프로그래밍언어를 어떻게 만들것인가?
JAVA로실습 : 간단한언어설계 → 인터프리터작성
↳ 자료구조정의가 C보다 낫다! + JAVA로좀더프로그래밍하기위해
(화) 이론, (목) 실습
중간30 + 기말30 + 실습30 + 출석10
↳ 감점으로반영
컴파일러구성론(개체)과 이어짐

프로그래밍 언어론

002분반

숙명여대 창병모
cs.sookmyung.ac.kr/~chang

교재 및 참고자료

- 교재
  - 창병모, 프로그래밍 언어론: 원리과 실제 인피니티북스, 2021.

- 참고문헌
  - Kenneth C. Louden, Programming languages : Principles and Practice, 2nd Edition, Thomson
  - Sebesta, Concepts of Programming languages Pearson, 10th Edition, 2012.
  - [번역판] 유원희, 하상호, 프로그래밍 언어론, 피어슨, 2012

강의 목표

- 프로그래밍 언어의 원리 및 실제
  - 프로그래밍 언어 이론
  - 프로그래밍 언어 설계
  - 프로그래밍 언어 구현 기술 → 인터프리터/컴파일러
  - 주요 프로그래밍 패러다임
  - 프로그래밍 언어의 역사

왜 프로그래밍 언어론을 배울까?

- 프로그래밍 언어에 대한 깊이 있는 이해
  - 이론을 바탕으로 언어에 대한 깊이 있는 이해
  - 새로운 언어의 학습 및 설계 능력 배양
  - 응용에 적절한 언어의 선택

- 언어 처리(language processor) 기술 → 인터프리터/컴파일러 : 언어처리기
  - 언어 처리 및 구현 기술의 이해
  - 자연어 처리, 언어 이해 등에 활용
  - 프로그램 분석 및 소프트웨어 공학 등에 활용

- 컴퓨팅에 대한 전체적인 시각
  - 객체지향 컴퓨팅
  - 인터넷 컴퓨팅 : JAVA
  - 인공지능

1장 서론

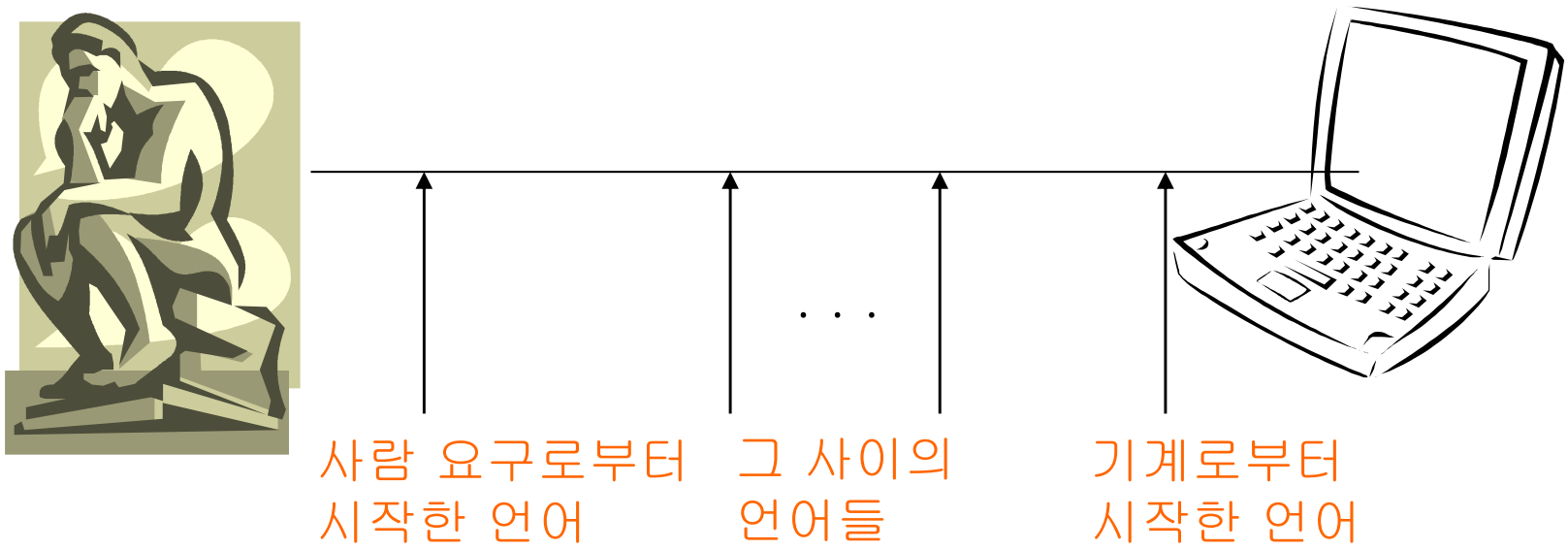
- 프로그래밍 언어란 ?
- 프로그래밍 언어의 종류
- 프로그래밍 언어의 역사
- 추상화와 명령형 언어의 발전
- 프로그래밍 언어의 정의 및 구현

1.1 프로그래밍 언어란?

프로그래밍 언어란 ?

사람 ↔ 기계 커뮤니케이션

- 여러 관점에서 생각해 볼 수 있다.



- 기계를 돌리기 위한 것이 프로그래밍 언어다.
- 사람이 작성한 프로그램을 돌리기 위한 것이 기계이다.

프로그래밍 언어란 무엇인가 ?

- 프로그래밍 언어는 계산 과정을 기계가 읽을 수 있고 사람이 읽을 수 있도록 기술하기 위한 일종의 표기법이다.
- *A programming language is a notational system for describing computation in machine-readable and human-readable form.*

프로그래밍 언어란 무엇인가 ?

- 계산(Computation)
  - 데이터 조작
  - 텍스트 처리
  - 알고리즘
- 기계 읽기(Machine readability)
  - 효율적인 번역 혹은 실행
- 사람 읽기(Human readability)
  - 프로그래밍 편의성
  - 컴퓨터 연산들의 이해하기 쉬운 추상화

프로그래밍 언어의 중요성

- 언어의 구조가 사고의 범위를 지배한다 ?

→ 언어에 따라 프로그래밍 방식이 다름

- 소프트웨어의 발전은 프로그래밍 언어를 매개로 하여 발전 !

- 언어에 따라 사고방식이 달라진다. 뒤에서설명
  - 절차형(명령형) 언어(procedural language) C
  - 함수형 언어(functional language)
  - 논리 언어(logic language)
  - 객체-지향 언어(object-oriented language)

주요 주제

- 주요 언어 이론
  - 구문법(syntax) : 문장작성
  - 의미론(semantics) : 작성된 문장의 의미
  - 타입 시스템(type system)
- 프로그래밍 언어의 설계
  - 동기, 원리
  - 샘플 언어 설계
- 프로그래밍 언어의 구현 기술
  - 주요 구현 기술
  - 샘플 언어의 인터프리터

주요 주제

- 주요 프로그래밍 패러다임 (방법론)
  - 명령형 프로그래밍
  - 객체지향 프로그래밍
  - 함수형 프로그래밍
  - 논리 프로그래밍

- 프로그래밍 언어의 역사
  - 고급 프로그래밍 언어 개발 역사

1.2 프로그래밍 언어 종류

프로그래밍 패러다임

(절차형) (procedural)

- 명령형 프로그래밍(imperative programming)
  - 문제를 해결하는 절차(명령)를 기술하는 방식의 프로그래밍
  - C, Pascal, Ada, Python

chap13

- 함수형 프로그래밍(functional programming)
  - 프로그램의 계산 과정을 수학 함수 형태로 프로그래밍
  - 프로그램은 함수 정의들로 구성됨
  - Lisp, Scheme, ML, Haskell


: 프로그래밍↔함수

프로그래밍 패러다임

교재에 없음! 과학기술대학원전공...

- 논리 프로그래밍(logic programming)

  predicate logic (P이면 Q이다)

  - 정형 논리(formal logic)를 기반으로 한 프로그래밍
  - 프로그램은 문제에 대한 사실/규칙을 표현하는 논리 문장
  - Prolog (programming in logic)

chap11,12

- 객체지향 프로그래밍(object-oriented programming)
  - 객체 개념을 기반으로 하는 프로그래밍
  - 프로그램 실행은 객체 사이의 상호작용에 의해 이루어진다.
  - C++, Java, C#, Objective-C, Swift
  - Python, Visual Basic

절차+객체지향

PL 발전 과정 programming language (high level)

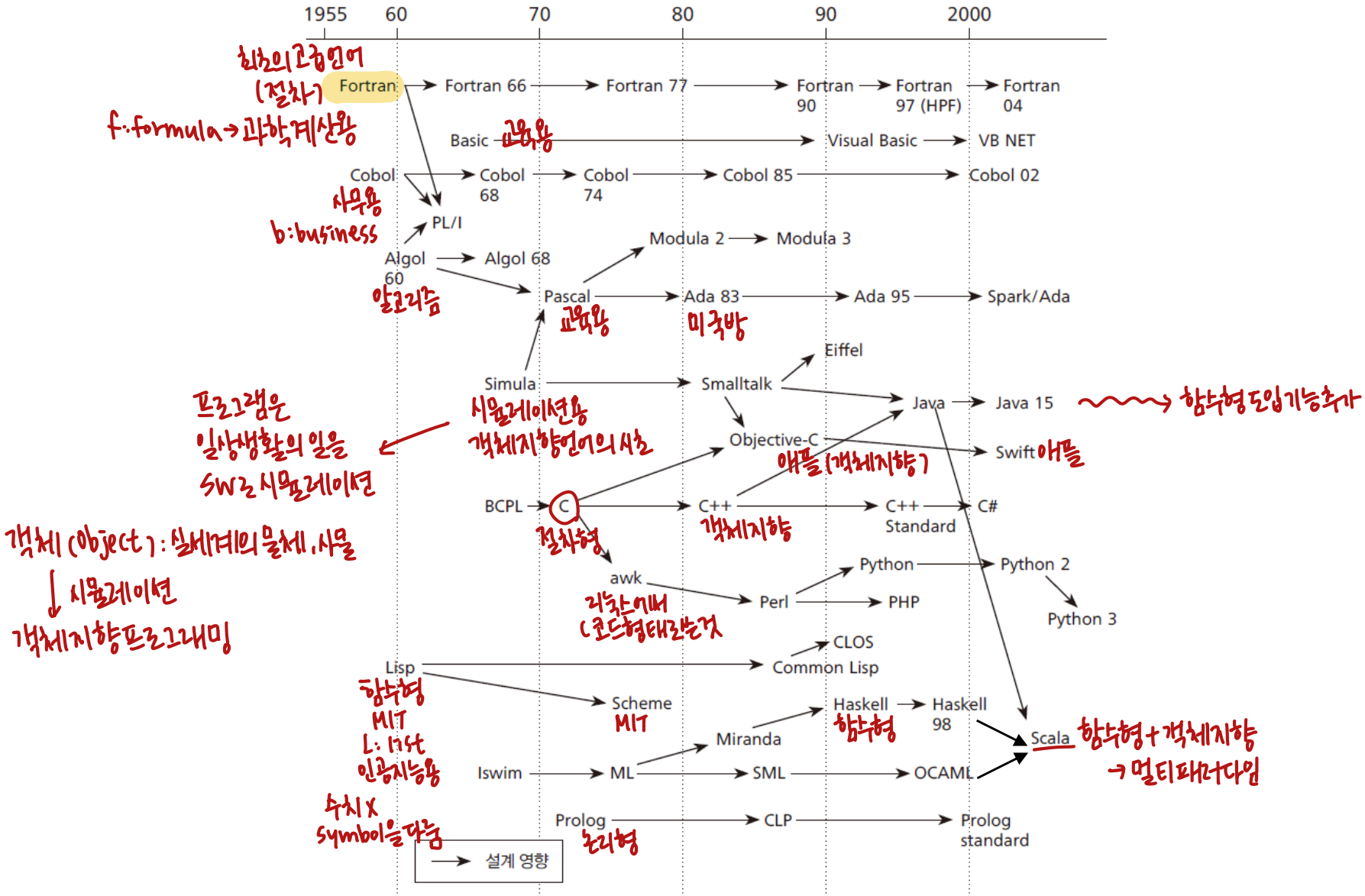
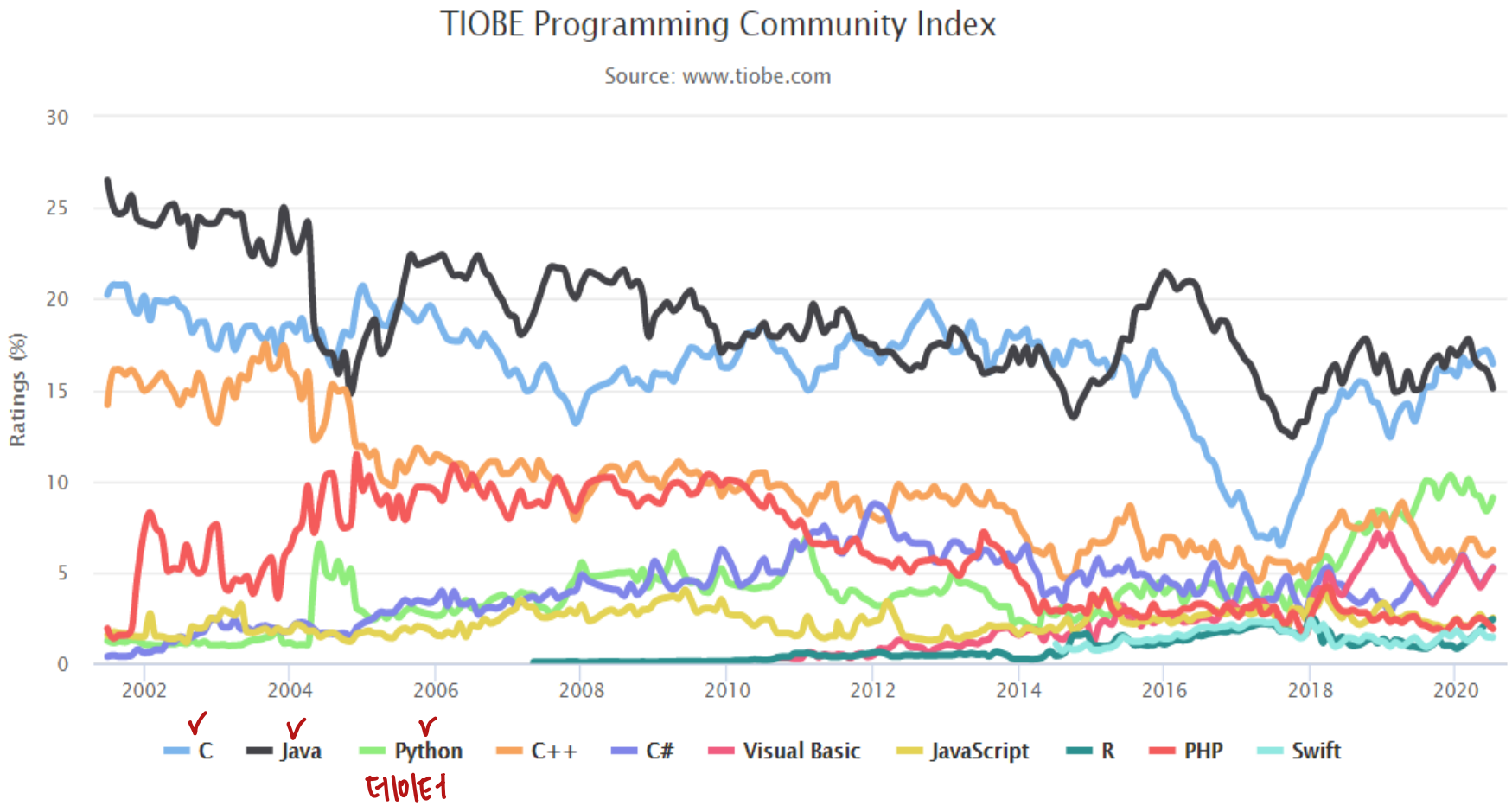


그림 1.2 주요 프로그래밍 언어의 발전 과정

프로그래밍 언어 사용 현황



함수형 언어(Functional language)

상차

- 기본 모델
  - 수학 함수를 기반으로 하는 언어
  - 프로그램

    매개변수로 입력을 받아 처리한 후에 반환값을 출력하는 함수
- 함수
  - 함수 정의(definition)
  - 함수 호출(application)
  - 매개변수(parameter)
  - 반환값(return value)
- 특징
  - 변수 및 대입문이 없음
  - 자기호출(recursion)에 의한 반복 n!을정의하는데에 (n-1)! 이용
  - 루프 같은 반복문 없음

함수형 언어(Functional language)

- 예제

매개변수 ↓ return

```
fun fact(n : int) : int =
    if n <=1 then 1
    else n * fact(n-1);
```

계산된 값이 return 값
언어에 따라 return 이라고 명시하기도 함

- 장점
  - 기계 모델과 무관
  - 수학을 기반으로 프로그램의 의미를 명확하게 정의할 수 있다.
- 예
  - Lisp, ML, Scheme, Haskell

논리 언어(Logic language)

- 기본 모델
  - p ->q 형태의 술어 논리(predicate logic)를 기반으로 함
  - 선언적으로 프로그래밍하는 언어
  - 프로그램
    문제를 해결하는 방법보다 문제가 무엇인지를 논리 문장들로 표현

- 예제 : 팩토리얼이라는 지식을 논리규칙으로 표현 → 인공지능의 초기연구단계

```
fact(0,1).  : 0의 팩토리얼은 1이다.(Q만있음)
fact(N, V) :- N > 0, fact(N-1, V1), V is N * V1.
?- fact(3, X)  : N>0 만족, (N-1)!의 값은 V1 , N!은 N·(N-1)!
                 2>0 만족, ...
```

query

  - 증명하는 것을 계산하는 것으로 간주함.

현재의 인공지능은 학습을 통함.(머신러닝,딥러닝)

논리 언어(Logic language)

- 특징
  - 루프나 선택문 등의 제어 추상화가 없다.
  - 제어는 하부 시스템(해석기)에 의해 제공된다.
  - 변수는 메모리 위치가 아니라 부분 결과 값에 대한 이름이다.

- 장점
  - 기계-독립적이고 정확한 의미구조를 가지고 있다.
  - 선언적 프로그래밍이 가능하다.

- Prolog
  - 대표적인 논리언어
  - 인공지능, 자연어 처리 등의 분야에서 많이 사용됨.

지금은 learning으로함

객체지향 언어(Object-oriented language)

실세계에서 일어나는 일을 프로그램 상에서 시뮬레이션

- 객체지향 언어의 시작
  - 실세계를 모의실험 하기 위한 언어로 고안되었으며
  - 객체 개념을 기반으로 하는 프로그래밍 스타일

- 객체(object)

객체를 프로그램 상에서 어떻게 표현할것인가?
⇒ 클래스 : 객체에대한정의

  - 속성과 관련 행동(함수, 메소드)들의 모음으로 표현

- 계산과정(computation)
  - 객체들 사이의 상호작용(메소드 호출)

- 클래스(class)
  - 클래스는 객체에 대한 타입 정의
  - 객체는 클래스의 한 실체(instance)이다.

클래스 ← 정의 — 실체 → 객체

클래스와연관시켜서 말한객체

1.3 프로그래밍 언어의 역사

1950년대: 고급 프로그래밍 언어의 시작

IBM
- FORTRAN(FORmula TRANslation)
  - 과학응용 분야를 위한 효율성을 강조한 최초의 고급 언어
  - 설계 목표 : 매우 빠르게 실행되는 코드를 생성함. 컴퓨터가 비쌌기 때문
  - 주요 기능: 배열(array), FOR 반복문, 분기 IF-문 등

행렬(2차원배열) 배열의 원소에 access

- COBOL(COmmon Business-Oriented Language)
  - 사무용으로 설계된, 영어와 비슷한 구문을 갖는 명령형 언어
  - 주요 기능: 레코드 구조, 프로그램의 실행부와 분리된 자료구조
  - 다양한 출력 기능 등

C의 구조체 예) 학생기록카드

MIT
- LISP(List Processor)
  - 리스트 자료구조와 함수 적용을 기반으로 함.
  - 재귀호출(recursive call)이 매우 일반적임.
  - LISP와 후속 언어 Scheme은 인공지능 분야에서 많이 사용됨

1960년대: 프로그래밍 언어의 다양성

- Algol60/68
  - 알고리즘을 기술하기 위한 강력한 범용 언어
  - Pascal, C, Modula-2, Ada 같은 현대의 명령형 언어에 영향을 줌.
  - 주요 특징
    - 구조적 문장, begin-end 블록,
    - 자유 양식(free format), 변수의 타입 선언,
    - 재귀 호출, 값 전달 매개변수
- Algol68
  - Algol60을 향상하여 더 표현력 있고 이론적으로 완전히 일관성 있는 구조를 생성하려고 함.

1960년대: 프로그래밍 언어의 다양성

- PL/I
  - 일반적이고 보편적인 언어, 즉 모든 언어를 통합하는 언어
    - FORTRAN, COBOL, Algol60의 가장 좋은 특징을 모두 결합하고
    - 또한 병행성과 예외처리 기능 등을 추가
  - 배우기도 어렵고 사용하는 데 오류가 발생이 많음.
    - 너무 복잡해서 언어 기능들 사이에 예측할 수 없는 상호작용이 많음.

- Simula-67
  - 최초의 객체지향 언어로 모의실험(simulation)을 위해 설계됨
  - 객체와 클래스 개념을 소개함으로써 공헌함.

- BASIC
  - 단순한 언어로 PC로 이전되어 교육용 언어로 많이 사용됨
  - 이후 마이크로소프트사에 의해 Visual Basic 형태로 발전됨.

1970 년대 : 단순성 및 새로운 언어의 추구

- PASCAL
  - 교육용 언어로 Algol의 아이디어를 작고, 단순하고, 효율적이고, 구조화된 언어로 세련되게 만듬.
  - 대표적인 블록 구조 언어

- C 언어
  - 유닉스 운영체제 개발을 위해 개발된 시스템 프로그래밍 언어
  - 기계에 대해 많은 접근을 제공하는 중급 언어(middle-level)
  - 모든 컴퓨터 시스템에서 사용할 수 있도록 설계된 언어

1970 년대 : 단순성 및 새로운 언어의 추구

- Prolog : program + logic
  - 술어 논리를 사용하는 대표적인 논리 프로그래밍 언어
  - 증명하는 것을 계산하는 것으로 간주함.
  - 인공지능, 자연어 처리 등의 분야에서 많이 사용됨.

- Scheme
  - 더 형식적이고 람다 계산에 더 가깝게 설계된 향상된 LISP 버전

- ML
  - Pascal과 가까운 구문을 가지고
  - 다형 타입 검사 메커니즘을 제공하는 함수형 언어

1980년대: 추상 자료형과 객체지향

- Ada
  - 미 국방성(DoD)의 후원으로 개발된 영향력 있고 포괄적인 언어
  - 주요 기능
    - 패키지(추상 자료형), 태스크(병행 프로그래밍 기능),
    - 예외처리 등과 같은 새로운 기능을 포함

- Modula-2
  - 범용 절차형 언어이면서 시스템 프로그래밍 언어로 개발
  - Pascal처럼 가능한 한 작고 단순한 언어로 설계
  - 주요 기능
    - 모듈(추상 자료형), 코루틴(부분적인 병행성)

1980년대: 추상 자료형과 객체지향

- Smalltalk
  - 순수한 객체지향 언어
  - Ruby, Objective-C, Java, Python, Scala 등의 언어에 영향을 줌
  - 최초로 GUI를 제공하는 언어

- C++
  - C 언어를 확장함 특히 C 언어의 구조체를 클래스 형태로 확장
  - C 언어의 효율성을 유지하면서도 객체지향 프로그래밍 가능
  - 포인터와 같은 C 언어의 중요한 특징을 그대로 포함하고 있음.

1990년대 : 인터넷 언어와 새로운 시도

- Python
  - 대화형 인터프리터 방식의 프로그래밍 언어
  - 플랫폼 독립성, 객체지향, 동적 타입(dynamic type)
  - 교육용 및 빅데이터를 비롯한 다양한 분야에서 응용되고 있음.
- Java
  - 인터넷 환경을 위한 객체지향 언어
  - 웹 애플리케이션, 모바일 앱 개발 등에 가장 많이 사용하는 언어
  - 플랫폼 독립성
    - 컴파일된 바이트 코드가 JVM이 설치된 어느 플랫폼에서도 실행 가능
- JavaScript
  - 웹 브라우저 내에서 실행되는 클라이언트 프로그램에 주로 사용
  - Node.js와 같은 런타임 환경과 같이 서버 프로그래밍에도 사용

2000년대 : 새로운 미래를 향하여

- C#
  - Java를 모방한 마이크로소프트 버전
  - 닷넷 프레임워크을 기반으로 함.

- Scala
  - 객체지향과 함수형 언어의 요소가 결합된 다중패러다임 언어
  - 자바 바이트코드를 사용하기 때문에 JVM에서 실행 가능
  - Java 언어와 호환: 대부분의 자바 API를 그대로 사용 가능

- Objective-C와 Swift
  - Swift는 기존의 Mac 용 언어인 Objective-C와 함께 공존
  - Objective-C처럼 LLVM으로 빌드되고 같은 런타임 시스템을 공유
  - 특징: 클로저, 다중 리턴 타입, 네임스페이스, 제네릭, 타입 유추

1.4 추상화와 명령형 언어의 발전

abstraction

프로그래밍 언어 발전 과정

- 역사적 발전 과정
  - 최초의 컴퓨터(ENIAC)가 만들어지면서
  - 프로그래밍 언어가 개발되기 시작했을 것이다.

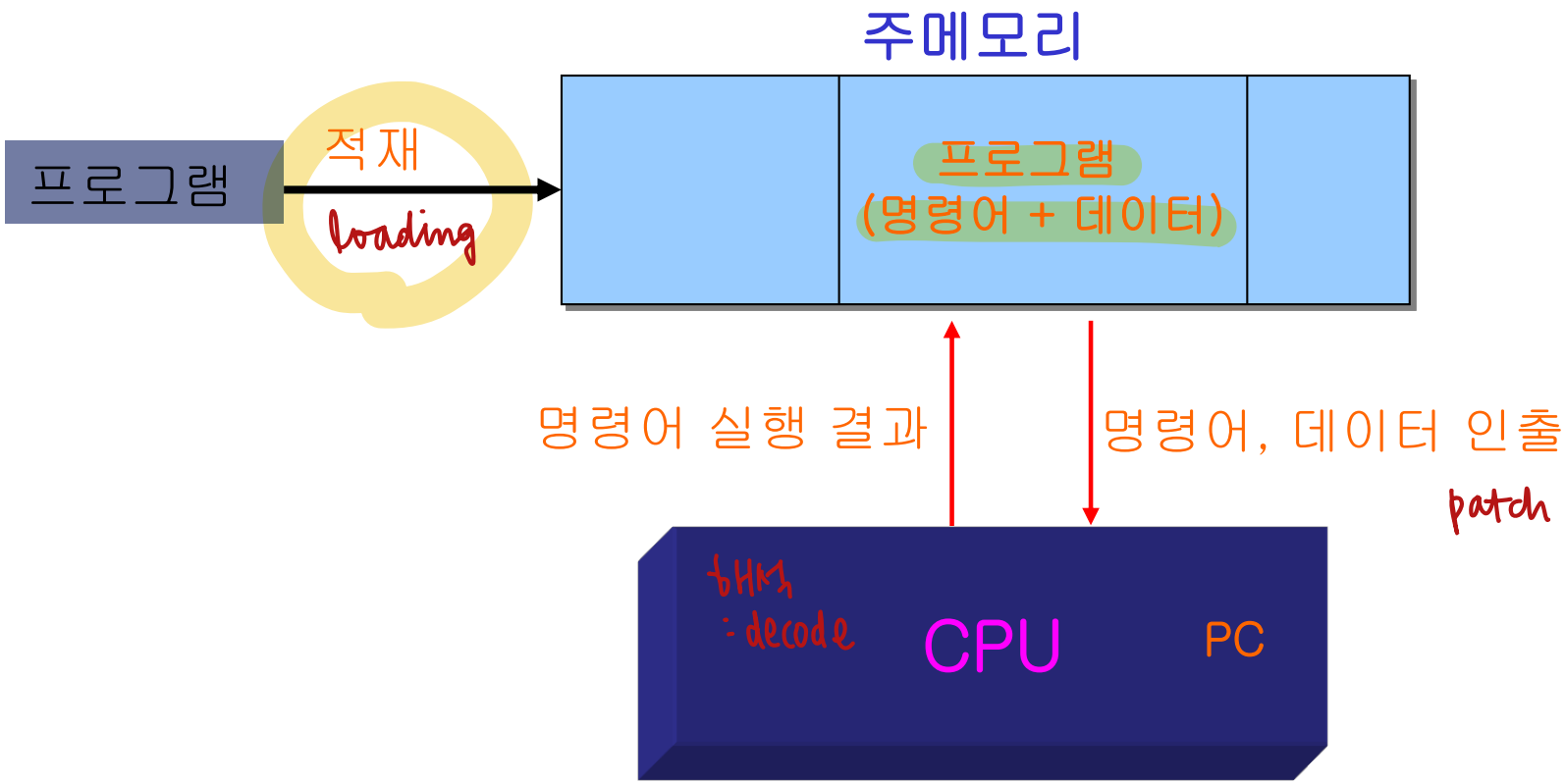
- 두 가지 질문?
  - 그때 컴퓨터는 어떤 컴퓨터였을까요?
  - 어떤 프로그래밍 언어가 개발되었을까요?

- 컴퓨터
  - Von Neuman model computer
  - 폰 노이만 모델 컴퓨터

- 초창기 프로그램
  - 컴퓨터에 명령하는 기계어 명령어들로 작성

Von Neuman 모델 컴퓨터

폰노이만



Von Neuman 모델 컴퓨터

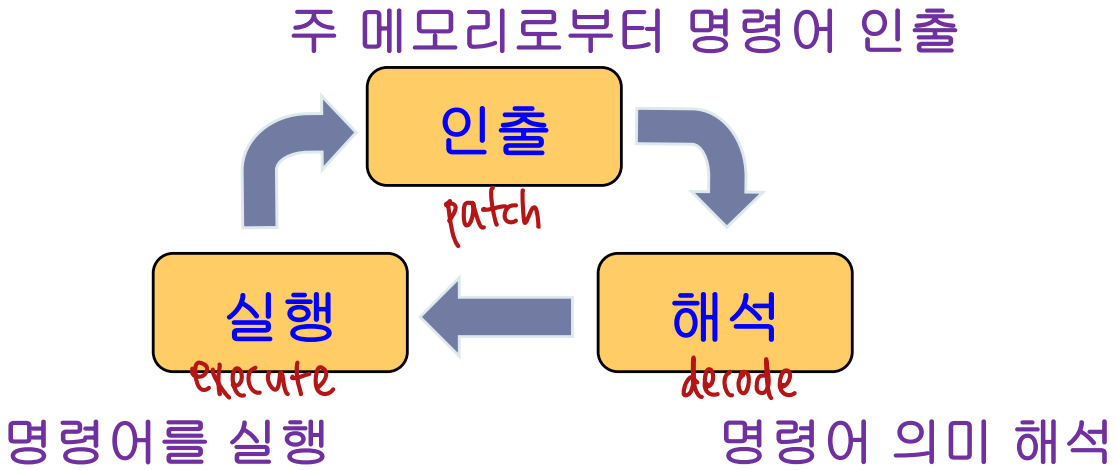
- 프로그램 내장 방식 컴퓨터
  - stored-program computer → 어떤 프로그램이 load 되느냐에 따라 다른 목적으로 사용가능해짐!
  - 메모리에 프로그램(명령어와 데이터) 저장 (load)

- 메모리에 저장된 명령어 순차 실행
  - CPU sequentially execute instructions in memory
  - PC (program counter)
    다음 실행할 명령어를 가리키는 레지스터

- 명령어
  - 메모리에 저장된 값을 조작 혹은 연산
  - instructions operate on values stored in memory

Von Neuman 컴퓨터 프로그램 실행

- CPU의 인출-해석-실행(fetch-decode-execute) 주기 반복
  - CPU는 메모리 내에 저장되어 있는 프로그램의 명령어를
  - 한 번에 하나씩 가져와서 해석하고 실행한다.



명령형 언어(Imperative language)

- 명령형 언어의 발전

  폰노이만 모델 컴퓨터를 흉내내고 요약(추상화)

  Imperative programming languages began by imitating and abstracting the operations of von Neuman model computer

- 예
  - **Fortran, C, Basic, …**

- 폰 노이만 모델 컴퓨터의 특징을 많이 갖고 있음!
  - 순차적 명령어 실행
  - 메모리 위치를 나타내는 변수 사용
  - 대입문을 사용한 변수 값 변경
  - 사람의 필요보다는 컴퓨터 모델을 기반으로 한 언어

추상화(Abstraction)

- 추상화(Abstraction) ?
  - 추상화는 실제적이고 구체적인 개념들을 요약하여
  - 보다 높은 수준의 개념을 유도하는 과정이다.

- 명령형 언어는 컴퓨터의 무엇을 추상화 했을까요?
  - 컴퓨터의 데이터, 연산, 명령어 등을 추상화
  - 데이터 추상화
  - 제어 추상화

데이터 추상화(Data Abstraction)

- 기본 추상화
  - 기본 데이터 관련한 요약
  - 예: 변수, 자료형

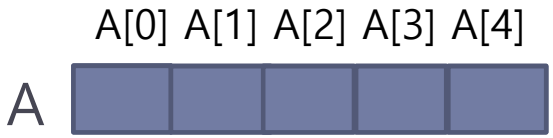
- 변수(variable) 메모리주소를추상화
  - 데이터 값을 저장하는 메모리 위치
  - 메모리 120 번지 => 변수 x

- 자료형(data type)
  - 값들의 종류에 대한 이름
  - 예: int, float, double, ...
    ㄴ선언

데이터 추상화 → 개념제공

프로그래밍을 추상화로 정의된 개념을 이해하고 ~ …

- 구조적 추상화
  - 관련된 여러 값/변수들의 모음을 요약
- 배열
  - 같은 타입의 연속된 변수들의 모음



- 레코드(구조체)
  - 다른 타입의 연관된 변수들의 모음

```
struct Employee
{    int age;
     char name[10];
     float salary;
}
```

- 프로그래밍 언어와 추상화
  - 프로그래밍 언어는 이러한 추상화된 개념을 제공하고
  - 프로그래머는 이러한 개념을 기반으로 프로그래밍 한다.

제어(Control) : 명령어의 추상화 == 제어 추상화

- QnA #1
  - 제어(control) 무엇을 제어한다는 말인가요? → 실행순서를 control
  - Control flow ?

- QnA #2
  - 제어와 관련된 문장들은 어떤 것들이 있을까요?

    → if, while, for, …

제어 추상화(Control Abstraction)

- 기본 추상화
  - 몇 개의 기계어 명령어들을 하나의 문장으로 요약
  - 대입문

X = X+3

추상화 ←

```
LOAD R1, X
ADD R1, 3
STORE R1, X
```

  - goto 문 : 몇번지로가라
    - jump 명령어의 요약

제어 추상화

- 구조적 제어 추상화
  - 테스트 내의 중첩된 기계어 명령어들을 하나의 문장으로 요약

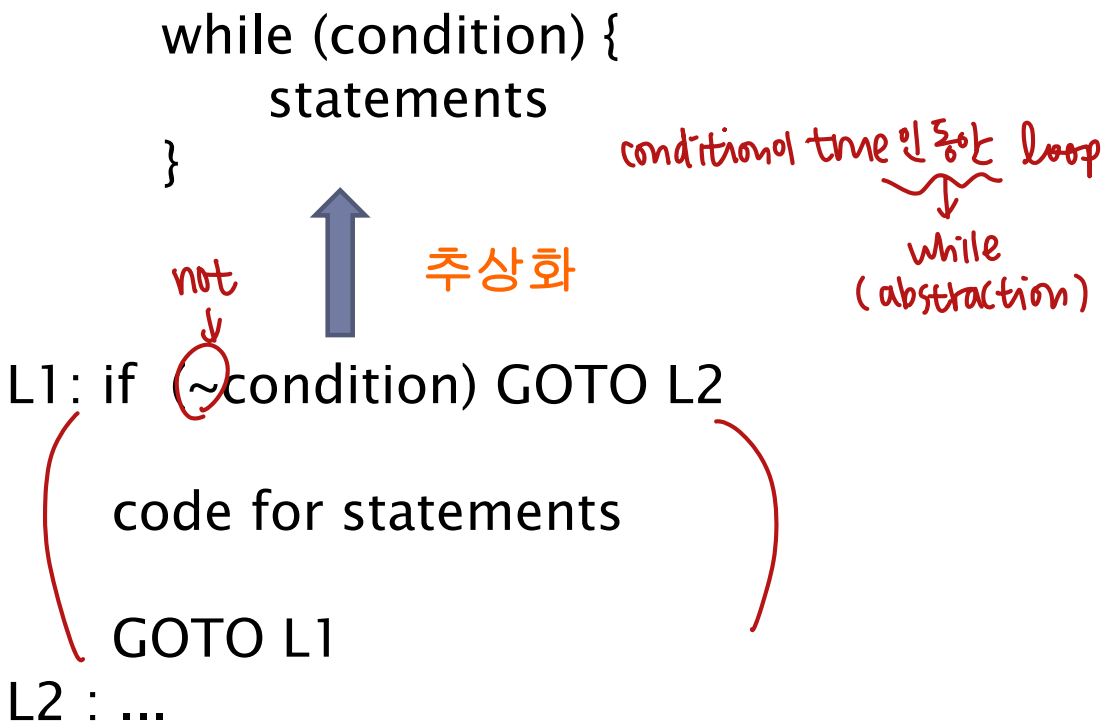
- 예 명령어 추상화
  - if-문, switch-문
  - for-문, while-문 등

ex) 팩토리얼 계산 프로그램

```
read x;
y = 1;
while (x != 1) {
      y = y*x; x = x-1
}
```

제어 추상화

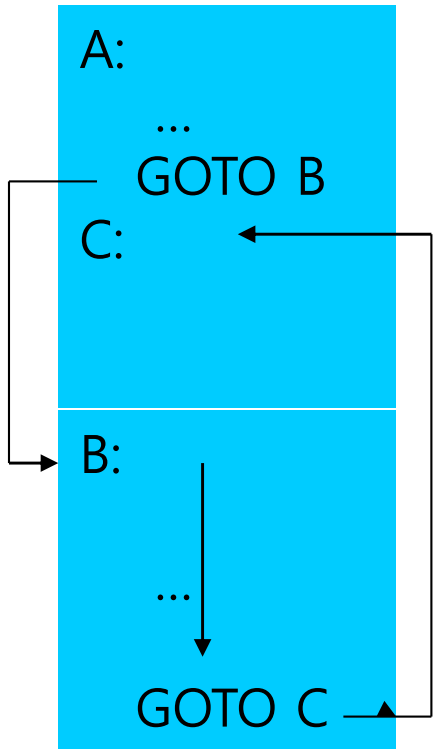
- 예

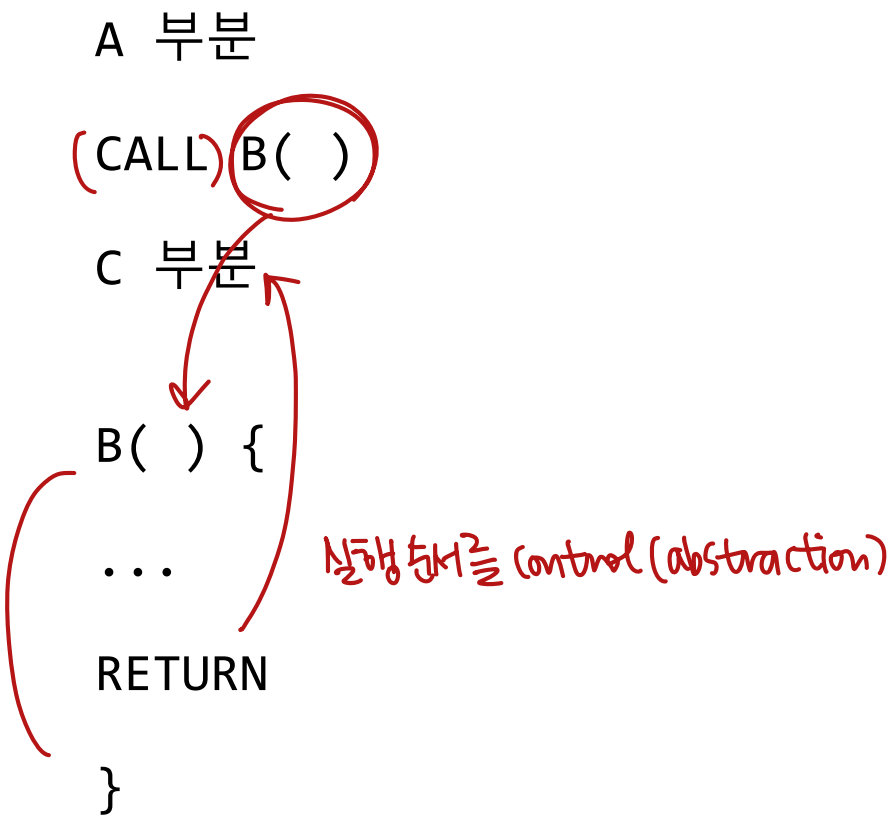


- 장점
  - 기계에 대한 추상화(요약된) 관점
  - 다른 제어 문장들과 중첩되어 사용될 수 있다.

제어 추상화

- 프로시저(함수, 메쏘드)
  - 선언 : 일련의 계산 과정을 하나의 이름으로 요약해서 정의
  - 호출 : 이름과 실 매개변수를 이용하여 호출





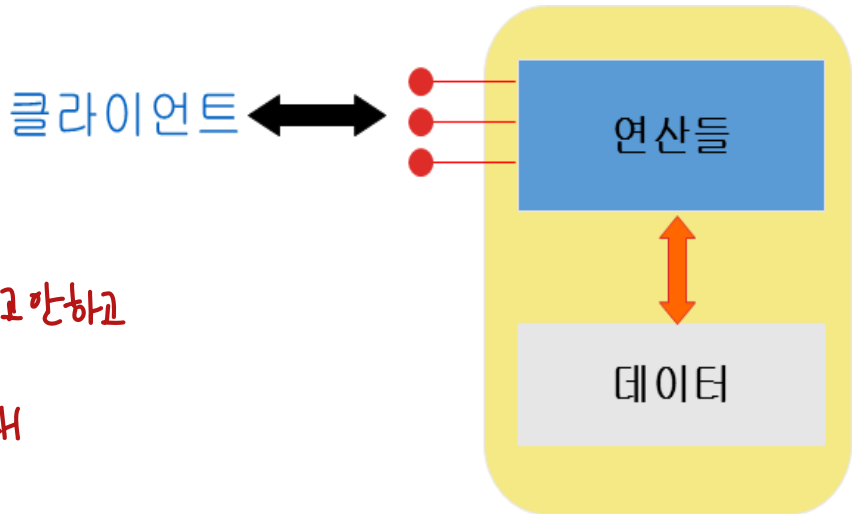
"B로 가서 할 일하고 돌아와"

추상 자료형(Abstract Data Type)

- 추상 자료형 ↗ Stack
  - (데이터 + 관련 연산)
  - 데이터와 관련된 연산들을 캡슐화하여 정의한 자료형

- 예
  - Modula-2의 모듈
  - Ada의 패키지
  - C++, Java 등의 클래스

↓
Abstract data type 이라고 안하고 그냥 class라고함
∵ 클래스에는 "상속" 개념존재



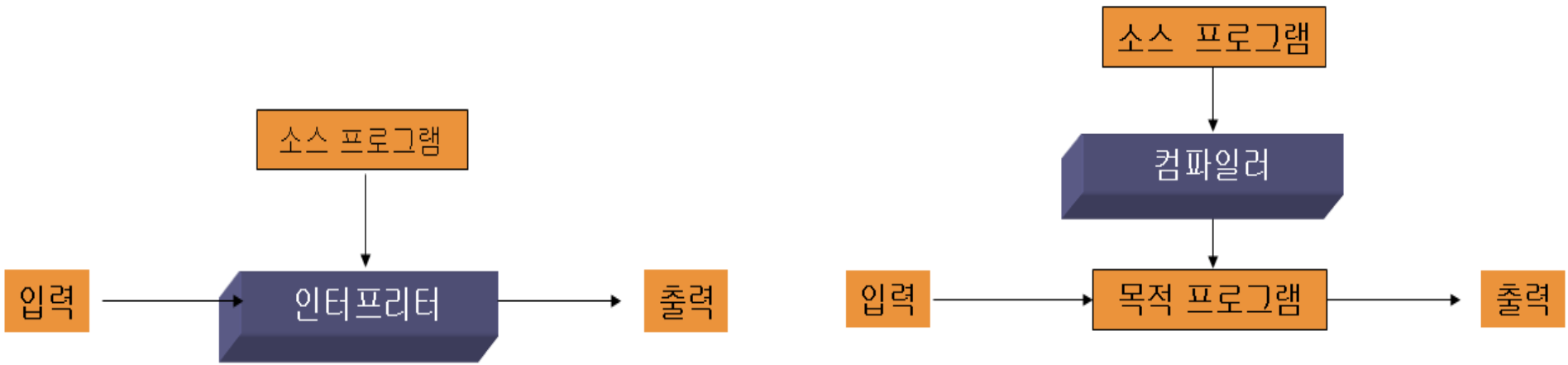
1.5 프로그래밍 언어의 정의 및 구현

(프로그래밍) 언어 정의

- 어휘구조(lexical structure)
  - 언어에서 사용하는 단어의 구조, 철자법 등을 의미한다.
- 구문법(Syntax)
  - 구성요소를 이용하여 문장/프로그램을 구성하는 방법
  - 문법을 이용하여 기술할 수 있다.
  - Context-free grammar in BNF(Backus-Naur Form)
- 의미론(Semantics)
  - 문장/프로그램의 의미를 정하는 것
  - 자연어 혹은 수학적으로 기술:
- 프로그래밍 언어 공부하는데 필요한 기본기

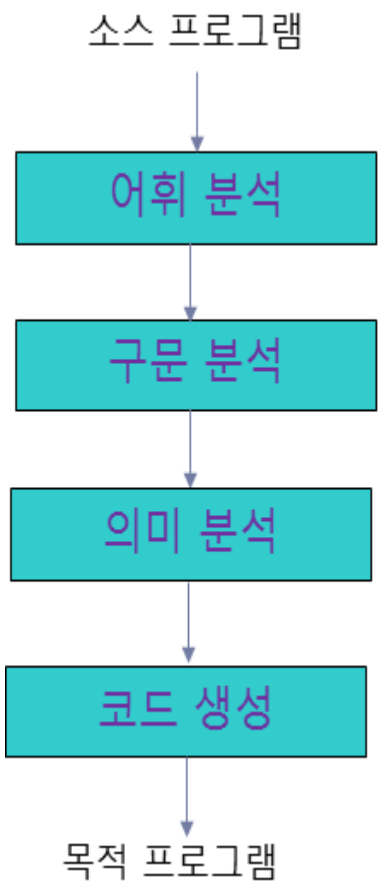
프로그래밍 언어 구현

- 프로그래밍 언어 구현은 무얼 해야 하나요?
  - 입력 프로그램 → Syntax → Semantics → Interpret/Compile
    검사 파악
- 프로그래밍 언어 구현은 어떻게 해야 하나요?
  - 입력 받은 소스 프로그램을 구문법에 맞는지 검사하고
  - 그 의미에 맞게 동작하도록 해석하거나
  - 기계어 명령어들로 번역해야 한다.



컴파일러와 인터프리터의 내부 구조

- 컴파일러



인터프리터