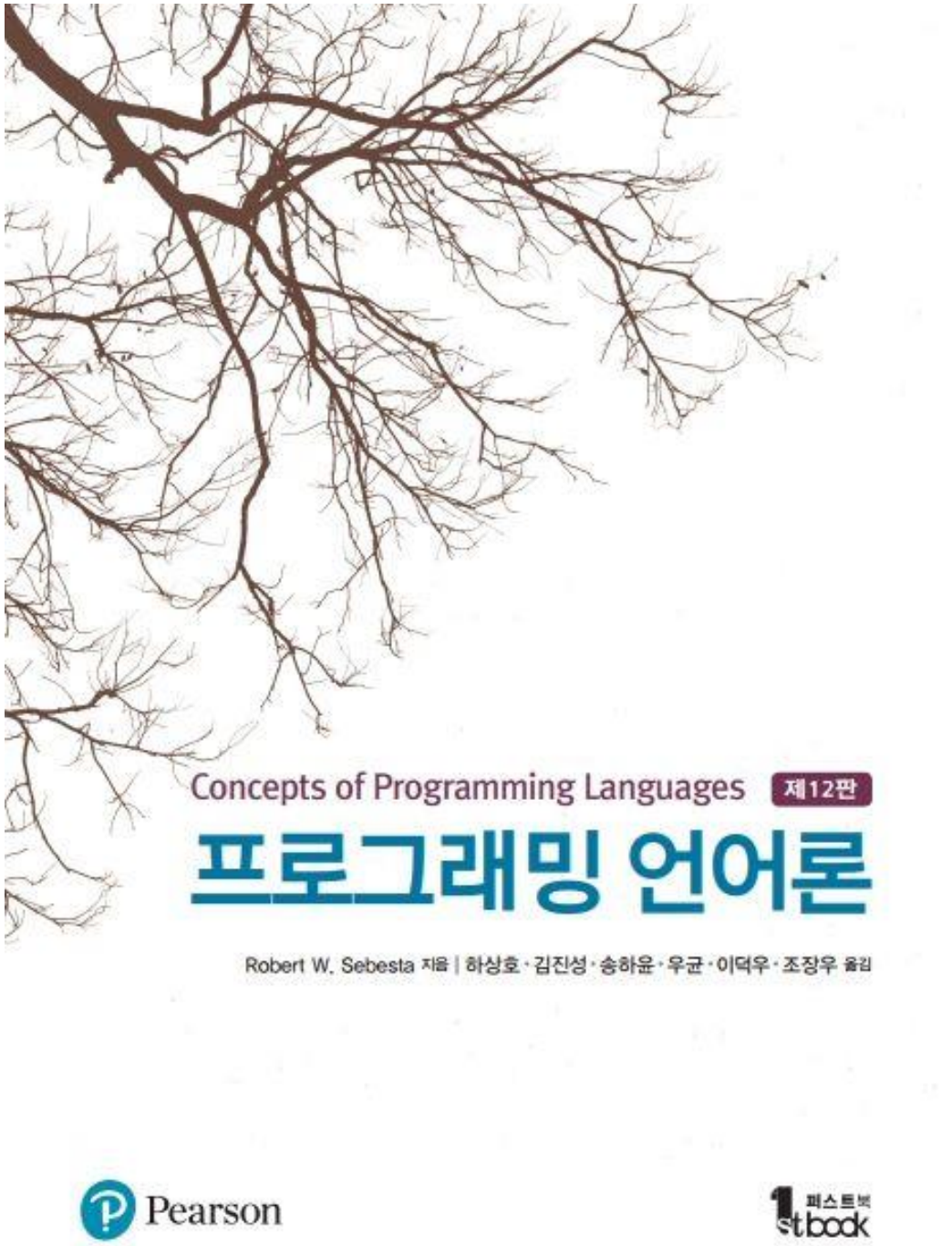
1장

기본사항


Concepts of Programming Languages
제12판
프로그래밍 언어론
Robert W. Sebesta 지음 | 하상호 · 김진성 · 송하윤 · 우균 · 이덕우 · 조장우 옮김
Pearson
1st book 퍼스트북

Chapter 1 Topics

- 프로그래밍 언어 개념을 학습하는 이유
- 프로그래밍 영역
- 프로그래밍 언어 평가 기준
- 프로그래밍 언어 설계에 영향을 미치는 영향
- 프로그래밍 언어 부류
- 프로그래밍 언어 설계의 절충(설계 상의 장단점)
- 구현 방법
- 프로그래밍 환경

프로그래밍 언어 개념을 학습하는 이유

- 생각을 표현하는 능력
- 적합한 언어를 선택하는데 필요한 배경지식
- 새로운 언어를 배울 수 있는 능력
- 구현의 중요성
- 이미 말고 있는 언어의 사용성
- 전반적으로 컴퓨팅 분야에 대한 이해

프로그래밍 영역

- 과학응용 분야
  - 단순한 자료구조, 많은 부동소수점 연산; 배열의 사용, 루프와 선택구조
  - Fortran
- 사무응용 분야
  - 상세보고서를 생성, 십진수와 문자 데이터를 표현하고 저장, 십진수의 산술연산
  - COBOL
- 인공지능(규칙기반)
  - 수치계산보다는 기호 계산, 배열보다는 연결리스트
  - LISP, Prolog
- 웹 소프트웨어
  - 프론트엔드(사용자 인터페이스): html, css, JavaScript, Dart
  - 백엔드(서버에서 데이터를 처리하고 비즈니스 로직을 수행): JavaScript, Python, Java, C#, PHP, Go, Kotlin

프로그래밍 언어 평가 기준

- **판독성**: 프로그램을 얼마나 쉽게 읽고 이해할 수 있는가
- **작성력**: 프로그램을 생성하는데 얼마나 쉽게 사용할 수 있는가
- **신뢰성**: 모든 조건하에서 주어진 명세에 맞게 수행하는가
- **비용**: 프로그래밍언어의 총비용

평가기준: 판독성

- 전반적인 단순성
  - 기본구조가 많은 것보다는 적은 것
  - 특징 다중성이 적은 것: 변수에 1 증가 시키는 4가지 방법
  - 최소한의 연산자 중복
- 직교성 (서로 독립적인 정보를 가짐)
  - 상대적으로 적은 수의 기본구조가 적은 수의 조합을 통해 제어구조와 자료구조를 생성
  - 기본구조의 가능한 조합은 적법하고 의미가 있음
  - 4개의 기본 데이터 타입(정수, 실수, 배정도 실수, 문자)와 2개의 타입구성자(배열, 포인터)
  - C 언어에서, 2개의 자료구조(배열과 구조체) 와 함수의 매개변수
- 데이터 타입
  - 적절한 데이터 타입의 제공
- 구문설계
  - 특수어(키워드 또는 예약어)
  - 복합문의 표현: 중괄호, end if와 end loop, 들여쓰기
  - 구문의 의미가 구문으로부터 직접 파악 가능: C 언어의 static

평가기준: 작성력

- 판독성에 영향을 미치는 특성은 대부분 작성력에도 영향을 줌
- 단순성과 직교성
  - 잘 모르는 구조를 사용하여 예상하지 못한 결과를 초래할 수 있음
  - 적은 수의 기본구조와 이들을 조합하는 일관된 규칙
- 추상화에 대한 지원
  - 복잡한 구조나 연산을 정의하는 능력으로, 구현의 세부사항을 모르고 사용할 수 있는 능력
- 표현력
  - 많은 양의 계산을 적은 프로그램으로 표현할 수 있는 능력
  - 강력한 연산자들과 라이브러리 함수

평가기준: 신뢰성

- 타입 검사
  - 컴파일 시 또는 실행 시에 프로그램에 포함된 타입 오류 검사, 6장
- 예외 처리
  - 실행 시간에 오류(비정상적인 조건)를 파악하여, 이를 올바르게 처리한 후, 계속 실행할 수 있는 능력, 14장
- 별칭
  - 동일한 메모리 공간에 대한 2 개 이상의 이름을 갖는 것, 포인터
  - 하나의 이름으로 값을 변경하면, 다른 이름에도 값이 변경됨
- 판독성과 작성력

평가기준: 비용

- 프로그래머를 교육시키는 비용
- 프로그램을 작성하고 유지보수하는 비용
  - 초창기 고급언어의 목표
  - 유지보수 비용은 개발 비용의 2~4배
- 컴파일과 실행 비용
  - 어느 정도의 최적화를 할 것인가?
- 이식성
  - 한 구현에서 다른 구현에서 사용될 수 있는 용이성
  - 언어의 표준화

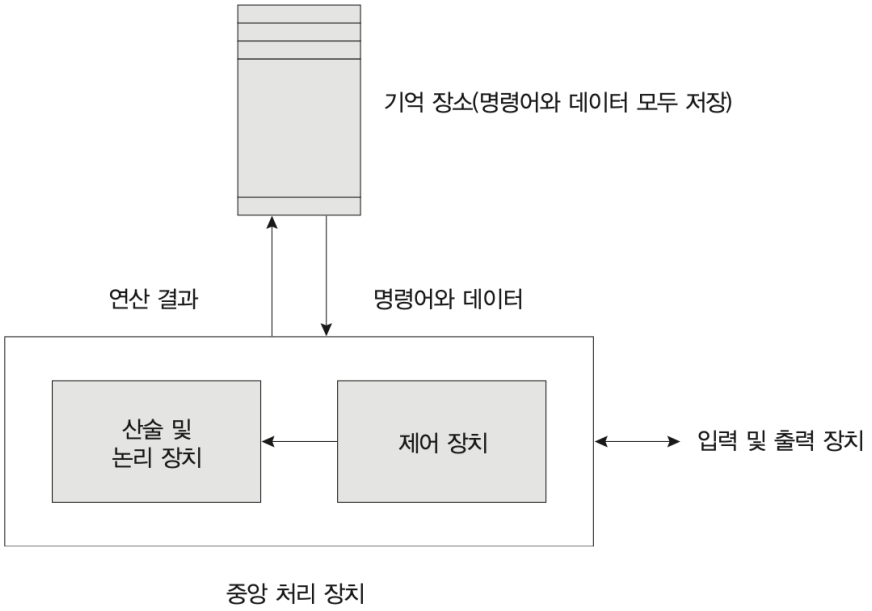
언어 설계에 미치는 다른 영향

- 컴퓨터구조
  - 폰 노이만 구조의 컴퓨터 구조에 기반하여 설계 됨
- 프로그래밍 개발 방법론
  - 새로운 개발 방법론(예를 들어, 객체지향)을 지원하는 언어가 개발됨

폰 노이만 구조

그림 1.1

폰 노이만 컴퓨터 구조



프로그램 계수기를 초기화한다

repeat 무한반복

    프로그램 계수기가 가리키는 명령어를 반입하라

    다음 명령어를 가리키도록 프로그램 계수기를 증가시켜라

    명령어를 해석하라(decode)

    명령어를 실행시켜라

end repeat

폰 노이만 구조

- 명령형 언어
  - 변수: 메모리 공간을 모델링
  - 배정문: 데이터가 메모리에서 CPU로 전달되고, CPU의 연산 결과를 다시 메모리로 전달 동작을 표현
  - 효율적인 반복문

프로그래밍 개발 방법론

- 1950년대부터 1960년대 초반: 간단한 응용, 효율적인 실행
- 1960년대 후반부터 1970년대 초반: HW비용보다는 SW 비용이 증가, 규모가 커지고 복잡해진 응용, 낮은 SW 생산성
  - 구조적 프로그래밍 (goto 사용자제)
  - 하향식 설계 및 단계적 세분화 방법론
- 1970년대 후반: 프로세스 지향 프로그래밍에서 데이터 지향 프로그래밍
  - 추상 데이터 타입
- 1980년대: 객체 지향 프로그래밍
  - 추상 데이터 타입 + 상속 + 다형성

언어의 부류

- 명령형
  - Central features are variables, assignment statements, and iteration
  - 예: C, Java, Perl, JavaScript, Visual BASIC .NET, C++
- 함수형
  - 수학의 함수에 기반, 함수와 함수를 매개변수에 적용
  - 명령형 언어의 특징을 가지지 않음
  - 예: LISP, Scheme
- 논리
  - 규칙 기반 (실행순서를 표현할 필요는 없음)
  - 예: Prolog

언어 설계 절충(trade-off)

- 신뢰성과 실행비용
  - 예: Java 배열에 대한 참조에 대한 실행시간 범위검사
  - 신뢰성 선택: 신뢰성 향상, 실행 비용 증가
- 판독성과 작성력
  - 예: APL 함축되고 간결한 많은 연산자들을 제공, 복잡한 계산을 매우 작은 프로그램으로 표현, 4줄의 프로그램을 이해하는데 4시간 걸린 예가 있음
  - 작성력 선택
- 작성력과 신뢰성
  - 예: C++ 포인터는 융통성 있는 주소 지정을 제공, 신뢰성 문제가 있음
  - 작성력 선택, 그러나 Java 에서는 지원하지 않음

구현방법–기초지식

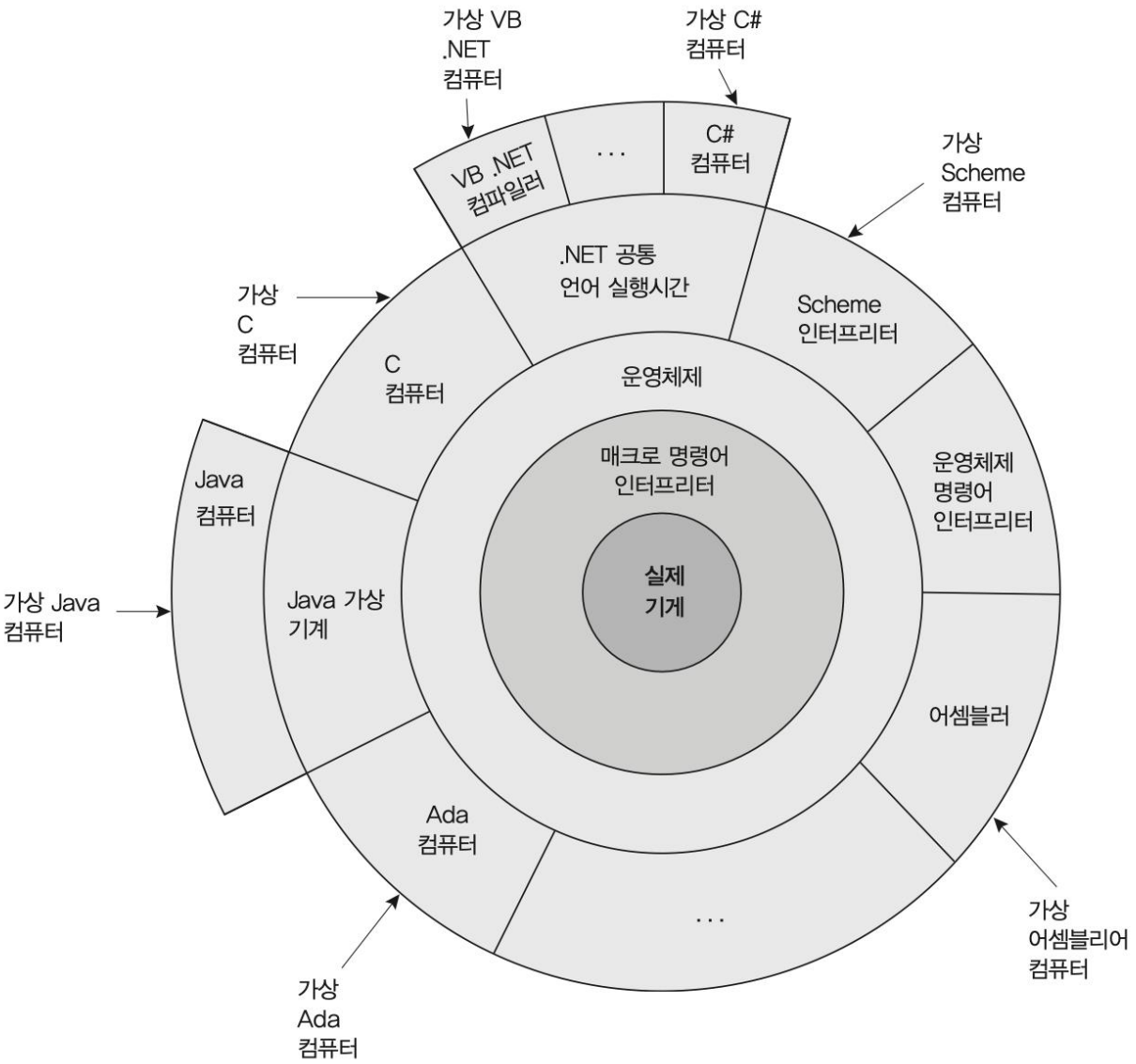
- 프로세서: 산술연산 및 논리연산과 같은 기본적인 연산을 포함한 기계 명령어들의 집합을 구현하는 회로의 모음
- 기계어: 기계 명령어들로 구성된 집합으로, 프로세서가 이해하는 유일한 언어

- 고급언어를 기계어로 하는 프로세서를 만들 수 있나?
  - 이론적으로 가능하지만, 매우 복잡하고 비쌀 것
  - 다른 고급언어를 인식하기 위해서는 ?
- 고급언어를 어떻게 실행?
  - 공통적으로 필요한 기본 연산으로 구성된 기계어를 HW로 구현하고, 고급 언어로 작성된 프로그램을 기계어로 변환해서 실행
    ➔ ?
  - 또는 시뮬레이션해서 실행 결과를 제공
    ➔ ?
  - 또는 위 두가지의 결합

컴퓨터 시스템의 계층적인 뷰

그림 1.2

전형적인 컴퓨터 시스템에서 제공되는 가상 컴퓨터의 계층적 인터페이스

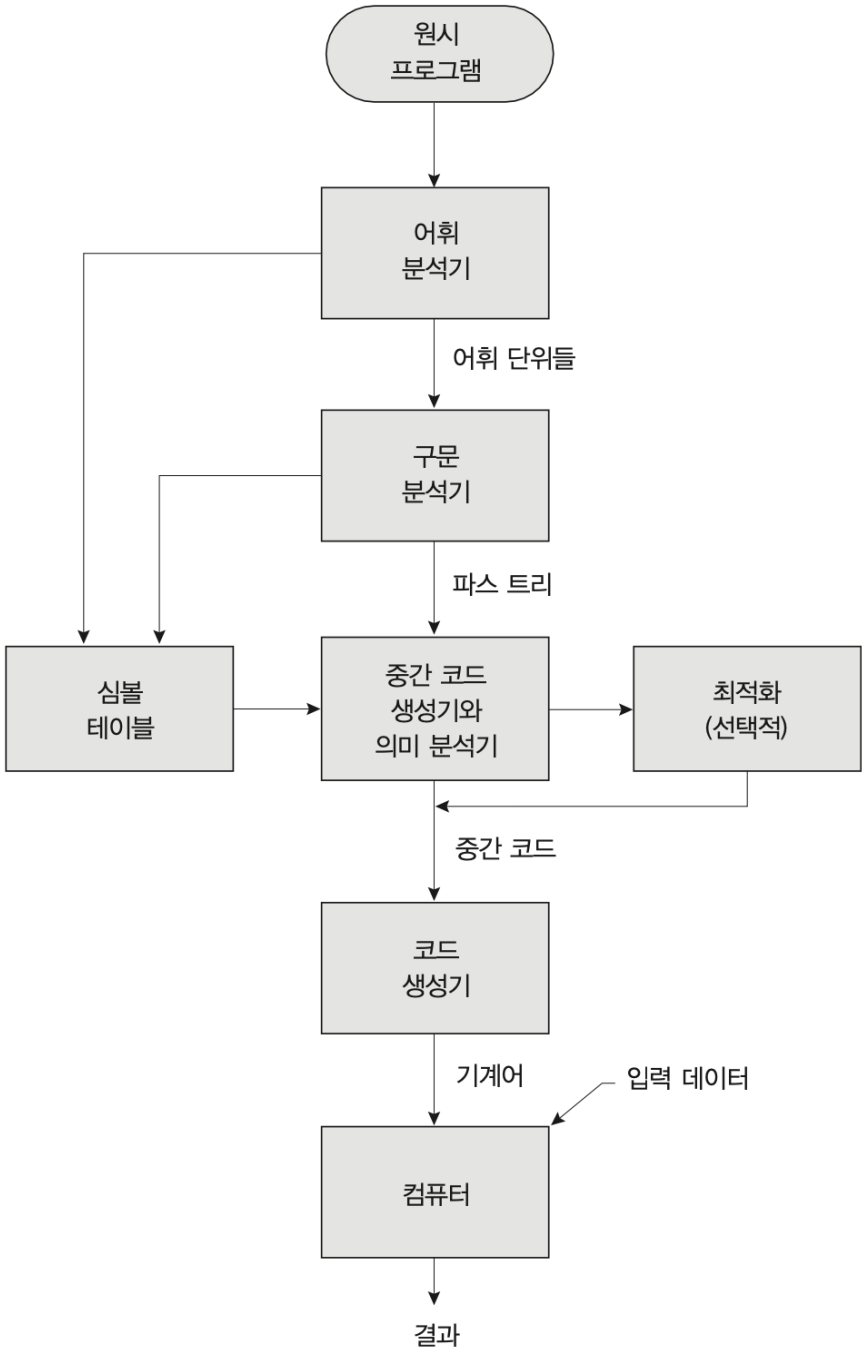


컴파일

- 고급언어 프로그램(원시 프로그램)을 기계어 코드(목적 프로그램)로 번역
- 컴파일 과정은 시간이 소요되지만, 실행 속도는 빠름
- 컴파일과 프로그램의 실행 과정은 여러 단계로 구성:
  - 어휘분석: 원시 프로그램을 어휘 단위로 나눔
  - 구문분석: 어휘 단위로 파스트리를 구성하면서, 구문 오류를 검사한다.
  - 의미분석: 의미 오류를 검사하고, 중간 코드를 생성
  - 최적화: 자원을 적게 사용하고 실행 속도가 빠른 코드로 변환
  - 코드생성: 기계어 코드 생성

그림 1.3

컴파일 과정

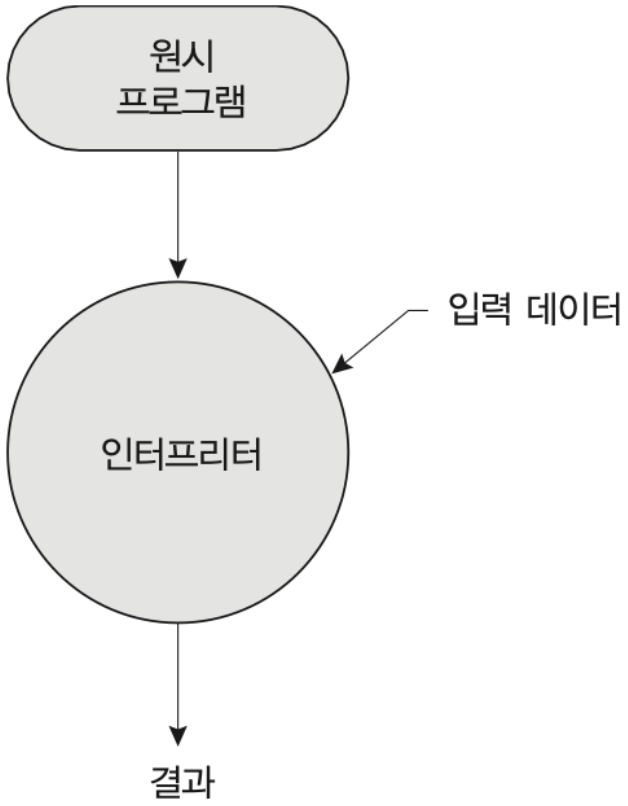

원시
프로그램
어휘
분석기
어휘 단위들
구문
분석기
파스 트리
심볼
테이블
중간 코드
생성기와
의미 분석기
최적화
(선택적)
중간 코드
코드
생성기
기계어
입력 데이터
컴퓨터
결과

컴파일 관련 용어들

- **링킹(linking):** 프로그램의 여러 부분(목적 파일, 라이브러리 등)을 하나로 묶어 실행 이미지를 만드는 과정
- **로딩(loading):** 실행 파일을 실행하기 위해서 메모리에 적재
- **적재모듈**(실행 이미지): 운영체제가 프로그램을 메모리에 로드할 수 있는 실행 가능한 코드 형태

그림 1.4

순수 해석



인터프리터(순수 해석)

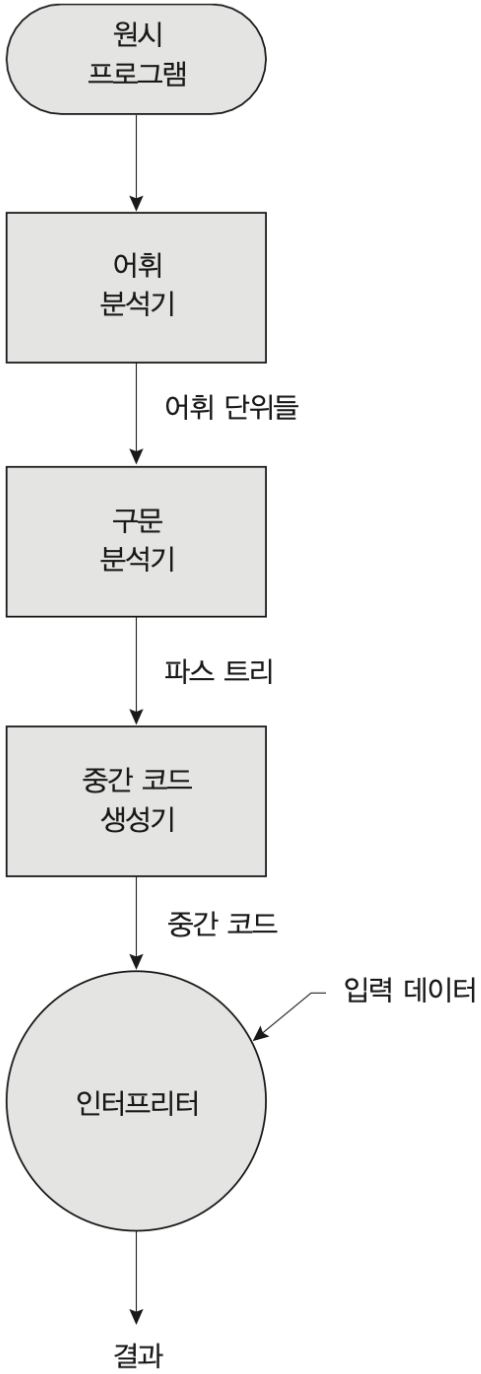
- 인터프리터(해석기)라는 SW에 의해서 해석된다.
  - 번역 과정이 없음
  - 코드를 한 줄씩 해석하고 바로 실행
  - 고급언어 프로그램의 문장을 기계어로 하는 기계에 대한 모의실험으로서 동작
- 느린 실행속도.(컴파일된 프로그램의 실행보다 10 ~ 100 배 느림)
- 더 많은 메모리를 요구
- 고급 언어에서는 드물지만, 프로토타입 구현에 사용됨(python)
- JavaScript, PHP 등에서도 사용

혼합형 구현 시스템

- 컴파일러와 인터프리터의 특징을 모두 혼합
  - 소스 프로그램을 해석하기 쉬운 중간 코드로 컴파일
  - 인터프리터가 중간 코드를 해석함
- 순수 해석보다는 실행속도가 빠름
- 예
  - **Perl**: 프로그램 해석 전에 오류를 탐지하고 인터프리터를 단순하게 하기 위해, 부분적으로 컴파일함
  - 초기의 **Java**; 바이트코드 라는 중간코드로 컴파일 후에, 자바 가상기계라고 불리는 인터프리터가 해석함.
    - efficiency?
    - portability?

그림 1.5

혼합형 구현 시스템



JIT(Just-in-Time) 구현 시스템

- 소스 프로그램을 중간코드로 번역
- 프로그램 실행에서, 중간 언어 메소드가 호출될 때, 그 메소드를 기계 코드로 컴파일한다.
- 기계코드 버전은 다음 번 호출에 대비해서 보관한다.
- 현재 Java 시스템에서 사용됨
- .NET 언어는 JIT 시스템으로 구현됨

사전처리기(preprocessor)

- 컴파일되기 전에 소스 프로그램에서 사전처리기 명령어(directives)를 처리한다
- 사전처리기 명령어(예는 C 언어)
  - 코드 재사용을 위한 헤더 파일을 포함 (#include)
  - 매크로를 정의하고 확장 (#define)
  - 조건부 컴파일 (#ifdef, #ifndef, #if, #elif, #else, #endif)

C 언어의 예

```
#include <stdio.h>

#define DEBUG  // DEBUG 매크로 정의

int main() {
    #ifdef DEBUG
        printf("디버그 모드 활성화\n");
    #endif
    printf("프로그램 실행 중...\n");
    return 0;
}
```

프로그래밍 환경

- A collection of tools used in software development
- UNIX
  - An older operating system and tool collection
  - Nowadays often used through a GUI (e.g., CDE, KDE, or GNOME) that runs on top of UNIX
- Microsoft Visual Studio.NET
  - A large, complex visual environment
- Used to build Web applications and non-Web applications in any .NET language
- NetBeans
  - Related to Visual Studio .NET, except for Web applications in Java

요약

- 프로그래밍 언어를 공부하는 이유:
  - 다양한 언어 구성 요소를 사용할 수 있는 능력을 향상
  - 프로젝트를 위한 프로그래밍 언어를 잘 선택할 수 있음
  - 새로운 프로그래밍 언어를 더 쉽게 배울 수 있음
- 프로그래밍 언어 평가의 중요한 기준:
  - 판동성, 작성력, 신뢰성, 비용
- 프로그래밍 언어 설계에 영향을 주요하게 영향을 미친 요소: 하드웨어 구조와 소프트웨어 개발 방법론
- 프로그래밍 언어를 구현하는 주요 방법에는 컴파일(Compilation), 순수 해석(Pure Interpretation), 혼합형 구현(Hybrid Implementation) 이 있다.