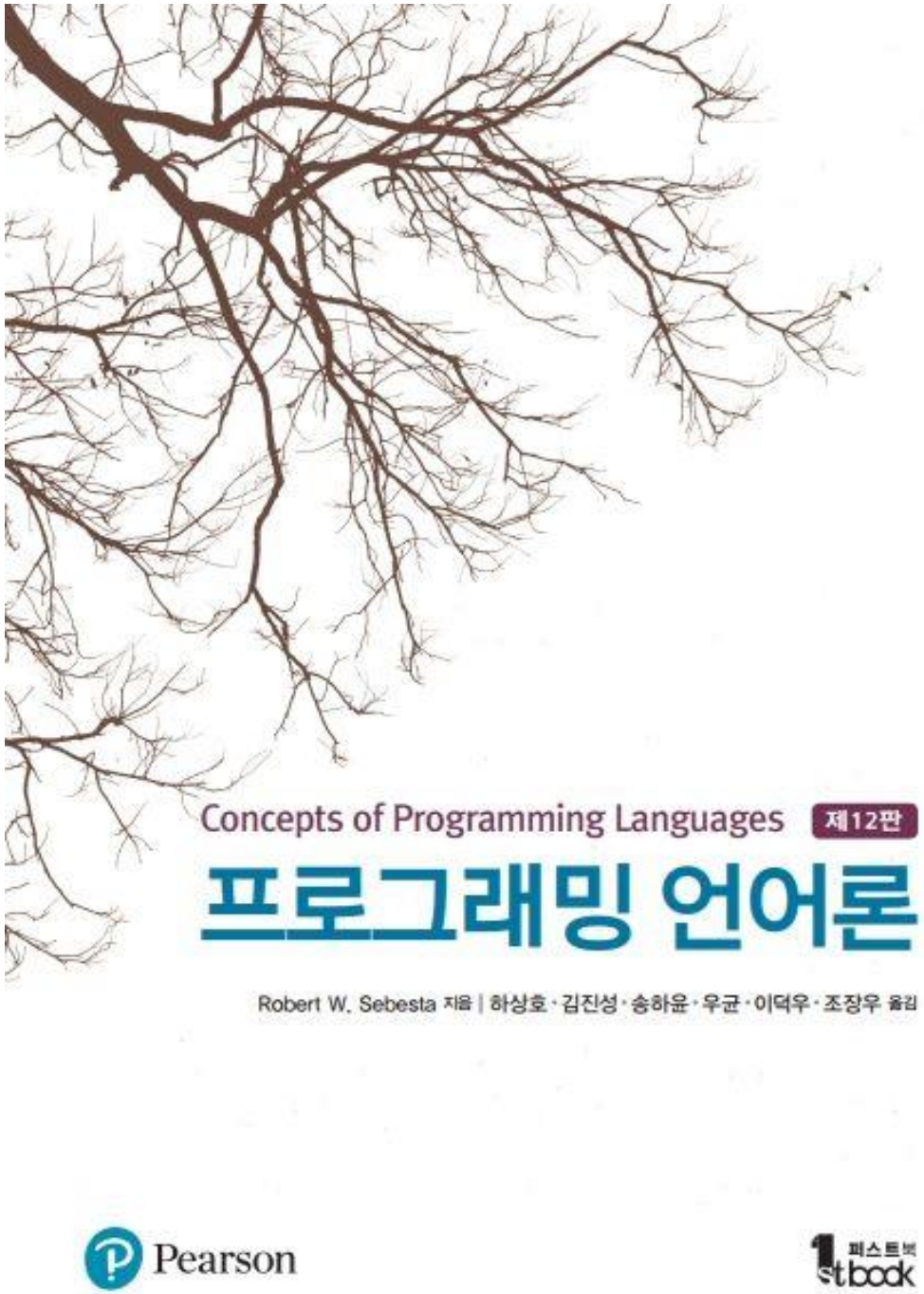
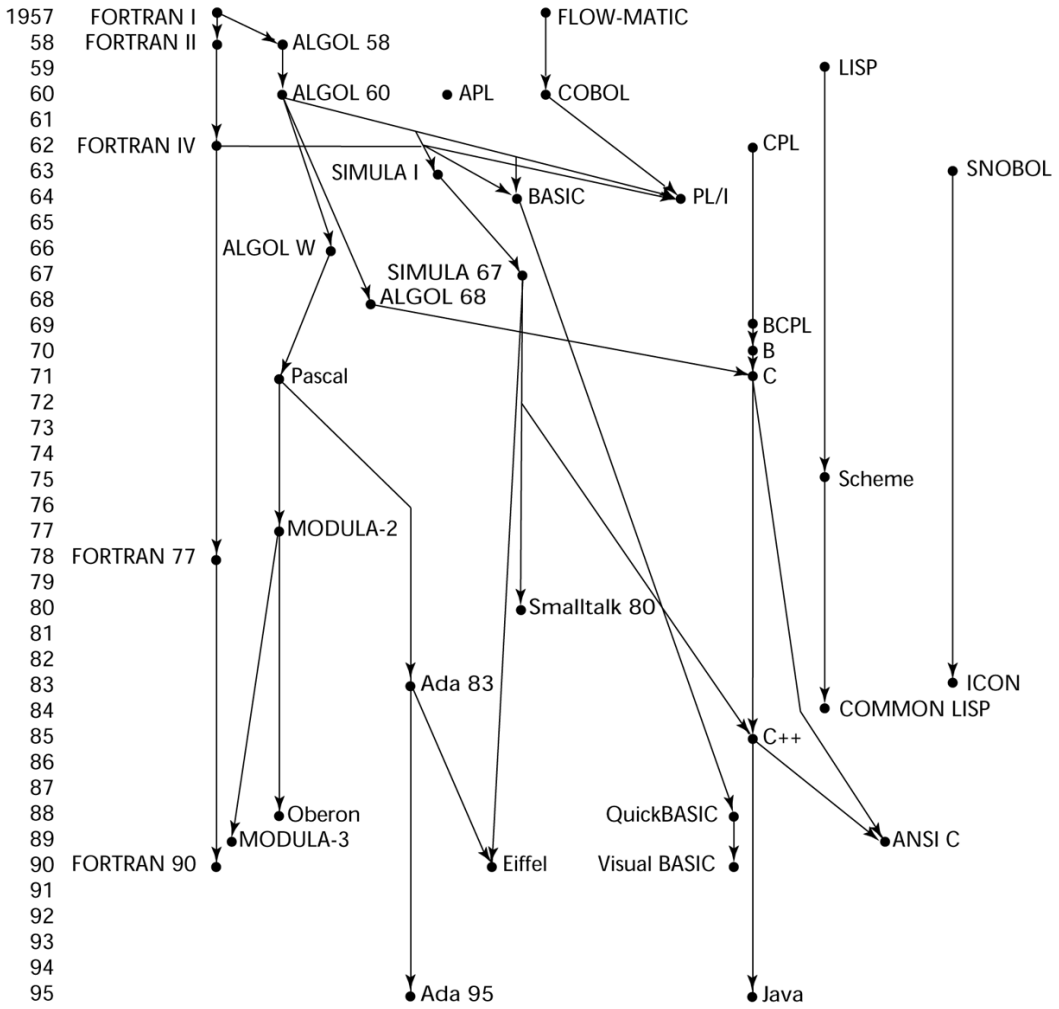
2장

프로그래밍 언어의 발전사



고급 프로그래밍언어의 계보



1950년대: 고급 프로그래밍 언어의 시작

- FORTRAN(FORmula TRANslation)
  - 과학응용 분야를 위한 효율성을 강조한 최초의 고급 언어
  - FORmula TRANslation, by John Backus at IBM
  - 간단한 자료구조를 가지나 많은 양에 대한 부동소수점 연산을 요함
  - 설계 목표 : 매우 빠르게 실행되는 코드를 생성함.
  - 주요 기능: 배열(array), FOR 반복문, 분기 IF-문 등

1950년대: 고급 프로그래밍 언어의 시작

- COBOL(COmmon Business-Oriented Language)
  - Common Business-Oriented language
  - Grace Hopper, Navy, US DoD (1959~1960)
  - 사무용으로 설계된, 영어와 비슷한 구문을 갖는 명령형 언어
  - 설계 목표 : 판독성
  - 주요 기능: 레코드 구조, 프로그램의 실행부와 분리된 자료구조
  - 다양한 출력 기능 등

- LISP(List Processor)
  - McCarthy by MIT (다트머스 워크샵, 존 매커시, 마빈 민스키, 등, AI 용어)
  - 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.
  - 새로운 기능.
    - 예외처리, concurrency, 포인터 타입, switch 문장

1960년대: 프로그래밍 언어의 다양성

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

- BASIC
  - Beginner's All-purpose Symbolic Instruction Code
  - 대화형(Interactive) 프로그래밍을 지원
  - 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 만듦.
  - 대표적인 블록 구조 언어

- C 언어
  - 유닉스 운영체제 개발을 위해 개발된 시스템 프로그래밍 언어
  - 중급 언어(middle-level)
    - 저급언어처럼 메모리를 직접 다룰 수 있고, 효율적인 실행
    - 고급언어처럼 가독성이 좋고, 모듈화 및 함수기반 프로그래밍
  - 모든 컴퓨터 시스템에서 사용할 수 있도록 설계된 언어

1970 년대 : 단순성 및 새로운 언어의 추구

- Prolog
  - 술어 논리를 사용하는 대표적인 논리 프로그래밍 언어
  - 사실(Fact), 규칙(Rule), 질의(Query) 로 구성됨
  - 명령형 언어처럼 절차를 지정하는 것이 아니라, 무엇을 할 것인가를 기술.
  - 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
  - Pascal의 한계를 보완하여 모듈개념을 추가
  - 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
    - 하위 호환성(backward compatibility)
  - 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 사용 가능
  - Objective-C처럼 LLVM으로 빌드되고 같은 런타임 시스템을 공유
  - 특징: 클로저, 다중 리턴 타입, 네임스페이스, 제네릭, 타입 유추

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)

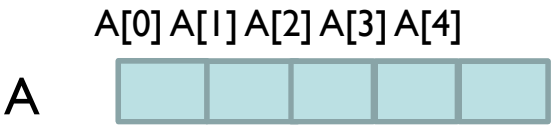
- 기본 추상화
  - 기본 데이터 관련한 요약
  - 예: 변수, 자료형

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

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

데이터 추상화

- 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)

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

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

제어 추상화(Control Abstraction)

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

X = X+3

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

  - goto 문
    - jump 명령어의 요약

제어 추상화

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

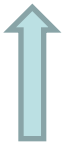
- 예
  - if-문, switch-문
  - for-문, while-문 등

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

제어 추상화

- 예

```
while (condition) {
      statements
}
```



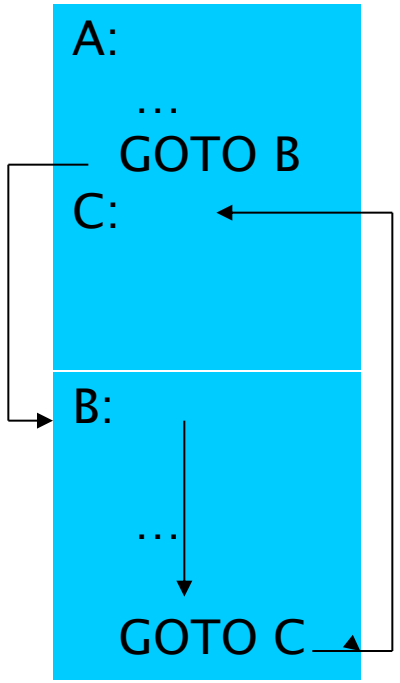
추상화

```
L1: if  (~condition) GOTO L2

      code for statements

      GOTO L1
L2 : ...
```

- 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




```
A 부분

CALL B( )

C 부분


B( ) {

...

RETURN
```

추상 자료형(Abstract Data Type)

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

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