




2/3장 역사 및 언어 설계 원리



차례

- 2.1 초기 역사 : 최초의 프로그래머
- 2.2 1950년대: 최초의 프로그래밍 언어
- 2.3 1960년대 : 프로그래밍 언어의 폭발적 증가
- 2.4 1970년대 : 단순성, 추상화 및 연구
- 2.5 1980년대 : 새로운 방향의 모색
- 2.6 1990년대 : 통합, 인터넷, 라이브러리
- 미래



2.1 초기 역사 : 최초의 프로그래머



초기의 프로그래밍

- Early electronic computer
- Machine language programming
- Assembly language programming



2.2 1950년대: 최초의 프로그래밍 언어



FORTRAN 개요

- 개요(Overview)
 - John Backus at IBM
 - 과학계산용 언어
 - 새로운 기능
 - array, loop, if,
 - 새로운 버전들
 - FORTRAN66, FORTRAN77, FORTRAN90, HPF
- 그 당시 컴퓨팅에서 가장 중요한 것은 ?
- FORTRAN의 주요 설계 목표는 무엇인가?



FORTRAN 설계 목표

- 실행 효율(Efficiency of execution)
 - 효율적인 실행 코드 생성
- 왜 실행 효율인가?
- 실행 효율과 관련된 FORTRAN의 기능은 무엇인가?



Algol60 (ALGOarithmic Language)

- Algol 위원회에서 개발됨 (1958–1960)
 - 알고리즘 기술을 위한 일반적(general)이고 표현력 좋은(expressive) 언어
- 현재 명령형 언어의 기초
 - Pascal, C, Modula-2, and Ada
- 새로운 개념
 - free-format, structured statements, begin-end blocks,
 - type declarations for variables
 - Recursion
 - call-by-value parameters



ALGOL의 주요 설계 원리

- **작성용이성(Writability)**
 - 계산과정을 명확하고, 올바르고, 간결하고 빨리 기술할 수 있는 능력
- 예
 - block structure, and recursion



COBOL: 개요

- **COBOL(Common Business-Oriented Language)**
 - 미국 국방성(U.S. DoD)
 - Grace Hopper 팀에 의해서 개발됨
- **사무용 언어(business-oriented language)**
- 새로운 기능
 - 구조체(record structure)
 - 자유스런 출력 포맷(versatile output formatting)
 - separation of data structures form the execution section



COBOL: 주요 설계원리

- **판독성(Readability)**
 - 계산과정을 쉽고 정확하게 이해할 수 있는 능력
- 예
 - 영어 문장 스타일의 구문 구조



LISP: 개요

- **LISt Processor**
 - MIT의 John McCarthy에 의해서 설계
- AI 응용을 위한 함수형 언어
 - 일반적인 **리스트 구조**
 - 함수



LISP: 주요 설계 원리

- 표현력(Expressiveness)
 - 복잡한 계산과정 및 구조를 쉽게 표현할 수 있는 능력
- 예
 - 자기호출(recursion)
- 단점
 - 표현력은 단순성과 상충될 수 있다.




2.3 1960년대 : 프로그래밍 언어의 폭발적 증가

*Dream of more general and
universal languages*




왜 이런 유행이 있었을까?



PL/I : 개요

- IBM PL/I 프로젝트
 - FORTRAN, COBOL, Algol60의 모든 좋은 기능을 통합하자 !
 - 병행성(concurrency) 및 예외처리(exception handling)
- 컴파일러
 - 작성이 어려웠다
 - 느리고, 크고, 신뢰성이 떨어졌다.
- 언어
 - 배우기 어렵다
 - 사용하는데 오류를 범하기 쉽다.
 - 언어 기능들 사이에 예상치 못한 상호작용으로 인하여



Algol-68 : 개요

- Algol60 개선
 - 보다 표현력있고 일반적인 언어
 - 완전히 직교적인(orthogonal) 언어



직교성 원리(Orthogonality Principle)

- 구문구조가 상황에 따라 의미가 달라지면 안 된다.
 - 문맥 의존적인 제약은 직교성을 위반한 것이다.



직교성 원리

- 동등비교 연산자의 일반성 부족
 - 직교성 부족으로 이해할 수 있다.
- C의 매개변수 전달
 - 배열을 제외한 모든 타입은 값 전달(pass by value)
 - 배열은 참조 전달(pass by reference)
- C/C++에서 반환
 - 배열을 제외한 모든 데이터 타입의 값은 함수로부터 반환될 수 있다.
 - Ada와 대부분의 함수형 언어에서 이러한 제약은 없어졌다.



Simula-67: 개요

- 시뮬레이션을 위해 설계됨
- 객체 지향 언어
 - 객체
 - 클래스
 - 상속
- 복잡도 제어(Complexity control)
 - 프로그래밍과 언어에 복잡도 증가
 - 추상화 메커니즘을 이용하여 프로그래밍의 복잡도를 제어



2.4 설계 원리: 규칙성



규칙성(Regularity)

- 언어가 제공하는 기능들을 어떻게 통합할 것인가?
 - 통합에 있어서 보다 규칙성이 있다면 좋을 것이다.
- 왜
 - 사용에 있어서 별난 제약이 적고
 - 구문구조 사이의 이상한 상호작용이 적고
 - 언어의 기능성에 있어서 놀라운 경우가 적을 것이다.
- Regular rules, without exceptions, are easier to learn, use, describe and implement



규칙성의 종류

- 일반성(Generality)
- 직교성(Orthogonality)
- 일관성(Uniformity)




일반성 원리(Generality Principle)

- 구문구조 사용에 있어서 특별한 경우를 피한다.
- 밀접하게 연관된 구문구조를 하나의 보다 일반적인 것으로 통합한다.



일반성 원리 예1

- 동등비교 연산자 “==”
 - C 언어에서 스칼라 타입 혹은 포인터 타입에 대해서 적용 가능하다.
 - 배열 및 레코드에 적용 불가
 - 이 제약은 Ada와 C++에서 부분적으로 해결됨



일반성 원리 예: 프로시저의 일반성

- Pascal
 - 프로시저 혹은 함수는 중첩가능
 - 매개변수로 전달 가능
 - 프로시저 변수는 없다
- C 언어
 - 함수를 중첩할 수 없다
 - 함수를 변수에 저장할 수 있고
 - 매개변수로 전달할 수 있고 반환할 수 있다.
- 함수형 언어
 - 함수가 완전히 일반적으로 사용될 수 있다.
 - Function is a first-class value in functional language



일반적일수록 좋을까?



직교성 원리 (Orthogonality Principle)

- 구문구조가 상황에 따라 의미가 달라지면 안 된다.
 - 문맥 의존적인 제약은 직교성을 위반한 것이다.



직교성 원리

- 동등비교 연산자의 일반성 부족
 - 직교성 부족으로 이해할 수 있다.
- C의 매개변수 전달
 - 배열을 제외한 모든 타입은 값 전달(pass by value)
 - 배열은 참조 전달(pass by reference)
- C/C++에서 반환
 - 배열을 제외한 모든 데이터 타입의 값은 함수로부터 반환될 수 있다.
 - Ada와 대부분의 함수형 언어에서 이러한 제약은 없어졌다.



일관성(Uniformity)

- *Things which look similar should be similar and things which look different should be different*
- C++ 세미콜론
 - `Class A { ... } ;` // 세미콜론 필요함
 - `Int f() { ... }` // 세미콜론 불필요
- Pascal 반환문과 배정문

```
function f: boolean;
begin
  ...
  f := true;
end;
```