프로그래밍 언어론

2nd edition

Tucker and Noonan

1장
소개

좋은 프로그래밍 언어는 프로그램에 대하여 사고하는 개념적인 세계를 제공해준다.

앨런 펄리스**(Allen Perlis)**

소목차

1.1 원리

프로그래밍 언어의 구성 요소:

- 구문구조Syntax
- 이름Names
- 타입Types
- 의미구조Semantics

프로그래밍 언어:

- 언어 설계자는 이 구성요소를 모두 정의해야 함
- 프로그래머는 이 구성 요소에 모두 정통해야 함

구문구조(Syntax)

프로그래밍언어의 **구문구조**는 문법적으로 맞는 프로그램이 어떻게 생겼는지를 서술한다.

구문구조를 공부할 때 생기는 궁금증

- 언어의 문법은 무엇일까?
- 사용할 수 있는 기본 어휘는 무엇일까?
- 구문 오류는 어떤 것일까?

이름(Names)

프로그램에서 다양한 종류의 엔티티에 이름 부여

- 변수, 타입, 함수, 매개변수, 클래스, 객체 등

엔티티에 부여된 이름은 프로그램이 실행되는 동안 다음에 묶여있다

- 영역Scope
- 가시성Visibility
- 타입Type
- 생명주기Lifetime

타입(Types)

타입은 값들과 그 값을 가지고 할 수 있는 연산들의 집합이다.

- 단순 타입
  - 수numbers, 문자characters, 불booleans, …
- 구조 타입
  - 문자열Strings, 리스트lists, 트리trees, 해시테이블hash tables, …
- 타입시스템type systems이 제공해주는 이점
  - 허용하는 연산을 결정
  - 타입 오류 감지

의미 구조(Semantics)

프로그램의 의미를 의미구조로 표현한다.

의미구조를 공부하면 다음과 같은 질문에 답할 수 있다.

- 프로그램 실행 중 큭정 변수의 값은 어떻게 변하나?
- 각 문장은 어떤 뜻일까?
- 함수 호출을 하면 일어나는 현상을 설명해줄 수 있는 모델이 있을까?
- 실행 중에 객체는 어떻게 메모리에 할당될까?

1.2 패러다임(Paradigm)

프로그래밍 팰다임은 특정 장르의 프로그램과 언어에 깔려있는 문제 해결 사고의 패턴이다.

대표적인 4가지 프로그래밍 패러다임

- 명령형 Imperative
- 객체지향형 Object-oriented
- 함수형 Functional
- 논리형 Logic (선언형 declarative)

명령형 패러다임

전통적인 폰노이만-엑커르트 계산 모델을 따름:

- 프로그램과 데이터가 메모리에 구별없이 저장됨
- 프로그램 = 명령문의 나열
- 상태State = 프로그램 실행 시, 변수의 값
- 프로그램이 길어지면 프로시저 추상화procedural abstraction

명령형 언어의 사례:

- *Cobol, Fortran, C, Ada, Perl, ...*

폰노이만-엑커르트 모델 The von Neumann-Eckert

Model

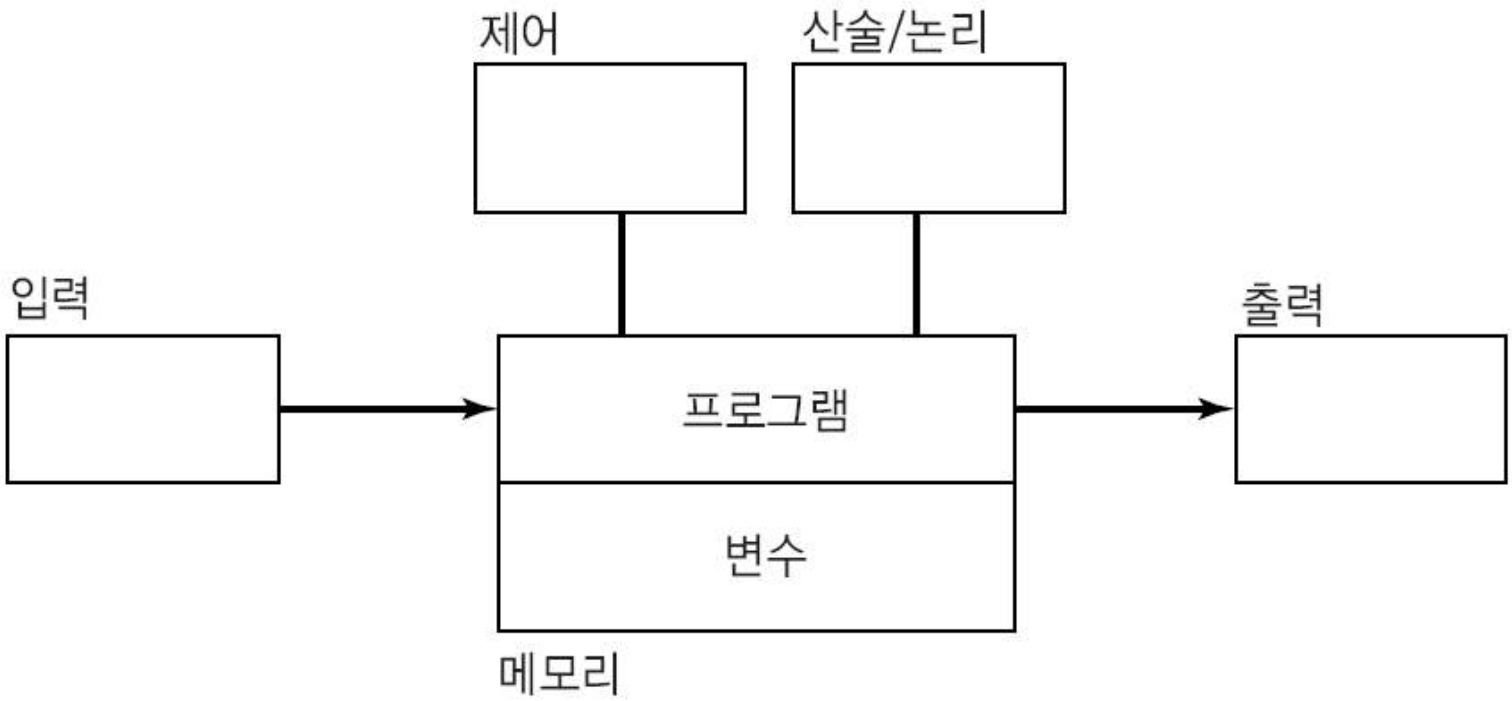

그림 1.1 폰노이만-엑커르트 컴퓨터 모델

객체지향(OO) 패러다임

OO 프로그램은 상태를 변환시키는 메시지를 주고받으며 상호 교류하는 객체의 모임이다.

OO를 공부하면서 익히는 개념

- 메시지 전달Sending Messages
- 상속Inheritance
- 다형성Polymorphism

OO 언어의 사례

Smalltalk, Java, C++, C#, Python

함수형 패러다임

함수형 프로그래밍은 계산 문제를 수학 함수의 집합으로 모델링한다.

- 입력Input = 정의역domain
- 출력Output = 치역range

함수형 언어의 특징

- 함수 합성Functional composition
- 재귀Recursion

함수형 언어의 사례

- *Lisp, Scheme, ML, Haskell, ...*

논리형 패러다임

논리형 프로그래밍은

- 어떻게 문제를 풀 것인지를 고민하는 대신,
- 프로그램에서 얻어내야 하는 결과가 무엇인지를 선언하도록 문제를 모델링하여 프로그램을 작성한다.

논리형 프로그램을 공부하면서 관찰할 사항

- 프로그램을 문제에 대한 제한식의 집합으로 취급
- 프로그램은 문제의 가능한 모든 답을 만들어 냄
- 비결정적nondeterministic인 프로그램 작성 가능

논리형 프로그래밍 언어의 사례

- *Prolog*

1.3 주요 개념

- 이벤트 처리Event handling
  - 예: GUI, 홈 시큐리티 경보 시스템
- 동시 계산Concurrency
  - 예: 클라이언트-서버 프로그램
- 프로그램의 정확성Correctness
  - 프로그램이 모든 가능한 상황에서 기획한 대로 작동하는지 어떻게 증명할 수 있을까?
  - 이게 왜 중요하지???

1.4 역사

언제 어떻게 프로그래밍 언어가 진화되었나?

어떤 집단이 개발하고 사용했는가?

- 인공지능
- 컴퓨터과학 교육
- 과학과 공학
- 정보 시스템
- 시스템과 네트워크
- 월드 와이드 웹World Wide Web

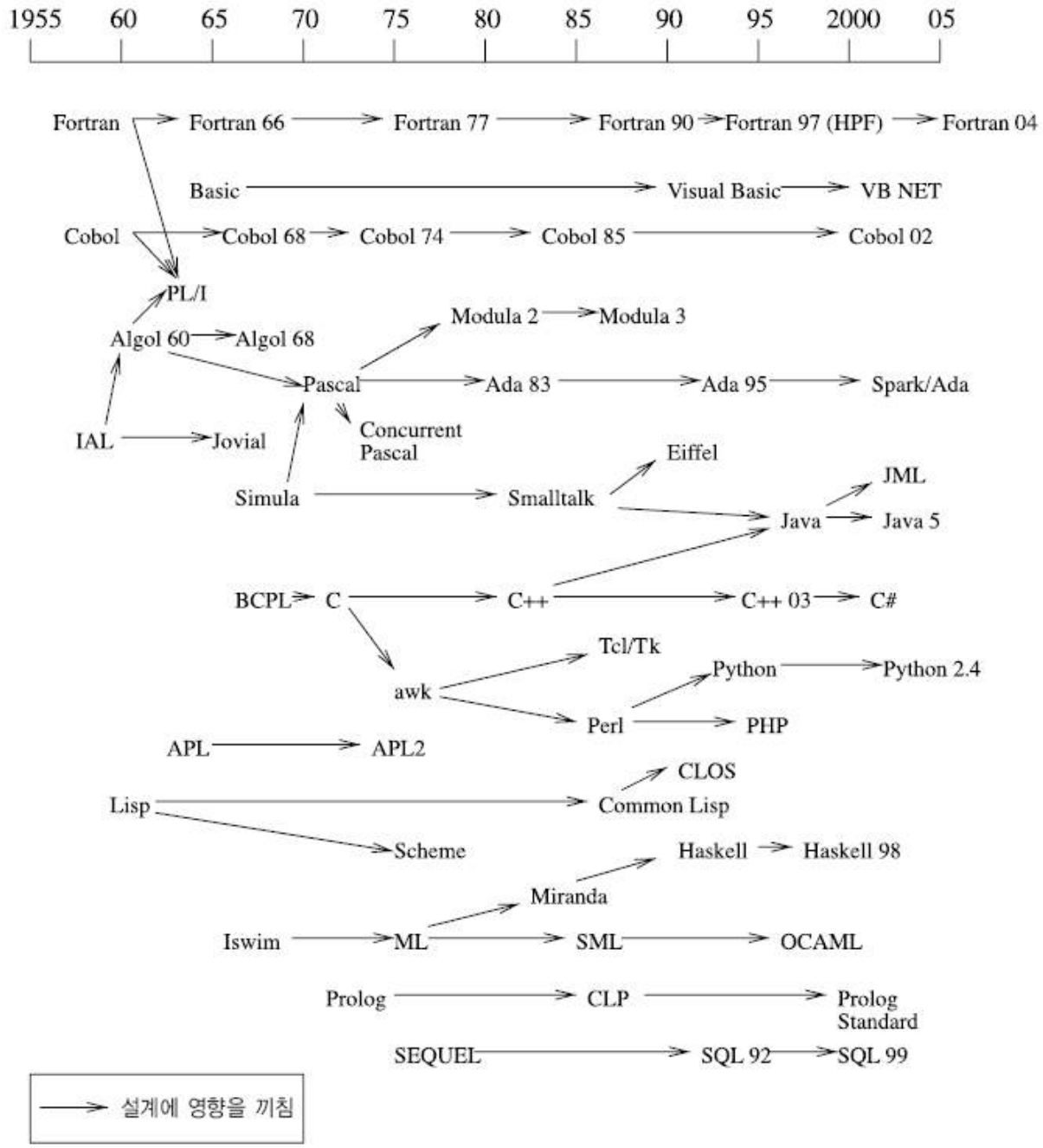

그림 1.2 프로그래밍 언어 변천사의 스냅 사진

1.5 언어의 설계에 대하여

설계 제약사항

- 컴퓨터 구조
- 기술적인 환경
- 표준
- 유물 시스템

설계 결과와 목표

- 프로그래밍 언어는 어떻게 생겨나고 어떻게 하면 성공하는가?
- 이상적인 프로그래밍 언어가 되려면 어떤 주요 특징이 필요한가?

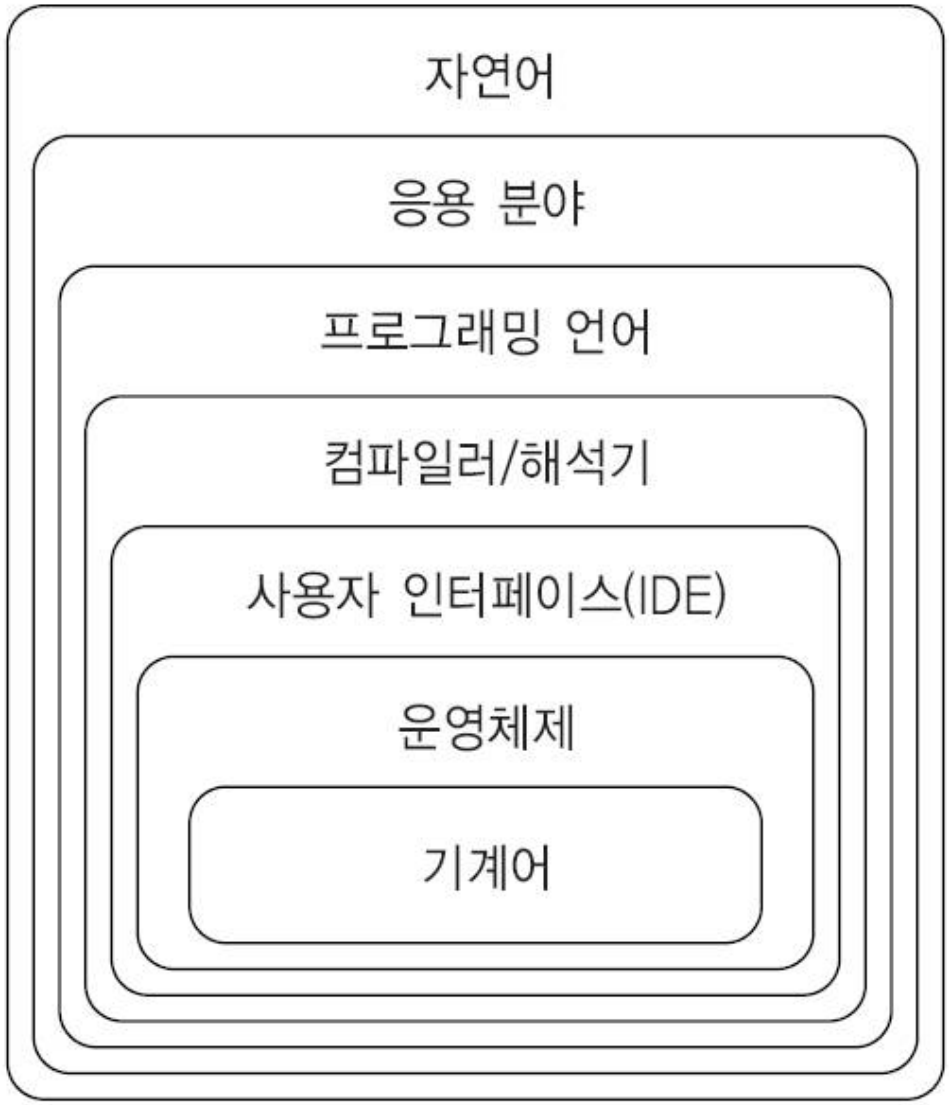

그림 1.3 컴퓨팅에서 추상화의 수준

어떻게 하면 성공적인 언어가 되는가?

주요 특징

- 단순성과 가독성
- 묶기binding의 명확성
- 신뢰성
- 지원성
- 추상화Abstraction
- 직교성Orthogonality
- 구현의 효율성

단순성과 가독성

- 명령문 개수의 소규모화
  - 예: Java vs Scheme
- 구문구조의 단순함
  - 예: C/C++/Java vs Python
- 이점
  - 배우기 쉬움
  - 프로그램 작성이 쉬움

묶기의 명확성

언어에서 필요한 요소들은 그 특성이 정의될 때 묶인다.

묶기binding는 어떤 개체와 그 개체의 성질을 결합하는 것이다.

- 예:
  - 변수와 그 변수의 타입
  - 변수와 그 변수의 값
- 조기 묶기는 프로그램 컴파일 시간에 일어남
- 만기 묶기는 프로그램 실행 시간에 일어남

신뢰성

언어를 신뢰할 수 있는 충분조건

- *다른 플랫폼에서 실행해도 같은 결과나 나옴*
  - 예: Fortran 초기 버전
- *타입오류를 모두 감지할 수 있음*
  - 예: C vs Haskell
- *의미 오류를 적절히 예외처리 할 수 있음*
  - 예: C vs C++
- *메모리 누수를 방지할 수 있음*
  - 예: C vs Java

지원성

- 공개 소프트웨어로 쉽게 구하여 사용할 수 있는 컴파일러와 인터프리터가 존재
- 양질의 교재와 지침서 존재
- 활성화된 사용자 커뮤니티의 존재
- 통합개발환경(IDE)의 존재

추상화

- 데이터
  - 프로그래머 정의 타입 및 클래스
  - 클래스 라이브러리
- 프로시저
  - 프로그래머 정의 함수
  - 표준 함수 라이브러리

직교성

상호 독립된 소규모 기본 연산만으로 프로그램을 구축할 수 있는 언어를 직교적orthogonal이라고 한다.

- 필요한 예외적 규칙의 최소화 = 개념적으로 단순
  - 예: 함수 인수의 타입을 제한함
- 효율성과 타협의 여지
  - 직교성이 좋아지면 효율성이 낮아질 수 있고,
  - 효율성이 좋아지면 직교성이 낮아질 수 있다.

구현의 효율성

- 내장형 시스템Embedded systems
  - 실시간 응답 속도 (예: 네비게이션)
  - 초기 Ada 언어 구현의 실패 사례
- 웹 응용프로그램
  - 사용자의 요구에 대한 응답 속도 (예: 구글 검색)
- 전사적 데이터베니스 응용프로그램
  - 효율적인 검색 및 수정
- AI 응용프로그램
  - 인간의 행동을 모델링

1.6 컴파일러와 가상기계

컴파일러Compiler – 기계코드를 생성

실행기(해석기)Interpreter – 가상 기계의 명령을 실행

- 컴파일하는 언어의 예
  - *Fortran, Cobol, C, C++*
- 실행기를 사용하는 언어의 예
  - *Scheme, Haskell, Python*
- 컴파일러와 실행기를 모두 사용하는 경우
  - *The Java Virtual Machine (JVM)*

컴파일 실행 과정

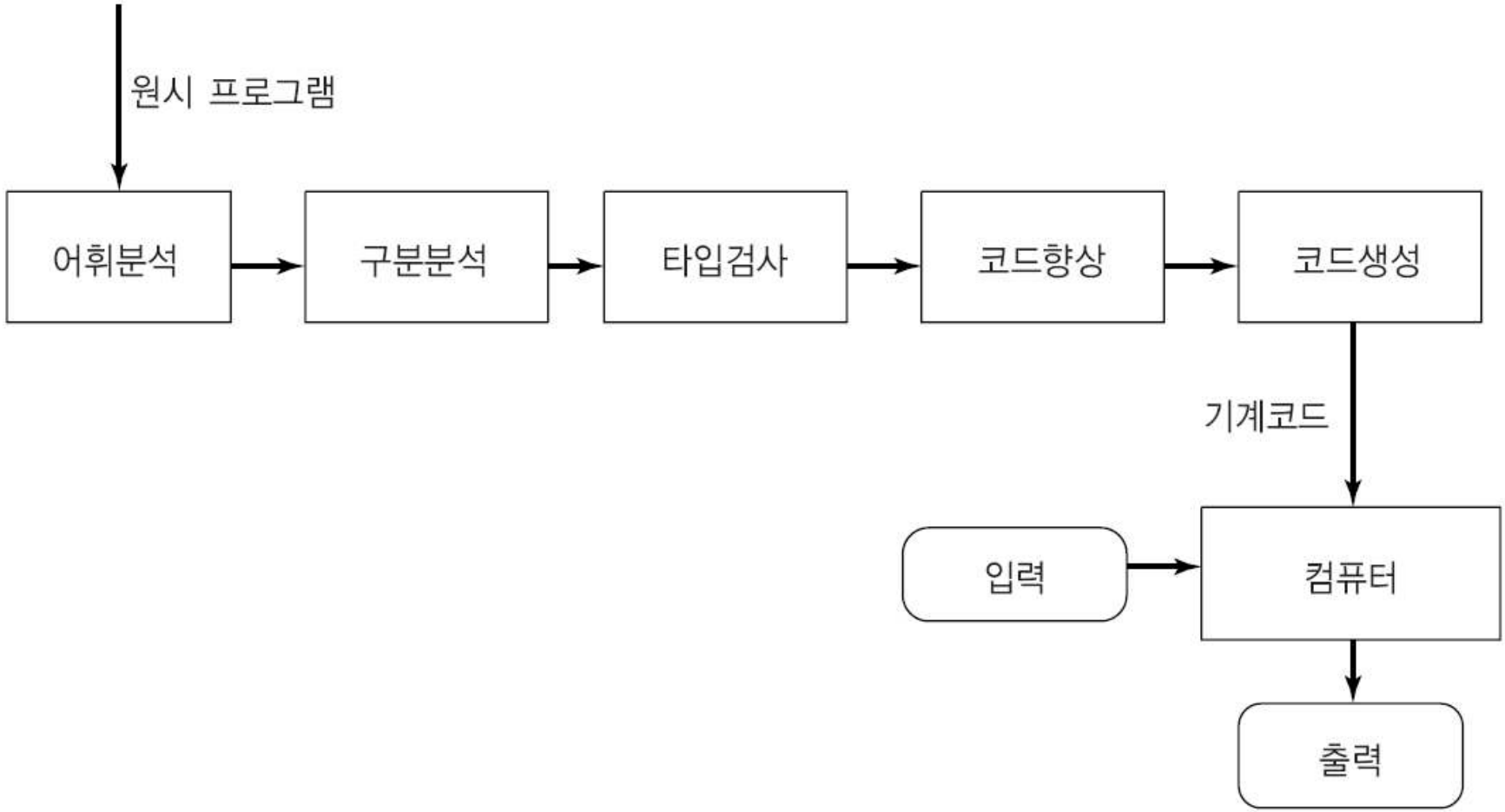

그림 1.4 컴파일-실행 과정

해석기 실행 과정

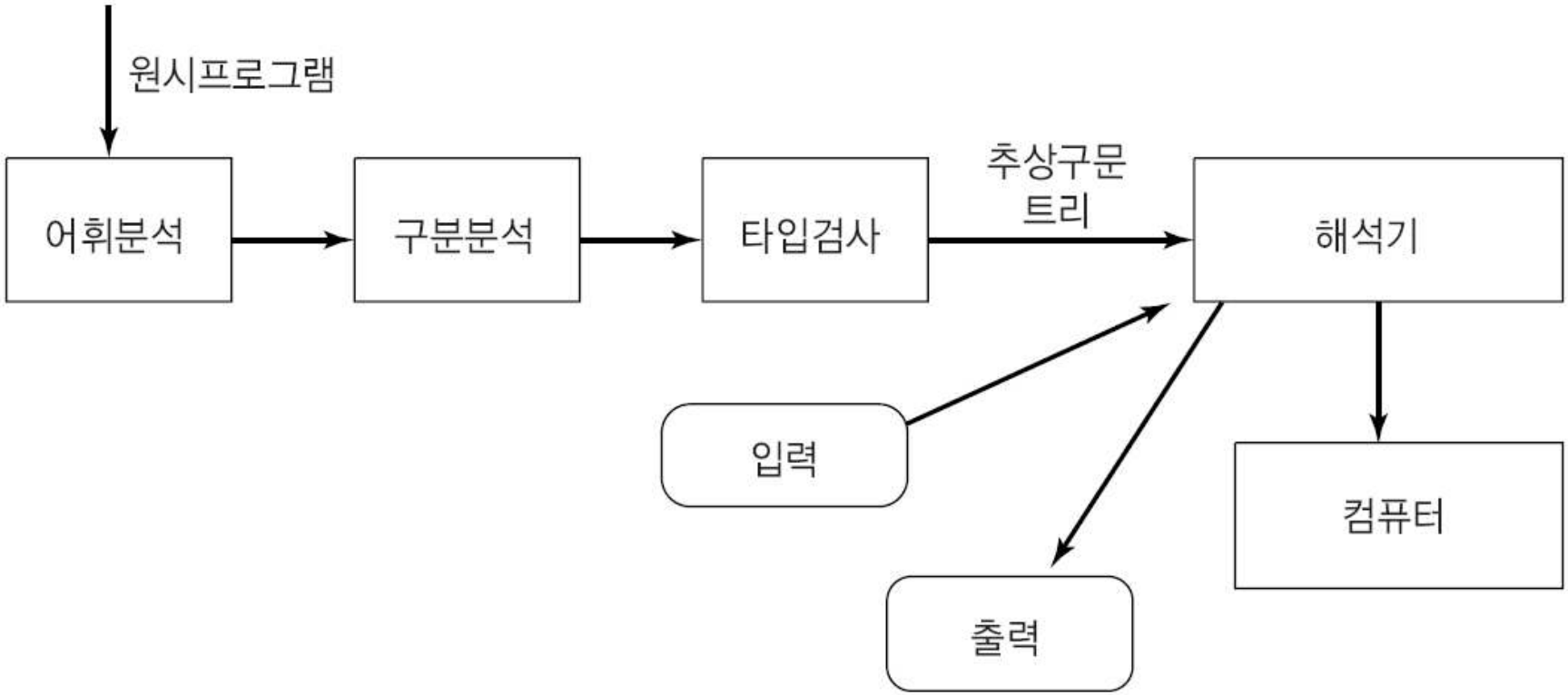

그림 1.5 가상기계와 해석기

토의 문제

1. 다익스트라(E. Dijkstra)의 다음 견해에 대한 토의

   BASIC을 먼저 습득한 학생들에게 프로그램을 잘 하도록 가르치는 것은 사실상 불가능하다. 잠재적인 프로그래머로서 재생성의 희망도 없이 정신적으로 파괴되었다.

2. 자신이 친숙한 언어에서 특별히 읽기가 힘들다고 생각되는 예제 문장을 하나 생각해 내보자. 예를 들면, C 언어에서 while (*p++ = *q++) 의 의미는 무엇일까?