - Các chữ cái: A, B, C, …, Z, a, b, c, …, z
  - Các chữ số: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - Các ký tự khác: *, /, +, -, …

Đặc tả ngôn ngữ lập trình

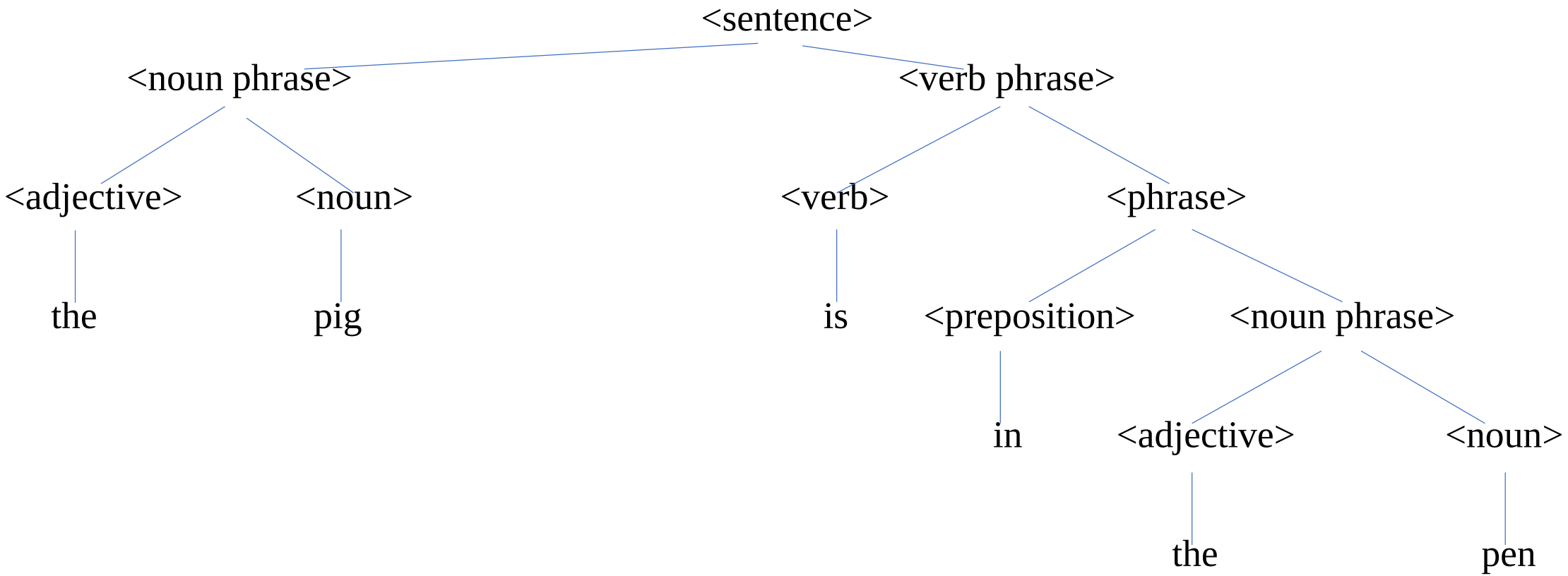
- Định nghĩa tập hợp các chương trình hợp lệ bằng tập hợp các luật của văn phạm (từ đó sản sinh ra các chương trình theo quy tắc)
- Định nghĩa ý nghĩa của chương trình hợp lệ bằng cách:
  - Phương pháp thứ nhất là định nghĩa bằng phép ánh xạ: ánh xạ mỗi chương trình vào một câu trong ngôn ngữ mà ta hiểu được ý nghĩa của nó.
  - Phương pháp thứ hai là xác định ý nghĩa của chương trình bằng một máy lý tưởng (idealized machine): ý nghĩa của mỗi chương trình được đặc tả theo ngôn từ của máy lý tưởng.
  - Phương pháp thứ ba là định nghĩa bằng đầu ra của trình biên dịch: đặc tả bằng cặp (x, y), trong đó x là chương trình nguồn và y là chương trình đích. Lúc đó xem (x,y) là sự biên dịch, với x là chuỗi được định nghĩa trên tập bảng chữ cái Σ và y là chuỗi được định nghĩa trên tập bảng chữ cái Δ thì sự biên dịch là phép ánh xạ từ Σ^* sang Δ^*

Cú pháp và ngữ nghĩa

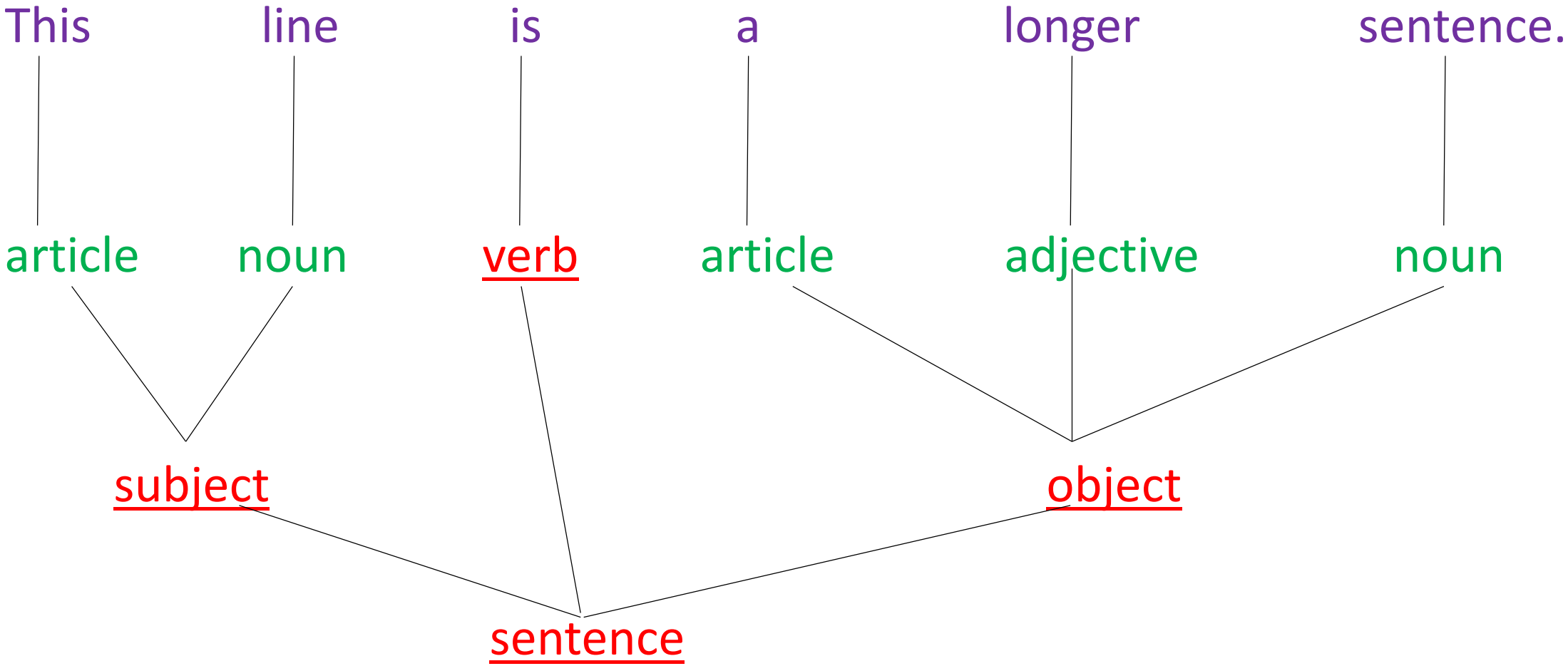
- Để tiện lợi hơn trong đặc tả và thể hiện sự biên dịch, ta xem sự biên dịch bao gồm hai phép ánh xạ đơn giản hơn:
  - Thứ nhất là phép ánh xạ cú pháp (syntactic mapping): ánh xạ một chương trình viết trong ngôn ngữ nguồn thành một cấu trúc (cấu trúc này trở thành đầu vào của phép ánh xạ thứ hai). Đầu ra (kết quả) của phép ánh xạ cú pháp này là cây cú pháp.
  - Thứ hai là phép ánh xạ ngữ nghĩa (semantic mapping): cấu trúc cú pháp sẽ được ánh xạ thành ngôn ngữ đích.

Cú pháp và ngữ nghĩa

- Ví dụ: Cấu trúc của câu tiếng Anh “The pig is in the pen” (kết quả của sự “bẻ” câu đầu vào thành các phần tử cú pháp nhờ vào luật văn phạm của tiếng Anh)

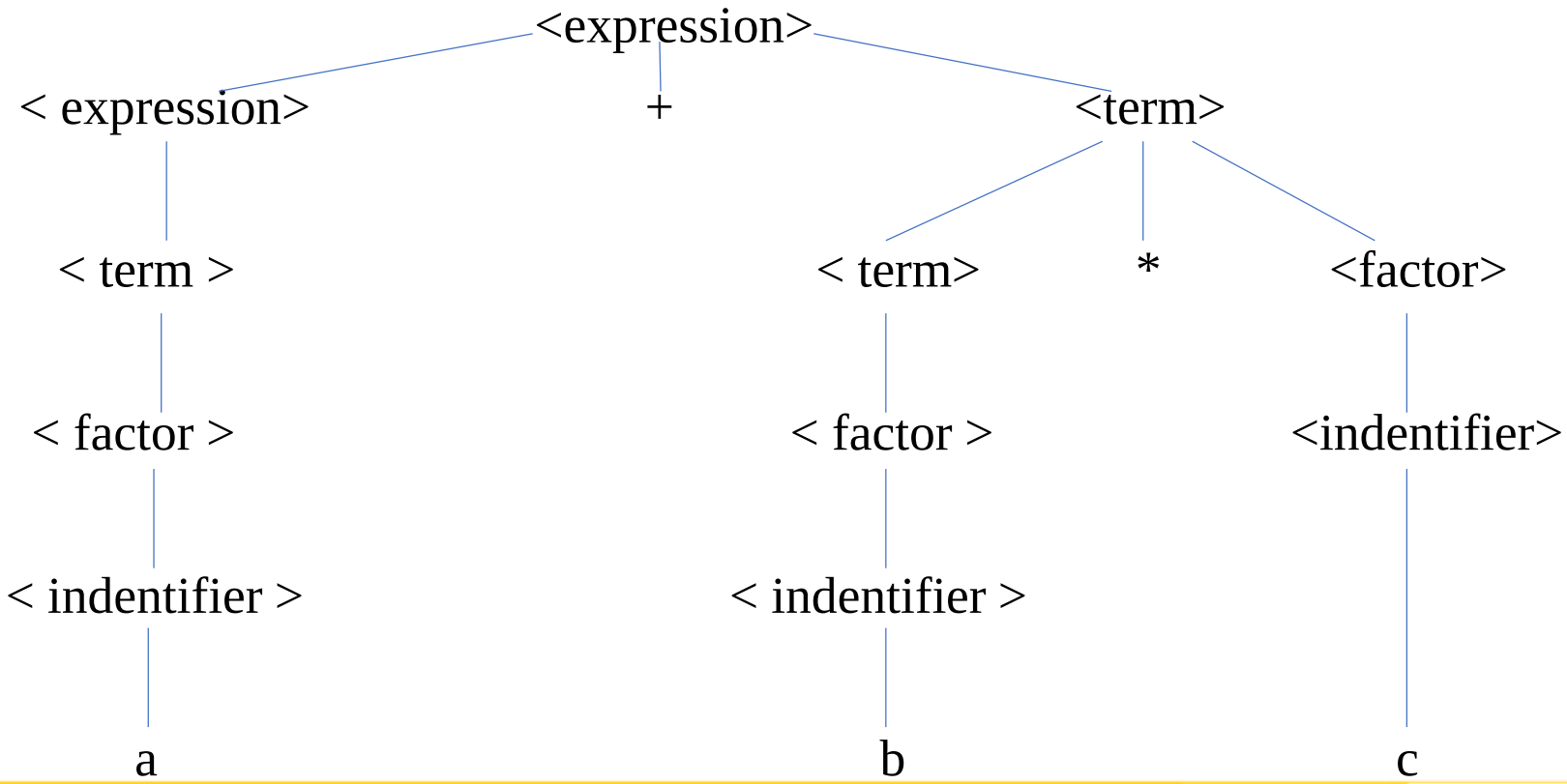


Cú pháp và ngữ nghĩa



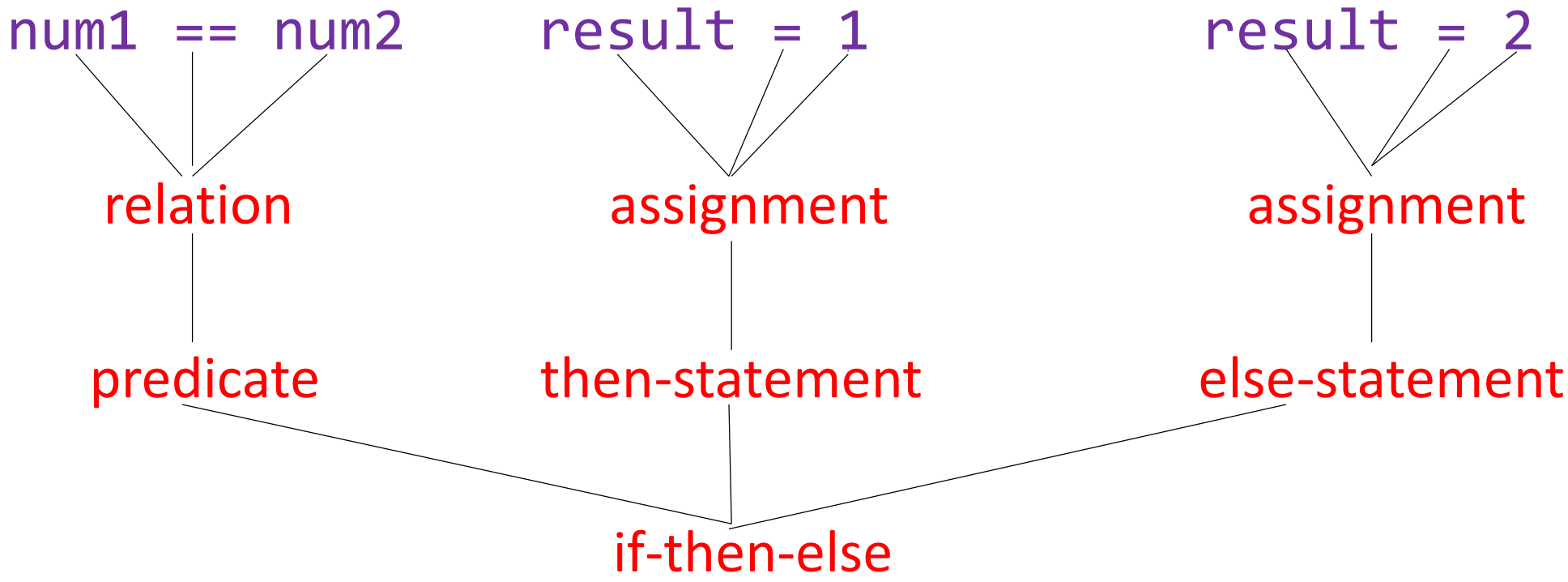
Cú pháp và ngữ nghĩa

- Ví dụ: Cấu trúc của chuỗi ký tự a+b*c (chương trình được viết trong ngôn ngữ lập trình) là kết quả của sự ‘bẻ” chuỗi đầu vào thành các phần tử cú pháp theo luật cú pháp của ngôn ngữ lập trình.



Cú pháp và ngữ nghĩa

```
if num1 == num2 then result = 1; else result = 2;
```



Cú pháp và ngữ nghĩa

- Mỗi câu của ngôn ngữ đều tồn tại cây cú pháp của nó.
- Quá trình tìm ra cây cú pháp của một câu gọi là quá trình phân tích cú pháp (syntax analysis/parsing) của câu đó. Quá trình này được thực hiện dựa trên cơ sở các luật cú pháp của ngôn ngữ, gọi là luật sinh (production rule)
- Việc đặc tả cú pháp và ngữ nghĩa của ngôn ngữ lập trình không đơn giản và chưa có phương pháp tổng quát để đặc tả.
- Tuy nhiên có thể sử dụng các khái niệm của lý thuyết ngôn ngữ để đặc tả (văn phạm phi ngữ cảnh,…)

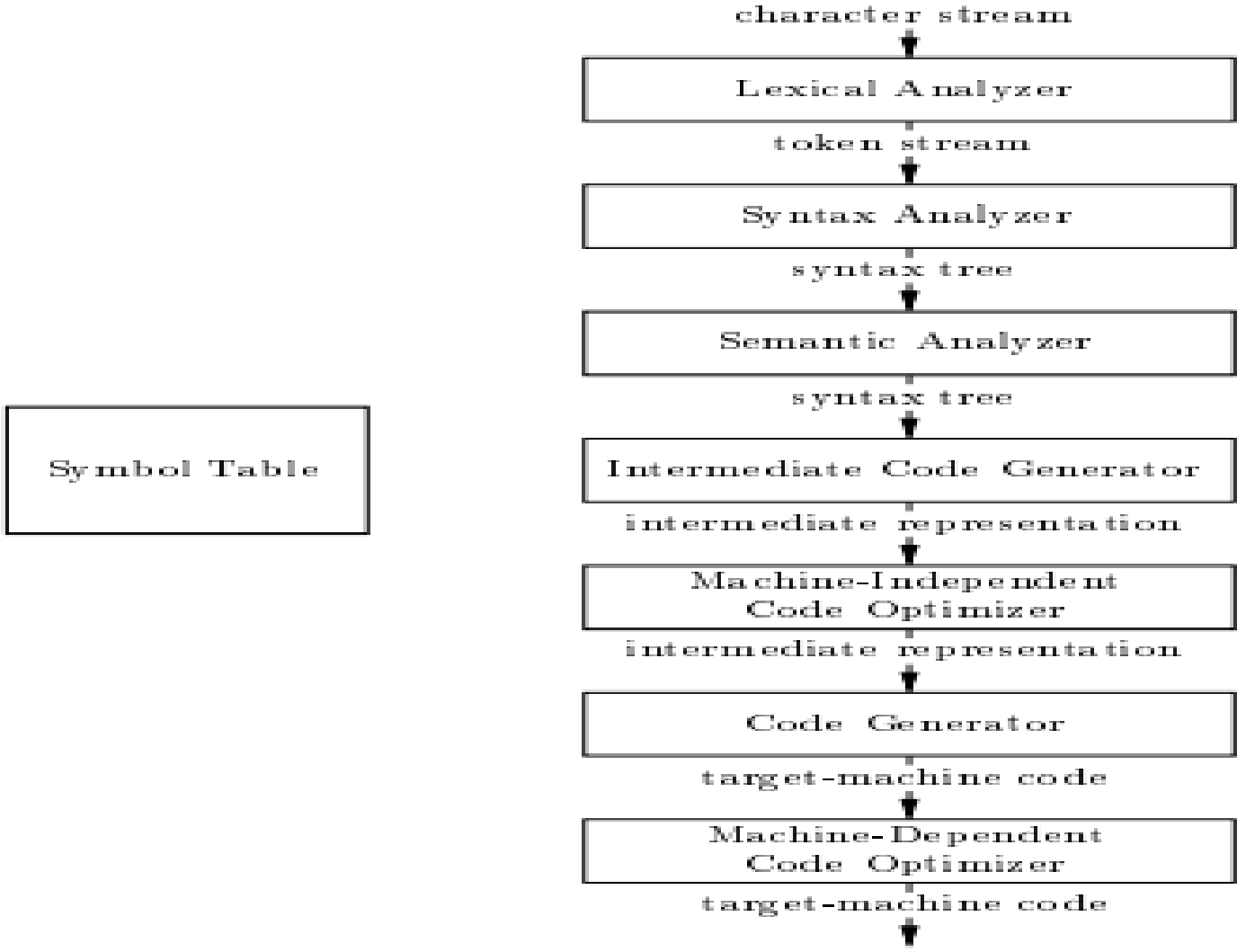
Cấu trúc của một trình biên dịch

- Lexical Analysis (phân tích từ vựng)
- Parsing (phân tích cú pháp)
- Semantic Analysis (phân tích ngữ nghĩa)
- Optimization (tối ưu mã)
- Code Generation (sinh mã)

Cấu trúc của một trình biên dịch

- Quá trình biên dịch được chia thành nhiều giai đoạn; qua mỗi giai đoạn chương trình nguồn được chuyển đổi từ dạng biểu diễn này sang một dạng biểu diễn khác.
- Trong thực tế xây dựng trình biên dịch, đôi khi các giai đoạn này được nhóm lại với nhau.
- Các giai đoạn chi tiết của một trình biên dịch:
  - Phân tích từ vựng
  - Quản lý bảng danh biểu
  - Phát hiện và thông báo lỗi
  - Phân tích cú pháp
  - Phân tích ngữ nghĩa
  - Sinh mã trung gian
  - Tối ưu mã trung gian
  - Sinh mã đối tượng

Cấu trúc của một trình biên dịch



Phân tích từ vựng

- Phân tích từ vựng (Lexical Analysis / Scanning): đây là giai đoan đầu của quá trình biên dịch và được thực hiện bởi bộ phân tích từ vựng (Lexical Analyser), trong đó:
  - Đầu vào là chuỗi các ký tự cho phép của một ngôn ngữ lập trình
  - Đầu ra là các token (thực thể cú pháp, từ tố): các ký tự đầu vào được nhóm thành các token
- Token là ký hiệu kết thúc (terminal symbol); mỗi token có một cấu trúc từ vựng là một cặp gồm loại token (*token-name*) và dữ liệu (*attribute-value*), trong đó:
  - *token-name* là phạm trù cú pháp gồm: định danh, hằng, từ khóa, phép toán, ký tự phân cách, ký tự đặc biệt,…
  - *attribute-value* là con trỏ (trỏ đến thông tin của token) được cất giữ trong bảng danh biểu (symbol table)

Phân tích từ vựng

- *token-name* được sử dụng trong giai đoạn phân tích cú pháp và *attribute-value* được sử dụng trong giai đoạn phân tích ngữ nghĩa, sinh mã đối tượng.
- Với ngôn ngữ lập trình cho trước, số lượng loại token là hữu hạn.
- Sau đây ta gọi chung mỗi cặp gồm loại token và dữ liệu là "token".
- Đầu ra của bộ phân tích từ vựng là đầu vào của bộ phân tích cú pháp.
- Ví dụ chương trình nguồn là phát biểu gán trong ngôn ngữ lập trình:

  COST := (PRICE + TAX) * 65
- Bộ phân tích từ vựng có nhiệm vụ nhân biết:
  - COST, PRICE, TAX là nhóm token thuộc loại định danh (biến)
  - 65 là token thuộc loại hằng
  - Các ký tự := ( ) + * tự bản thân là các token

Phân tích từ vựng

- Giả sử tất cả hằng có loại token là <num>, định danh có loại token là <id> (*token-name*)
- Lúc đó *attribute-value* là con trỏ, trỏ đến vị trí của các token trong bảng danh biểu, chứa đựng trị từ vựng (lexeme) của token và các thuộc tính khác của token.
- Đầu ra của bộ phân tích từ vựng của ví dụ trên sẽ là:

  (<id>,1) < := > (<(><id>,2<)> < + > <(><id>,3<)>) < * > (<num>,4)

  và viết gọn lại là $id_1 := (id_2 + id_3) * num_4$

Quản lý bảng danh biểu

- Quản lý bảng danh biểu (symbol table management): lưu thông tin của các token
- Ví dụ với phát biểu COST := (PRICE + TAX) * 65, bộ phân tích từ vựng sẽ cho bảng danh biểu như sau:

attribute-value	*token-name*	*lexeme*	*Các thuộc tính khác*
1	id	COST	Biến thực
2	id	PRICE	Biến thực
3	id	TAX	Biến thực
4	num	65	Hằng số nguyên

Quản lý bảng danh biểu

- Nếu bộ phân tích từ vựng nhận tiếp các chuỗi ký tự của chương trình đầu vào để nhận dạng token thì bảng danh biểu thường xuyên được truy xuất
- Việc truy xuất bảng danh biểu nhằm 2 mục đích:
  - Nếu token vừa nhận dạng đã được lưu trữ trong bảng danh biểu thì phần *attribute-value* sẽ được truy xuất
  - Nếu token vừa nhận dạng là mới thì token sẽ được lưu vào bảng danh biểu
- Bảng danh biểu thường xuyên được truy xuất để thêm token mới hoặc truy xuất token đã có nên phải thỏa mãn 2 điều kiện:
  - Thực hiện nhanh các thao tác thêm token hoặc thông tin của token
  - Có khả năng truy xuất nhanh các thông tin của token

Phát hiện và thông báo lỗi

- Ở mỗi giai đoạn của quá trình biên dịch một chương trình nguồn đều có thể có lỗi xảy ra.
- Sau khi phát hiện một lỗi, trình biên dịch xem xét lỗi đó để xem có tiếp tục quá trình biên dịch hay không (không phải dừng hẳn).
- Trong giai đoạn phân tích từ vựng, phân tích cú pháp, phân tích ngữ nghĩa thường xuất hiện nhiều lỗi do trình biên dịch phát hiện:
  - Phân tích từ vựng: lỗi được phát hiện khi phần còn lại trên băng nhập không thể phân tích để tạo thành token
  - Phân tích cú pháp: lỗi xảy ra khi bộ phận tích cú pháp không thể xây dựng cấu trúc cú pháp cho chuỗi token nhận được
  - Phân tích ngữ nghĩa: Lỗi xuất hiện khi trình biên dịch kiểm tra kiểu dữ liệu của hai toán hạng thuộc một phép toán không phù hợp

Phân tích cú pháp

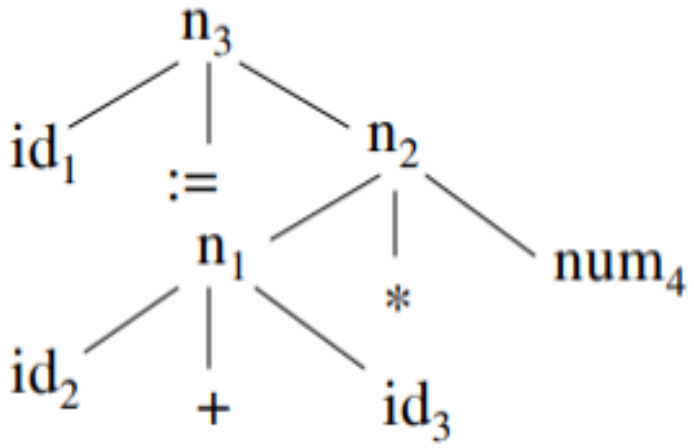
- Phân tích cú pháp (Syntax Analysis / Parsing): được thực hiện bởi bộ phân tích cú pháp (Syntax Analyser)
- Đầu vào là chuỗi các token kết quả (gồm 2 thành phần <*token-name*, *attribute-value*>) của bộ phân tích từ vựng
- Tuy nhiên bộ phân tích cú pháp chỉ xét thành phần thứ nhất *token-name*
- Đây là quá trình mà chuỗi các token được kiểm tra xem có thể được biểu diễn bằng cấu trúc cú pháp của ngôn ngữ lập trình đầu vào cho trước hay không
- Nếu có một cấu trúc cú pháp cho cho chuỗi các token đầu vào thì cấu trúc cú pháp được sinh ra đó là đầu ra của bộ phân tích cú pháp
- Ở giai đoạn sinh mã, cấu trúc cú pháp sẽ được xem xét để từ đó sinh ra mã cho chuỗi ký tự của chương trình nguồn

Phân tích cú pháp

- Ví dụ chương trình nguồn là phát biểu gán trong ngôn ngữ lập trình:

  COST := (PRICE + TAX) * 65

  - Kết quả của phân tích từ vựng là id_1 := (id_2 +id_3) * num_4
  - Kết quả của phân tích cú pháp là



Phân tích cú pháp

- Với mỗi chuỗi token đầu vào và tập luật sinh cho trước, bộ phân tích cú pháp sẽ tìm ra cây cú pháp cho chuỗi token đầu vào.
  - Khi cây cú pháp được xây dựng xong thì quá trình phân tích cú pháp kết thúc thành công.
  - Ngược lại, nếu bộ phân tích cú pháp áp dụng tất cả các luật hiện có nhưng không thể xây dựng được cây cú pháp của chuỗi token đầu vào thì bộ phân tích cú pháp sẽ thông báo là chuỗi đầu vào không được viết đúng cú pháp của ngôn ngữ lập trình
- Ví dụ với cây cú pháp kết quả ở trên, ta thấy:
  - n_1 là nút mô tả phép toán id_2 + id_3 tức là (PRICE + TAX)
  - n_2 là nút mô tả phép toán n_1 * num_4 (tức là n_1 * 65)
  - n_3 là nút mô tả phép toán id_1 := n_2 (tức là gán n_2 có kết quả là n_1* 65 vào biến COST)
  - Dấu ( ) không có, nhưng được thể hiện qua thứ tự thực hiện phép toán

Phân tích cú pháp

- Ví dụ một số ngôn ngữ lập trình định nghĩa đệ quy biểu thức số học như sau:
  - Định danh (identifier/id) là một biểu thức (expression/expr),
  - Số (number) là một biểu thức,
  - Nếu expr1 và expr2 là các biểu thức thì:

    expr1 + expr2

    expr1 - expr2

    expr2 - expr1

    expr1 * expr2

    (expr1)

    là các biểu thức.

Phân tích cú pháp

- Ví dụ một số ngôn ngữ lập trình định nghĩa câu lệnh (statement) theo cách như sau:
  - Nếu id1 là một định danh và expr2 là một biểu thức thì id1 = expr2 là một câu lệnh (stmt),
  - Nếu expr1 là một biểu thức và stmt2 là một câu lệnh thì:

    while (expr1) do stmt2

    if (expr1) then stmt2

    là các lệnh.

Phân tích ngữ nghĩa

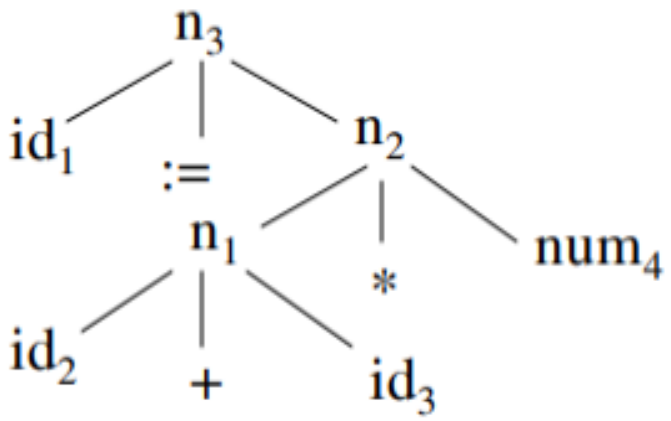
- Phân tích ngữ nghĩa (Semantic Analysis): được thực hiện bởi bộ phân tích ngữ nghĩa (Semantic Analyser)
- Đầu vào là cây cú pháp (kết quả của phân tích cú pháp)
- Bộ phân tích ngữ nghĩa sẽ kiểm tra lỗi về ngữ nghĩa:
  - Kiểm tra kiểu dữ liệu (type checking) và ép chuyển đổi kiểu (nếu có)
  - Xử lý từng phép toán dựa trên cây cú pháp: với mỗi phép toán bộ phân tích ngữ nghĩa sẽ kiểm tra các toán hạng xem kiểu dữ liệu của chúng có cho phép tham gia vào phép toán đó không
- Ví dụ:
  - Với biểu thức f+b, trong đó f có kiểu số thực và b có kiểu logic: phép toán không không thực hiện được
  - Với biểu thức f+i, trong đó f có kiểu số thực và i có kiểu số nguyên: phép toán có thể thực hiện được

Phân tích ngữ nghĩa

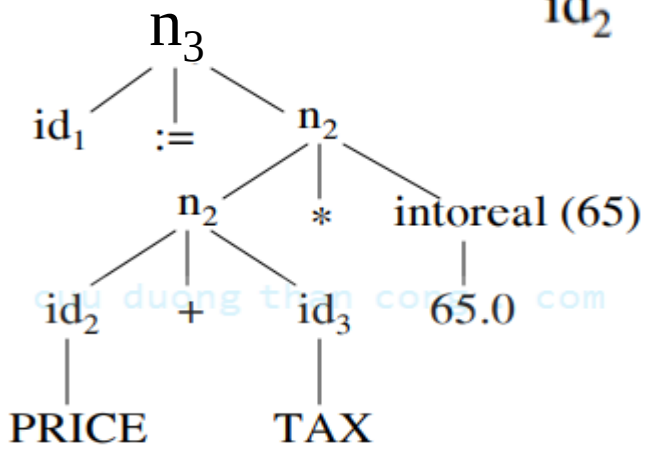
- Ví dụ chương trình nguồn là phát biểu gán trong ngôn ngữ lập trình:

  COST := (PRICE + TAX) * 65

  - Kết quả của phân tích từ vựng là $id_1 := (id_2 + id_3) * num_4$
  - Kết quả của phân tích của pháp là



  - Kết quả của phân tích ngữ nghĩa



Sinh mã trung gian

- Sinh mã trung gian (Intermediate Code Generator): đây là giai đoạn tạo ra sự biểu diễn trung gian của chương trình nguồn
- Sự biểu diễn trung gian này được hiểu như là chương trình của máy tính trừu tượng (abstract machine)
- Ngôn ngữ dùng cho máy tính trừu tượng là mã trung gian.
- Mã trung gian có 2 đặc điểm quan trong:
  - Dễ được sinh ra
  - Dễ chuyển sang mã đối tượng của chương trình đích

Sinh mã trung gian

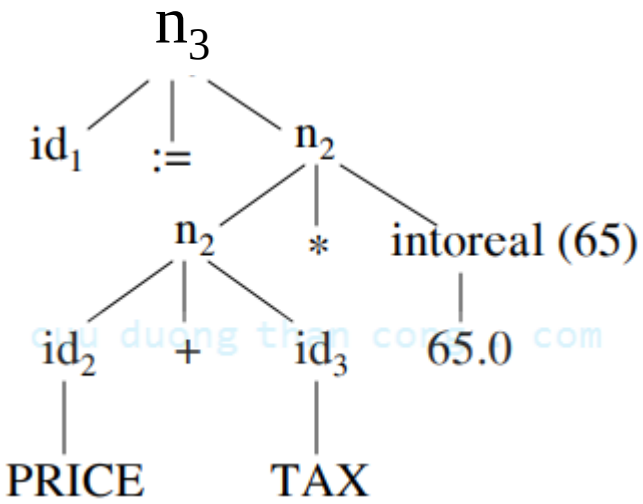
- Bước này sử dụng cây cú pháp đã được xử lý ngữ nghĩa (kết quả phân tích ngữ nghĩa) để sinh mã trung gian
- Có nhiều cách biểu diễn mã trung gian.
- Thông thường sử dụng mã trung gian dạng "mã máy 3 địa chỉ" (three-address code), tương tự như dạng hợp ngữ cho một máy mà trong đó mỗi vị trí bộ nhớ có thể đóng vai trò như một thanh ghi (register).

Sinh mã trung gian

- Ví dụ chương trình nguồn là phát biểu gán trong ngôn ngữ lập trình:

  COST := (PRICE + TAX) * 65

  - Kết quả của phân tích ngữ nghĩa:



  - Kết quả sinh mã trung gian:

    $temp_1$ = intoreal(65)

    $temp_2$ = id_2 + id_3

    $temp_3$ = $temp_2$ * $temp_1$

    id_1 = $temp_3$

Tối ưu mã trung gian

- Tối ưu mã trung gian (Code Optimization): giai đoan này sẽ giảm bớt một số bước trong mã trung gian để khi sinh ra mã đối tượng làm cho thời gian thực thi mã đối tượng ngắn hơn.
- Thực hiện tối ưu mã trung gian trong thời gian biên dịch sẽ làm thời gian tăng lên, do đó cần có kỹ thuật để khắc phục
- Ví dụ phân tích kết quả sinh mã trung gian

$temp_1$ = intoreal(65) (1)

$temp_2 = id_2 + id_3$ (2)

$temp_3 = temp_2 * temp_1$ (3)

$id_1 = temp_3$ (4)

Tối ưu mã trung gian

- Ta thấy:
  - Bước (1): chuyển số nguyên sang số thực sẽ được thực hiện ngay trong thời gian dịch, nên phép toán *intoreal* sẽ được bỏ đi.
  - Bước (4): $temp_3$ chỉ dùng 1 lần để gán giá trị cho id_1, do đó có thể ghép (3) và (4)
- Do đó kết quả sinh mã trung gian

$temp_1 = intoreal(65)$ (1)

$temp_2 = id_2 + id_3$ (2)

$temp_3 = temp_2 * temp_1$ (3)

$id_1 = temp_3$ (4)

- Có thể tối ưu thành:

$temp_1 = id_2 + id_3$

$id_1 = temp_1 * 65.0$

Sinh mã đối tượng

- Sinh mã đối tượng (Code Generation): giai đoạn cuối của trình biên dịch
- Ví dụ chương trình nguồn là phát biểu gán trong ngôn ngữ lập trình:

COST := (PRICE + TAX) * 65

- Kết quả tối ưu mã trung gian:

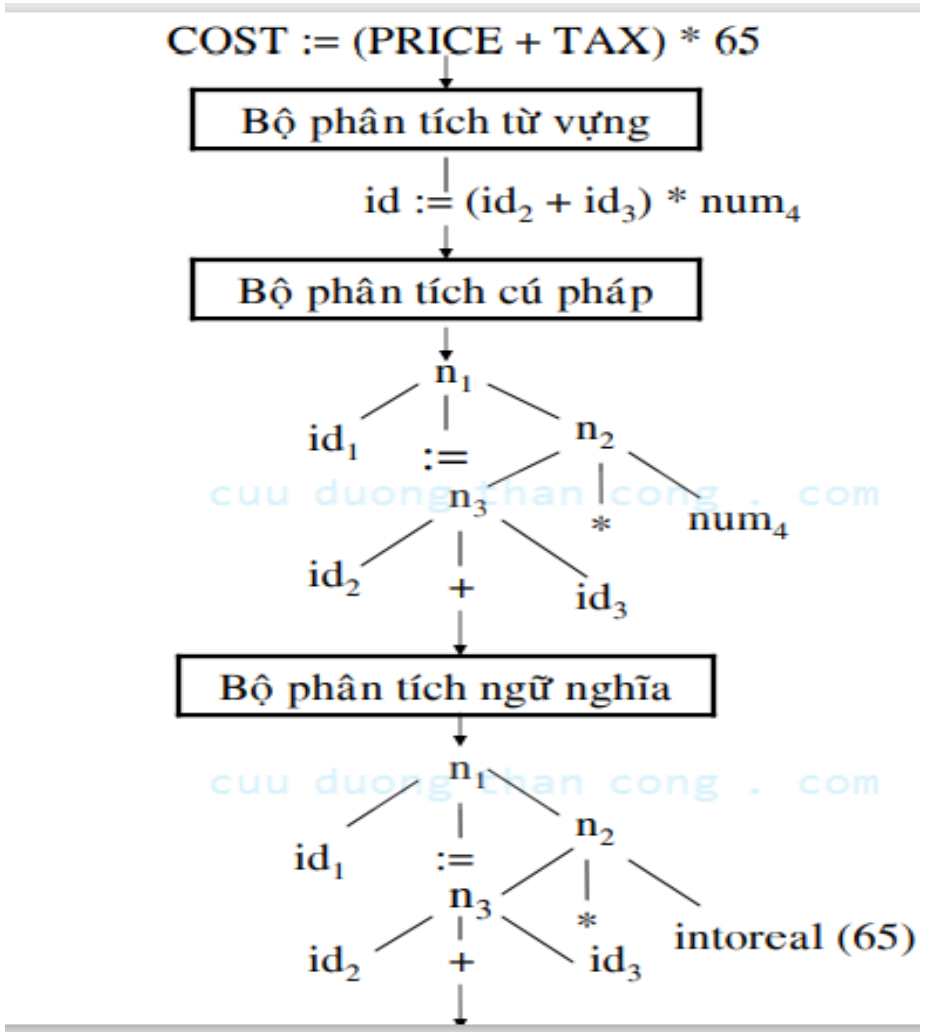
$temp_1 = id_2 + id_3$

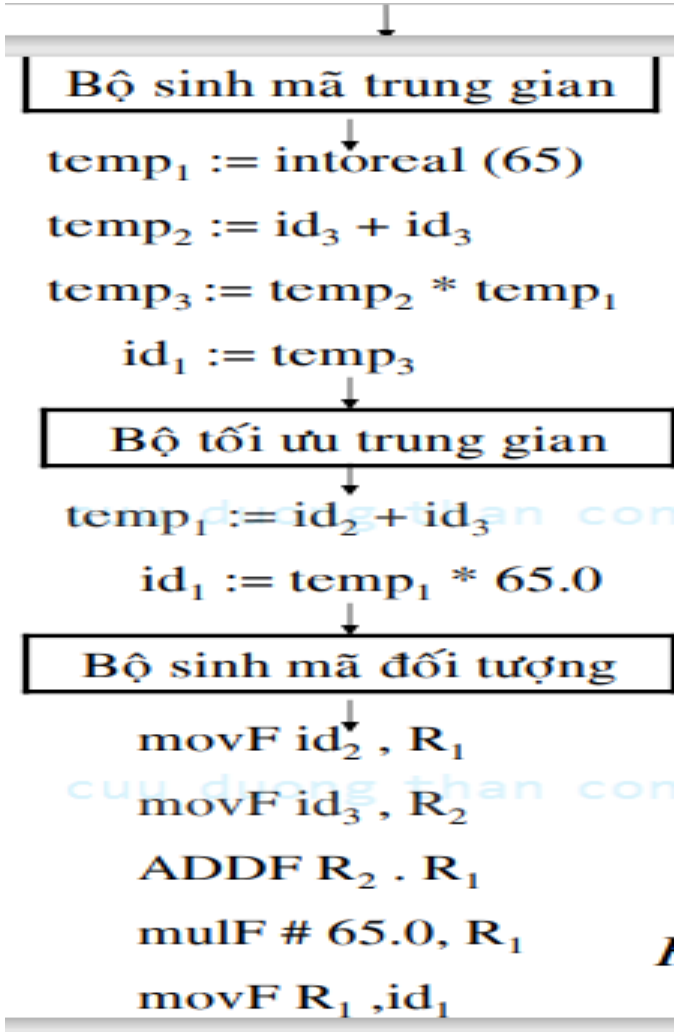
$id_1 = temp_1 * 65.0$

- Kết quả sinh mã đối tượng:

movF id_2, R_1	chuyển giá trị từ vị trí nhớ có tên PRICE, thuộc loại token id_2 vào thanh ghi R_1
movF id_3, R_2	chuyển giá trị từ vị trí nhớ có tên TAX, thuộc loại token id_3 vào thanh ghi R_2
addF R_2, R_1	cộng nội dung 2 thanh ghi R_1 và R_2 (kết quả lưu trong R_1)
mulF #65.0, R_1	nhân hằng có giá trị 65.0 với giá trị nằm trong thanh ghi R_1 (kết quả lưu trong R_1)
movF R_1, id_1	chuyển nội dung trong thanh ghi R1 vào vị trí nhớ có tên COST thuộc loại token id_1

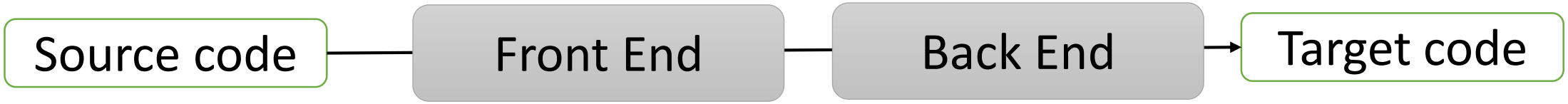
Cấu trúc của một trình biên dịch





Cấu trúc của một trình biên dịch

❑ Các giai đoạn của một trình biên dịch có thể được nhóm lại thành các giai đoạn lớn hơn gồm: Front End (giai đoạn trước) và Back End (giai đoạn sau)



- *Front End* (Analysis) gồm các giai đoạn (hoặc các phần của các giai đoạn) chỉ phụ thuộc vào ngôn ngữ nguồn mà hầu như không phụ thuộc vào ngôn ngữ đích

Giai đoạn này gồm: phân tích từ vựng, phân tích cú pháp, phân tích ngữ nghĩa, sinh mã trung gian, tối ưu mã trung gian, tạo và quản lý bảng danh biểu, thông báo lỗi.

- *Back End* (Synthesis) gồm các các phần phụ thuộc vào ngôn ngữ đích mà không phụ thuộc vào ngôn ngữ nguồn

Giai đoạn này gồm: sinh mã đối tượng, tối ưu mã và các tác vụ của quản lý bảng danh biểu, thông báo lỗi

Cấu trúc của một trình biên dịch

- Thông thường một số giai đoạn của một trình biên dịch có thể được thực hiện trong một chuyến (passes)
- Ví dụ phân tích từ vựng, phân tích cú pháp, phân tích ngữ nghĩa và sinh mã trung gian có thể được gom lại thực hiện trong một chuyến. Lúc đó:
  - Chuỗi token được nhận dạng ở bước phân tích từ vựng sẽ được dịch thẳng sang mã trung gian
  - Vai trò của bộ phân tích cú pháp là bao trùm, nó "trông coi" toàn bộ hoạt động của chuyến
  - Bộ phân tích cú pháp có nhiệm vụ: phát hiện cấu trúc văn phạm của các token đưa đến cho nó; biết lúc nào cần lấy tiếp token và sẽ gọi bộ phận phân tích từ vựng nhận dạng token tiếp theo
  - Khi phát hiện xong một cấu trúc văn phạm, bộ phân tích cú pháp sẽ gọi bộ sinh mã trung gian, để thực hiện phân tích ngữ nghĩa và tạo mã trung gian.