Chapter 3
언어 설계의 원칙들

역사와 설계의 지표들
효율성과 규칙성
다른 언어 설계의 원칙들
케이스 스터디: C++

무엇이 좋은 언어인가?

- 언어의 설계
  - 좋은 언어가 가지는 바람직한 특성은 무엇인가?
- 언어가 성공할지는 예측하기 어렵다
  - Pascal은 성공했으나, Modula-2는 실패
  - Algol60 은 성공했으나, Algol68 는 실패
  - FORTRAN 은 성공했으나, PL/I 는 실패
- 언어가 성공하는 이유는?
  - 실용성: 사용의 용이함(availability), 가격, 번역기의 질
  - 킬러 응용: C는 유닉스, 자바는 인터넷, ...
  - 설계 개념의 일정함 (필요 조건)

언어의 중요한 특성들

- 설계의 목표를 명심해야 함
- 좋은 프로그래밍 언어가 가지는 중요한 특성은
  - 단순성 (Niklaus Wirth)
  - 언어 구성 요소(language constructs) (C. A. R. Hoare)
  - 멋진 기능들의 모음만은 아님 (Bjarne Stroustrup)

  → 서로 상충되는 대답… 즉, 애매한 문제
- 따라서 이 장에서는

  설계 원리를 조망해 보고

  프로그래밍 언어의 좋은 예와 나쁜 예를 리뷰함

Java는 언제나 C보다 쉬운가?

- http://www.borlandforum.com/impboard/impboard.dll?action=read&db=bcb_qna&no=22383

역사와 설계 지표

- 1950 년대
  - 효율성이 제일 중요함
  - 쓰기 쉬운 점과 읽기 쉬운 점은 부차적인 지표
- 1960 년대
  - 추상화 메커니즘이 중요해짐
- 1960 년대 ~ 1980 년대 초기
  - 단순성과 신뢰성이 강조됨
- 1980 년대 ~ 1990 년대
  - 수학적인 정밀도가 높아짐

효율성

- FORTRAN의 주된 목표
- C/C++같은 언어들에게는 아직도 중요한 목표
- 효율성의 다양한 측면
  - 실행 코드의 효율성: 컴파일 최적화와 연관됨
  - 번역기의 효율성: 원-패스 컴파일러?
  - 구현의 어려운 정도(구현용이성): 번역기를 만드는 데의 효율성
  - 프로그래밍 효율성: 표현의 풍부함(표현력)
  - 신뢰성(reliability)와 유지보수 용이성(maintainability): 판독성(readability)과 연관됨

C 코딩에서의 최적화

- Writing Efficient C and C Code Optimization
- http://www.joinc.co.kr/modules/moniwiki/wiki.php/Site/C/Documents/COptimization

규칙성(Regularity)

- 언어의 특성들이 어떻게 통합되어 있는가?
- 규칙성은 다음과 연관되어 있음 (중요!)
  - 일반성(Generality): 특별한 경우를 피함(75쪽)
  - 직교성(Orthogonality): 언어 기능을 사용함에 있어 문맥에 독립적으로 함
  - 일률성(Uniformity): 보이는 바와 행동하는 바가 일관성 있음 (외모와 행동이 일치~)

일반성(Generality)의 예(76쪽)

- 프로시저 변수
  - 파스칼은 프로시저 매개 변수는 있으나, 프로시저 변수는 지원하지 않음
  - C 언어는 함수 포인터로 프로시저 변수 지원
- 가변 길이 배열의 예
  - 파스칼은 지원하지 않으나, C, Ada, 그리고 FORTRAN 은 지원함
- 등가 연산자의 응용
  - C 언어는 배열의 비교를 지원하지 않으나, Ada는 지원
- 상수(Constant)의 선언
  - 대부분의 언어들은 상수의 초기값에 대한 제한이 있음

C의 함수 포인터

```
int func()
{
  return 0;
}
int main()
{
  int (*fp1)() = &func;
  (*fp1)(); // 함수 포인터로 함수 호출
}
```

직교성(Orthogonality)의 예(76쪽)

- 반환 형(Return Type)
  - 파스칼: 스칼라(scalar) 또는 포인터(pointer) 형
  - C, C++: 배열 형은 제외
  - Ada와 대부분의 함수 언어들: 모든 형들
- 선언(Declaration) ← 실습해 볼 것 (C# 포함)
  - C: 블록의 처음
  - C++: 블록의 어디에서도 선언 가능
- 매개 변수 전달(Parameter Passing)
  - C에서의 매개 변수 전달 방법은 배열을 제외하고는 값 전달(pass-by-value) → 직교적이지 않음

C의 선언

```
#include <stdio.h>
int main()
{
   int i = 10;
   printf("i = %d\n",i);
   int k = 30; // 에러 (C에서 선언은 블록의 처음에만 가능)
   // C++에서는 에러 아님
   {
        int j = 20;
        printf("j = %d\n",j);
   }
   printf("j = %d\n",j); // 에러 (범위 밖)
}
```

C에서 다차원 배열의 매개 변수 전달

```
void func1(int *ar1, int len) {            }
void func2(int (*ar2)[3], int len)  {         }
void func3(int (*ar3)[5][6], int len)  {        }

int main()
{
   int ar1[5];
   int ar2[2][3];
   int ar3[4][5][6];
   func1(ar1,5);
   func2(ar2,2);
   func3(ar3,4);
}
```

일률성(Uniformity)의 예

- 딜리미터(Delimiters) ← 실습 (C# 및 자바 포함)

  C++에서

  – 클래스 정의의 경우 ; 으로 끝나야 한다

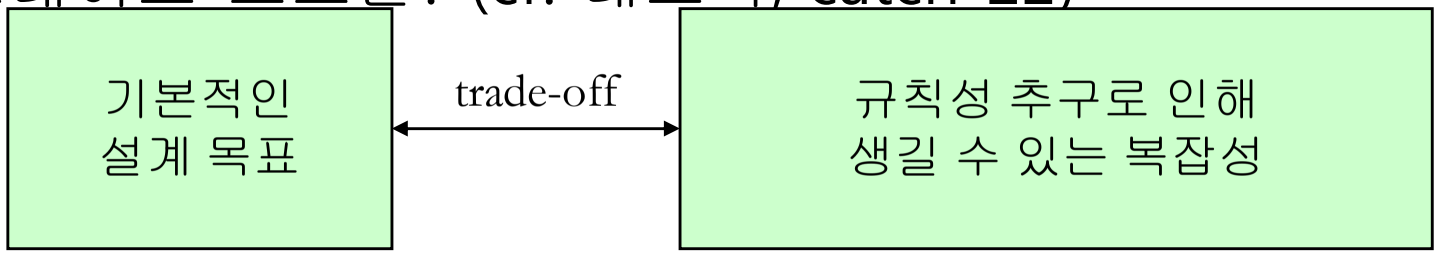
  – 함수의 정의는 ; 으로 끝나면 안된다

- 값의 반환

  – 파스칼에서의 값 반환은 배정문assignment)처럼 보임 → 일률적이지 않음 (77쪽)

규칙성의 몇 가지 더 할 얘기들

- 왜 불규칙적인 것들이 생기는가?
  - 역사적인 이유: 과거의 기능과의 호환성
  - 실제적인 이유: 스택 기반 실행 환경의 한계 (8장)
- 언어 설계에 있어 지나치게 일반성 또는 직교성을 기대하는 것은 위험할 수 있음 (78쪽)
  - 어느 정도의 트레이드-오프 (trade-offs) 가 필요함
  - 트레이드-오프란? (cf: 데드락, catch-22)



단순성(Simplicity) (79쪽)

- C 언어의 특징 중 하나임
- 파스칼이 주된 성공한 이유
- 단순성은 규칙성이 아님 – Algol68은 규칙적이지만 단순하지 않음
- 단순성은 언어 구성의 개수가 적다고 달성되는 것이 아님: LISP은 언어 구성 요소의 개수가 적으나, 매우 복잡한 실행 환경을 가짐
- 즉, 지나치게 단순화된 언어는 사용하기 어려움

"Everything should be made as simple as possible, but not simpler" — Albert Einstein

모든 것은 가능한한 단순하게 만들어져야 하지만, 너무 단순해서는 안된다.

- 오캄의 면도날, 또는 검약의 원리 (principle of parsimony)
- 출처: Model Selection and Multimodel Inference by Burnham and Anderson

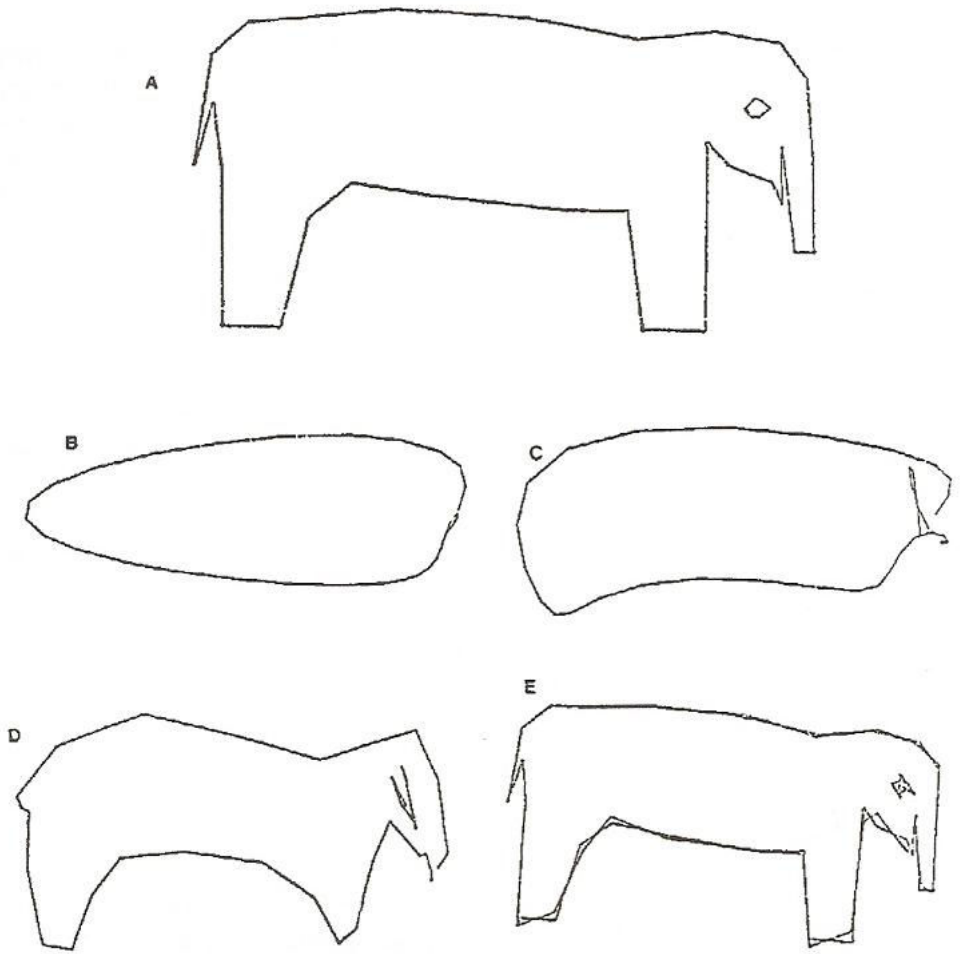


FIGURE 1.2. "How many parameters does does it take to fit an elephant?" was answered by Wel (1975). He started with an idealized drawing (A) defined by 36 points and used least squares Fourier sine series fits of the form $x(t) = \alpha_0 + \sum \alpha_i \sin(it\pi/36)$ and $y(t) = \beta_0 + \sum \beta_i \sin(it\pi/36)$ for $i = 1, \ldots, N$. He examined fits for $K = 5, 10, 20$, and 30 (shown in B–E) and stopped with the fit of a 30 term model. He concluded that the 30-term model "may not satisfy the third-grade art teacher, but would carry most chemical engineers into preliminary design."

표현력(Expressiveness)

- 복잡한 과정이나 구조를 언어가 얼마나 쉽게 표현할 수 있는가를 나타내는 것
- 단순성과 상충될 수 있음: Algol68은 표현력이 좋지만 복잡한 언어
- 객체지향 언어가 인기 있는 이유
- 표현력은 간결함이라고도 볼 수 있는 데, 간결함이 지나치면 가독성(readability)이 떨어짐

  C 코드 예)

```
while (*s++ = *t++); // 유명한 예제이나 피해야 함
(cf: C 언어의 시퀀스 포인트)
```

확장성(Extensibility)

- 사용자가 새로운 기능을 언어에 추가할 수 있게 하는 능력
- 새로운 데이터 형이나 함수 정의
- 키워드나 언어 구성 요소를 번역기가 처리할 수 있도록 추가: LISP 에서는 이러한 환경의 변경이 가능함
- 연산자 오버로딩 (Ada, C++, C#) ← 실습 C++, C#
- 새로운 연산자 정의(operator) (Haskell은 가능)
- 확장성이 없는 단순함을 가진 언어는 무용(無用)함
- 언어의 설계자는, 언어를 처음 설계할 때, 어떤 특성이 언어의 핵심으로 포함되어야 하는지를 결정해야 함

제약가능성(Restrictability)

- (가능한한 간편하게) 언어 주제들을 정의하는 능력 ← 뭔 소류?
  - 프로그래머가 그 언어에 대한 최소한의 지식과 최소의 구문을 사용하여 프로그램을 작성할 수 있도록 함
- 어떤 특성이 언어의 기본 기능으로 들어가야 하는가? 아니면 라이브러리로 지원되야 하는가? (자바 Thread vs. C PThreads)
- 구문적 당의(構文的 糖衣; syntactic sugar)로 프로그램의 이해도를 높일 수 있음 (C에서 for 루프는 while 루프의 syntactic sugar)
- 사용하지 않는 기능으로 인해 속도나 시간같은 성능의 저하가 있으면 안된다

이미 채택된 표기법과 관례와의 일관성

- 언어는 경험 있는 프로그래머들이 배우기 쉬워야 한다

  반례들)

  – Algol68 에서는 "type" 대신 "mode" 키워드 사용
  – FORTRAN 에서는 공백 무시
    - DO 99 I = 1.10 → DO99I = 1.10
  – FORTRAN의 키워드들은 예약되어 있지 않다
- 최소 경악의 법칙 (The law of least astonishment)

  "Things should not act or appear in a completely unexpected way"

  모든 것들은 완전히 예상 밖으로 행동하거나 나타나서는 안됨

다른 설계 원칙들

- 엄밀성 (Preciseness (또는 명확성 Definiteness))
  - 정확한 언어의 정의가 있어야 한다
  - Backus Naur Form 또는 Extended Backus Naur Form (중요! 4장에서 다룸)
- 기계 독립성
  - 기계 종속적인 부분을 명확히 찾아내서 고립시킴
  - 예: C 언어에서 limits.h 와 float.h
- 보안
  - 프로그래밍 오류를 줄이고, 오류가 발견되면 보고하기 쉽도록 함

"Maximize the number of errors that could not be made" — C. A. R. Hoare, 1981

범할 수 없는 오류의 수를 극대화한다

케이스 스터디: C++ (1/3)

- 배경
  - Simular67이 너무 비효율적이다
  - C 기반의 시뮬레이션에 적합한 새로운 언어 필요
- 초기의 설계 목표
  - 프로그램 작성이 쉬워야 한다 – 클래스와 엄밀한 형 체크
  - 효율성: C와 호환가능해야 한다
  - 실제적인 문제들: 이식성, 구현가능성, 상호작용성

케이스 스터디: C++ (2/3)

- C with Classes (1079~80)
  - Cpre: C++ 코드를 C 코드로 번역함
  - 중요한 특징들이 빠져있었음
    - 가상 함수 (virtual functions)
    - 템플릿 (template)
    - 일반적인 오버로딩
- C++ (1985): 확장된 설계 목표들
  - C와의 호환성
  - 단순히 가능하니까 멋진 기능을 추가하는 featurism 배제
  - 오버헤드를 최대한 줄인다 – "zero-overhead" 규칙
  - 다중 패러다임 언어 ("multiparadigm")
  - 엄밀한 형 체킹
  - 단계적으로 배울 수 있게 함
  - 다른 언어들과의 호환성

케이스 스터디: C++ (3/3)

- 성장과 표준화
  1985. 저자의 첫번째 C++ 책이 나옴
  1986. 첫번째 상업적인 컴파일러 등장
  1987. USENIX에서 C++에 대한 첫번째 학술대회
  1989. ARM (Annotated Reference Manual)이 나옴
  1994. STL(Standard Template Library)이 추가됨
  1998. 미국 표준 (ANSI/ISO) C++
- 실수들(mistakes)
  - 비슷한 일들을 하는 데, 너무 많은 방법이 있음
  - 표준적인 라이브러리가 처음부터 나오지는 않음 (cf. 자바)