

Chapter 2

1. 언어 번역기 종류

학습 목표

- 01 어떤 종류의 번역기가 있는가?
- 02 인터프리터와 컴파일러의 차이점은?
- 03 자바 언어 실행 과정은 어떻게 다른가?
- 04 크로스 컴파일은 왜 필요한가?
- 05 언어처리시스템에서 링커와 로더의 역할은?

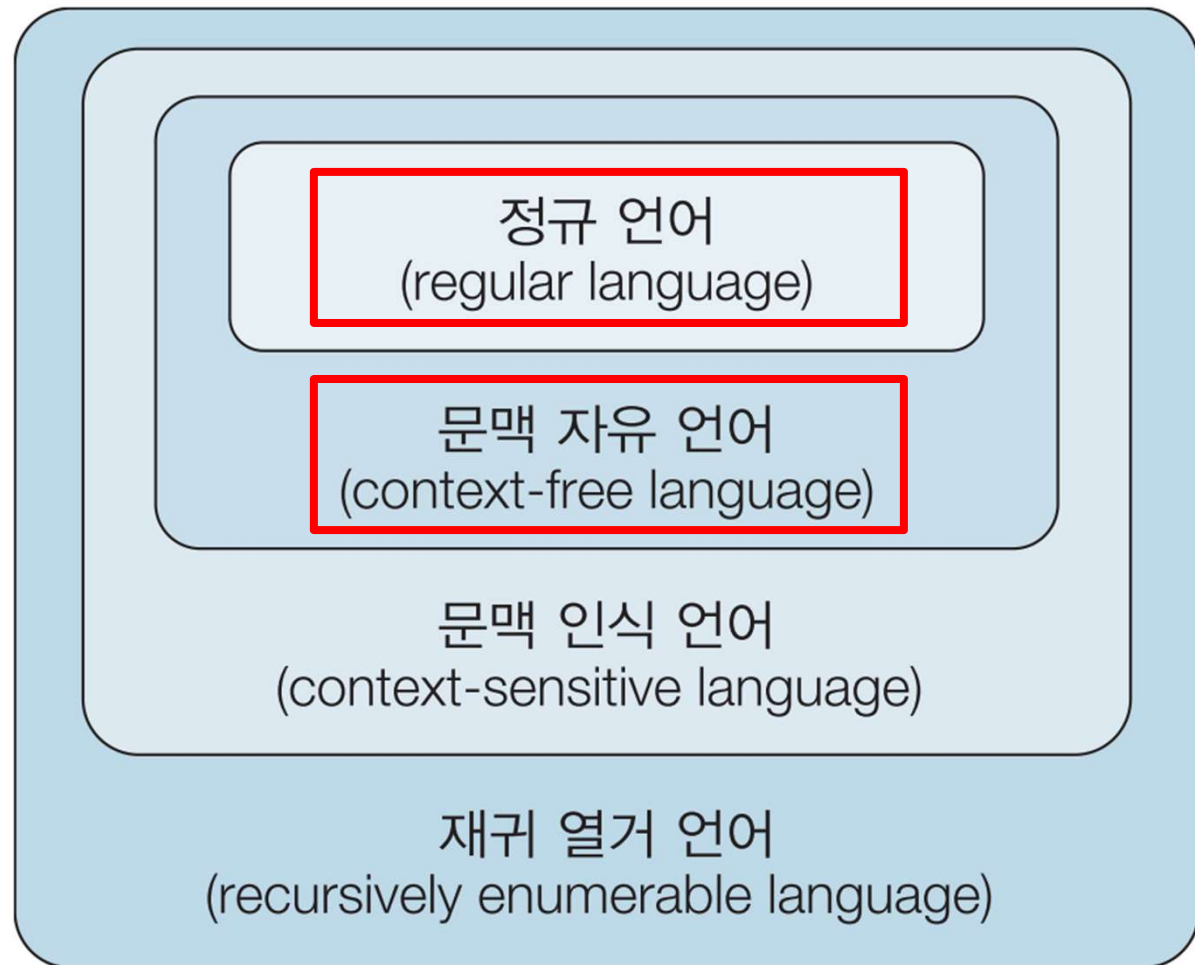
형식 언어

■ 언어의 종류

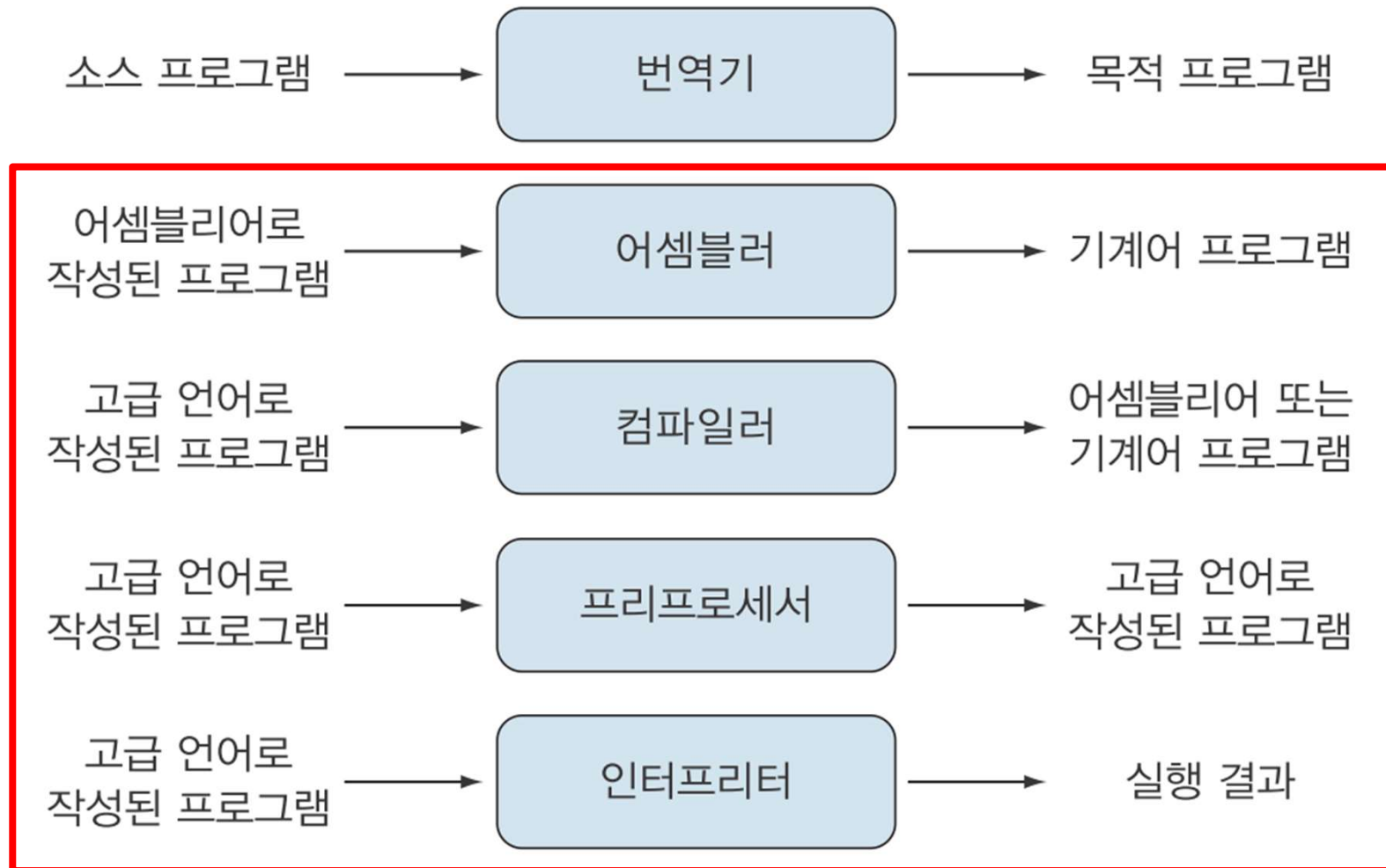
- natural language
 - 자연어
- formal language
 - 형식 언어

■ 형식언어 종류

- 4종류
- Chomsky 분류에 따름



번역기 종류



번역기 종류 : Assembler (1/2)

■ 어셈블러

- 어셈블리어 코드를 기계어로 변환

mov al, 061h → 10110000 01100001

- 어셈블리어 코드

- 사람이 이해하기 쉽게 binary code를 기호화(mnemonics)

- 어셈블리어 코드 예

LOAD R_1, a -- 변수 a의 값을 레지스터(register) R_1 에 저장

ADD $R_1, \#2$ -- register R_1 의 값과 숫자 상수 2를 더해 다시 R_1 에 저장

STORE b, R_1 -- register R_1 에 저장되어 있는 값을 변수 b에 저장

- 변수 a, b는 메모리 주소 ➔ 이 주소에 해당 변수 값이 저장됨.
 - 실행 결과, 변수 b 에 a+2 값이 저장됨 ($b = a + 2$)

번역기 종류 : Assembler (2/2)

- 대부분 two-pass assembler로 구성
 - **pass** : 입력 파일을 한 번 읽으면 one-pass, 두 번 읽으면 two-pass.
 - **first pass** : 식별자(identifier)를 찾아 symbol table에 저장.

식별자 이름	주소
a	0
b	4

- **second pass** : 연산 코드를 기계가 인식할 수 있는 이진수로 표시.

0001 01 00 00000000*	↔	LOAD R ₁ , a
0011 01 10 00000010		ADD R ₁ , #2
0010 01 00 00000100*		STORE b, R ₁

- **Bits 0~3** : 연산 코드. 예: **0001**은 LOAD, 0011은 ADD.
- **Bits 4,5** : 레지스터를 가리킴. 예: **01**은 레지스터 1을 의미
- **Bits 6,7**: Bits 8~15에 대한 주소 지정 방식(addressing mode).
예: **00**: Bits 8~15가 피연산자가 저장된 메모리 주소를 가리킴.
- * : 절대 주소가 아니라 상대 주소임을 나타냄

번역기 종류 : preprocessor

■ 프리프로세서 (전처리기)

- 프로그래밍 언어의 기능을 확장시켜 주는 역할

- 원시 언어와 목적 언어가 모두 고급 언어

- 예: C 프리프로세서

1) 파일 포함 지시어 : 프로그램에 헤더 파일(header file)을 포함

`#include <global.h>` → preprocessing → 해당 코드를 <global.h> 파일로 통째로 바꿈

2) 매크로(macro) : 매크로로 정의한 내용을 확장

- `#define max 45` → preprocessing → 프로그램 전체에서 max 를 모두 45로 바꿈

3) 조건부 컴파일

- 조건에 따라 선택적으로 실행

```
#if SYSTEM == WINDOWS
#include "stdio.h"
#elif SYSTEM == UNIX
#include "unix.h"
#else
#include "etc.h"
#endif
```

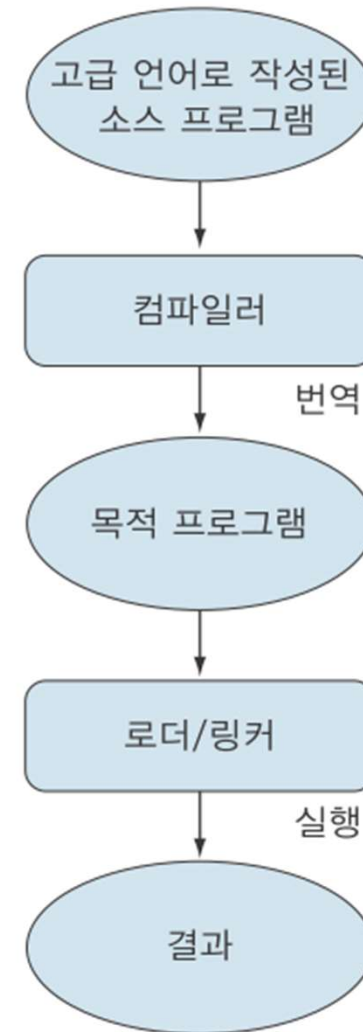
번역기 종류 : 컴파일러

■ 인터프리터와 컴파일러

- 인터프리트 방식은 반복 실행해야 할 때마다 다시 번역
 - 반면, 컴파일 방식은 번역이 끝나면, 다음부터는 번역 없이 이전에 생성한 목적 코드를 재사용할 수 있음.

■ 컴파일 방식

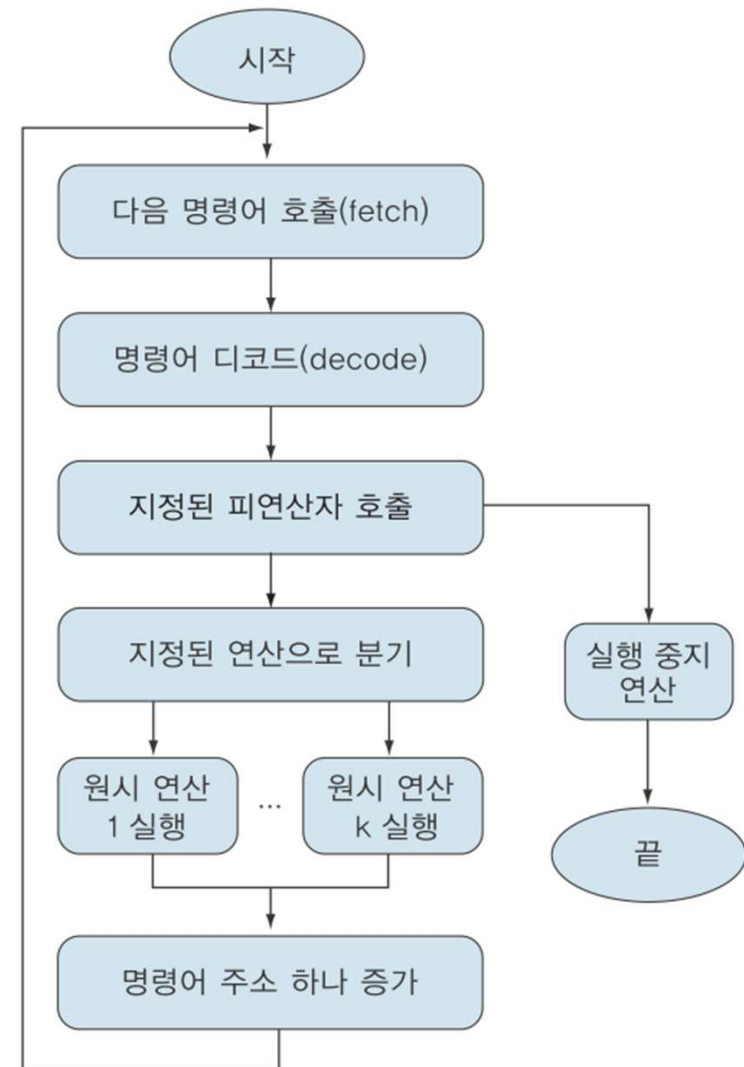
- 반복문이나 자주 호출되는 부프로그램(subroutine)을 실행해야 하는 프로그램에 유리.
- 인터프리트 방식에 비해 **실행 속도가 빠름**.
- 몇 줄의 원시 프로그램이 몇 백 줄의 기계어로 번역되기 때문에 인터프리트 방식에 비해 **메모리가 많이 필요**.



번역기 종류 : interpreter

■ 인터프리터 : 한 줄 단위로 실행

- 컴파일 방식에 비해 메모리 사용을 줄일 수 있다.

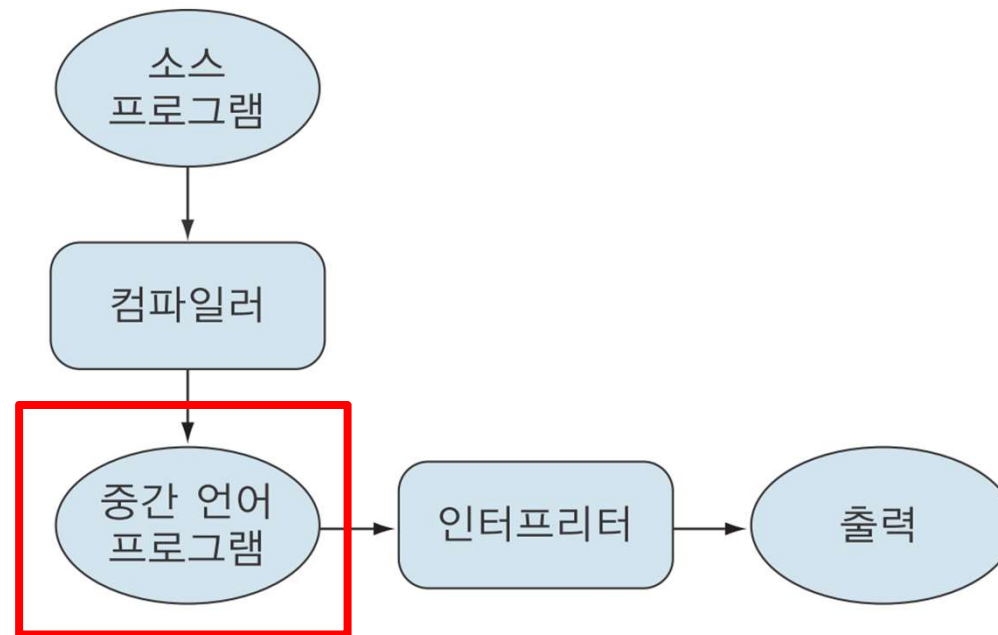


번역기 종류 : 혼합(compiler + interpreter)

■ 혼합형 컴파일러

■ Java

- 자바 소스 프로그램을 바이트코드(bytecode)라 불리는 중간 코드로 컴파일
- 바이트코드는 자바 가상 기계(java virtual machine)에서 인터프리트 방식으로 실행



번역기 종류 : Cross compiler

■ 크로스 컴파일러

- 원시 프로그램을 컴파일러를 실행하고 있는 컴퓨터의 기계어로 번역하는 것이 아니라 다른 컴퓨터의 기계어로 번역.
 - 최신 CPU를 장착할 새 컴퓨터 개발이 아직 끝나지 않았을 때
 - 기존 컴퓨터에서 원시 프로그램을 최신 CPU 기계어 코드로 컴파일하고, 이 기계어 코드를 개발 중인 새 컴퓨터에서 실행

언어 처리 시스템

- **relocatable machine code**

- 메인 메모리 어디에 가져다 놓을지 결정되지 않음

- **absolute machine code**

- 메인 메모리에 loading 완료
→ 메모리 공간, 주소 결정
→ 실행 준비 끝

