
프로그래밍 언어 기본적인 사항

Concepts of Programming Languages

2024년 1학기

한양대학교 인공지능대학원
임덕선

목차

- 프로그래밍 언어(론) 학습 이유
- 프로그래밍 영역(Domain)
- 프로그래밍 언어 분류
- 프로그래밍 언어 평가 기준
- 구현 방법

프로그래밍 언어 학습 이유

□ 생각(idea)을 표현할 수 있는 능력이 향상

- 프로그래머의 프로그래밍 능력은 언어에 대한 이해 정도에 따라 좌우
- 소프트웨어 개발에 사용되는 언어가 프로그래머가 사용할 수 있는 제어 구조, 데이터 구조, 추상화의 종류, 알고리즘의 형태도 제약

□ 문제에 따라 적합한 언어를 선택할 수 있는 배경이 향상

- 프로그래머들이 언어와 언어 구조들에 더 폭넓게 잘 알고 있으면 문제를 해결할 수 있는 특징들을 갖는 언어를 더 잘 선택할 수 있음

□ 새로운 언어를 쉽게 배울 수 있는 능력이 향상

- 이미 알고 있는 언어에 대한 이해를 바탕으로 새로운 언어도 용이하게 이해할 수 있음
- 예) 객체 지향 언어인 C++를 잘 아는 사람은 모르는 사람에 비해 JAVA를 쉽게 배울 수 있음

프로그래밍 언어 학습 이유

□ 구현의 중요성에 대한 이해

- 어떤 경우에는 구현상의 고려 사항을 이해함으로써 언어가 왜 이렇게 설계되었는지를 이해할 수 있음

□ 이미 알고 있는 언어에 대한 나은 사용

- 프로그래밍 언어의 개념들을 학습함으로써 프로그래머는 이전에는 몰랐던 다양한 기능들을 알게 되고 이를 활용해서 프로그램을 작성할 수 있는 능력 향상

□ 전반적인 전자계산 분야의 이해도 향상

- 프로그래밍 언어 중에서 우수한 기능을 가진 언어보다 그렇지 않은 언어가 더 광범위하게 사용되는 것에 대해 이해할 수 있게 됨
- 예) Fortran vs. Algol 60

프로그래밍 영역(Domain)

□ 과학 응용 분야

- 최초 컴퓨터(1940년대)의 응용 분야
- 단순한 데이터 구조 사용
- 다량의 실수형 (floating point) 연산
- 사용 언어 : Fortran(1956), Algol 60
 - ✓ Fortran보다 상당히 나은 후속 언어는 없으며, 과학 분야에서 아직까지 사용되고 있음

□ 사무 응용 분야

- 보고서 작성, 10진수(BCD) 및 문자열을 주로 다룸
- 사용 언어 : COBOL(1960)

□ 인공지능 분야

- 인공지능 분야는 수치 계산보다는 기호 계산으로 특징지어지는 광범위한 컴퓨터 응용 분야
- 사용언어 : LISP(1959, McCarthy), Scheme, Prolog

프로그래밍 영역 (Domain)

□ 시스템 프로그래밍

- 시스템 소프트웨어 : 한 컴퓨터 시스템의 운영체제와 프로그래밍 지원 도구들 총괄
- 시스템 소프트웨어는 지속적으로 사용되므로 효율적이어야 함
- 외부 장치와의 소프트웨어 인터페이스를 작성할 수 있도록 저급 수준의 특징을 포함해야 함
- 사용 언어 : Algol, PL/I, C

□ 웹 소프트웨어

- 동적 web문서를 생성을 위해서 다양한 언어들이 사용
- Markup 언어 : XHTML
- Scripting 언어 : PHP
- 범용 언어 : java

프로그래밍 언어 분류

□ 명령형(Imperative) 언어

- 주로 변수(variable), 할당문(assignment statement), 반복문(iteration)등을 이용하여 프로그램을 작성함.
- 예) C, Pascal, Fortran, Cobol, Algol, Basic등

□ 함수형(Functional) 언어

- Parameter를 받아 실행하는 함수(function)를 이용하여 프로그램을 작성
- 예) LISP, Scheme, ML, Haskell, Erlang 등

□ 논리(Logic) 언어

- 순서가 중요하지 않은 규칙을 기반으로(rule-based) 프로그램을 작성
- 예) Prolog

□ 객체-지향(Object-oriented) 언어

- 명령형 언어의 발전된 형태로서, 명령형 언어에 기반을 둠
- 데이터의 캡슐화(encapsulation), 속성의 상속(inheritance), 동적 바인딩 등을 이용하여 프로그램을 작성함
- 예) C++, Java, Smalltalk

□ 가독성 (Readability)

❖ 프로그램을 얼마나 쉽게 읽고 이해할 수 있는지에 대한 것

1. 간결성(Simplicity)

- 너무 많은 종류의 문장을 지원하는 언어는 좋지 않음
 - ✓ 언어가 지원하는 모든 종류의 문장을 잘 알기 어려움
 - ✓ 프로그래머마다 주로 사용하는 문장이 다르기 쉬움
- 동일한 기능의 연산 표현 방법이 많을 수록 좋지 않음
 - ✓ 변수 k의 값을 1만큼 증가하기 (k++, ++k, k+=1, k=k+1)
- 연산자 overloading (Operator overloading)
 - ✓ 하나의 연산자(operator)에 여러 type의 피연산자(operands)를 사용할 수 있음
 - ✓ 연산자 overloading은 프로그램을 이해하는데 좋지 않은 기능
 - `int x, y; float n, m; x + y, n + m`
- 함수형(functional) 언어가 간결성에서 우수함

□ 가독성 (Readability)

2. 직교성(Orthogonality)

- 언어의 모든 구성 요소를 임의로 조합해서 프로그램을 작성할 수 있는 기능
- 참고 : 수학에서의 직교성, x 축과 y 축으로 이루어진 2차원 평면의 모든 점은 x 성분과 y 성분의 조합으로 표현
- 예) m 과 n 의 최대값에 3을 곱한 값을 변수 `max3`에 저장하기
 - ✓ `max3 = 3 * (if (m>n) m else n);`
- 비교적 소수의 구성 요소들을 비교적 소수의 방법으로 조합해도 어떤 표현이든지 가능
 - ✓ 간결성 조건 충족
- 직교성이 부족한 언어에서는 구성 요소의 사용 규칙에 예외적인 경우가 많음
 - ✓ C언어에서 구조체는 `function`의 결과로 `return`할 수 있으나, 배열은 `return`할 수 없음

□ 가독성 (Readability)

2. 직교성(Orthogonality)

- 완벽에 가까운 직교성을 지원하는 언어는 문제를 유발하기도 함
 - ✓ 상식적이지 않은 조합은 표현이 복잡해지므로 가독성을 저하시킬 수 있음(복잡성)
 - ✓ 예) (Algol) : (if (a>b) x else y) = 5; // c언어는 지원하지 않음
- 함수형(functional) 언어가 직교성에서 우수함

□ 가독성 (Readability)

3. 데이터 타입과 구조체(Data types and Structures)

- 적절한 data type이 지원되어야 함
 - ✓ C언어(original 버전)는 bool타입을 지원하지 않아 가독성이 좋지 않다.
 - ✓ 예) bool 타입을 지원하는 언어 : `tag = true;`
C언어 : `tag = 1; (int or bool?)`
- 다양한 자료 구조(data structure)를 data type으로 정의할 수 있어야 함

4. 문법의 구문(syntax)

- 식별자(identifier)의 형태
 - ✓ 길이에 제한이 있는 경우 가독성이 떨어짐
예) A, A3, Avrge, AverageOfScore
 - ✓ 특수 문자를 식별자에 사용가능하면 가독성이 좋음
예) AverageOfScore, average_of_score

□ 가독성 (Readability)

4. 문법의 구문(syntax)

- 특수 단어(Special words)

- ✓ 의미를 자연스럽게 알 수 있는 특수 단어들은 가독성에 좋음
- ✓ if, while, for, then, else, begin, end, class, ...

- 형식과 의미(Form and meaning)

- ✓ 표현 형식은 같거나 비슷한데 경우에 따라 다른 의미를 가지면 가독성이 떨어짐
- ✓ `static int gvar;`
- ✓ `int foo(...) { static int lvar ...}`

□ 작성력 (Writability)

❖ 새로운 프로그램을 작성하는데 언어가 얼마나 쉽게 사용될 수 있는지에 대한 것

1. 간결성(Simplicity)

- 소수의 구성 요소(constructs, primitives)
- 구성 요소들의 조합에 대한 소수의 규칙

2. 직교성

- 모든 구성 요소의 임의의 조합이 가능하므로 작성력이 좋음
- 지나친 직교성은 가독성에 좋지 않은 면이 있음

예) (Algol) : (if (a>b) x else y) = 5; // C언어는 지원하지 않음

□ 작성력 (Writability)

3. 추상화의 지원

- 데이터의 추상화(Data abstraction)
 - ✓ 복잡한 자료구조(data structure)를 정의해서 사용할 수 있는 기능 지원
예) stack, queue, tree, graph, ...
- 프로세스의 추상화(Process abstraction)
 - ✓ 복잡한 연산(operation)을 정의해서 사용할 수 있는 기능 지원

□ 작성력 (Writability)

4. 표현력

- 기존의 구성 요소와 동일한 기능을 수행하지만 좀 더 편리한 구문 지원
 - ✓ switch(Algol) : 중첩된 if-then-else
 - ✓ for : counter를 사용하는 while

예) $k++$, $++k$, $k+=1$, $k=k+1$: 1을 증가시키는 다양한 방법

- APL 언어는 매우 강력한 연산자를 지원한다.
 - ✓ 행렬 더하기, 곱하기, 전치 행렬 구하기 등이 하나의 연산으로 가능
 - ✓ 작성력은 매우 뛰어나나, 가독성이 매우 떨어짐

예) 행렬 A와 B 곱하기 : $A \times B$

□ 신뢰성 (Reliability)

❖ 모든 조건에서 프로그램이 주어진 명세에 따라 정확히 수행되는
지에 대한 것

1. 타입 검사(Type checking)

- 신뢰도에 영향을 주는 매우 중요한 요소
- Compile할 때나 프로그램 실행시 type error를 검출
- Type error는 일찍 발견할 수록 오류 수정 비용이 적음
 - ✓ 타입 검사는 컴파일할때 하는 것이 바람직함
 - ✓ 프로그램 실행시 이루어지는 타입 검사는 비용이 큼

2. 예외 처리(Exception handling)

- 프로그램 실행시 run-time error가 발생한 경우, 이를 인식하고 오류
를 처리할 수 있는 필요한 조치를 취할 수 있는 기능
 - ✓ C++, Java, Ada : 예외처리 기능 지원
 - ✓ C, Fortran : 예외처리 기능 지원하지 않음

□ 신뢰성 (Reliability)

3. 별칭 (Aliasing)

- 메모리의 한 저장 장소를 여러 이름으로 접근할 수 있는 기능
- 신뢰성에 부정적인 영향을 끼침

4. 가독성과 작성력 (Readability and writability)

- 작성하기 쉽고 이해하기 쉬운 프로그램을 작성할 수 있는 언어로 작성된 프로그램은 오류 가능성이 적음
- 자연스러운 방법으로 프로그램을 작성할 수 없는 언어는 자연스럽지 않은 방법으로 프로그램을 작성해야 하므로 오류 가능성이 크게 됨

□ 비용 (Cost)

❖ 프로그램에 관련된 시간 및 경비는 얼마나 되는지에 대한 것

❖ 비용에 영향을 끼치는 요인

- 프로그래머 양성(교육) 비용
 - ✓ 단순성과 직교성 영향을 받음
- 프로그램 작성(개발) 비용
 - ✓ 작성력에 영향을 받음
- 작성된 프로그램의 컴파일 비용

□ 비용 (Cost)

- 프로그램을 실행시키는 비용
 - ✓ 언어 설계의 영향을 받음
 - ✓ Compile 비용과 Trade-off관계에 있음
 - ✓ 최적화(Optimization)하면, compile 시간은 길어지고, 실행시간은 짧아짐
- 언어 구현 시스템(Compiler/Interpreter)의 비용
 - ✓ 무료 또는 저렴한 경우, 널리 사용됨->JAVA
 - ✓ 고가인 경우, 잘 사용되지 않음(Ada)
- 신뢰성 부족에 따른 비용
 - ✓ 신뢰도가 낮은 언어로 작성된 프로그램은 오류로 인한 비용 부담이 매우 큼
- 유지보수 비용
 - ✓ 비교적 오래 사용되는 대형 소프트웨어 시스템의 유지보수 비용은 개발 비용의 2~4배에 달함
 - ✓ 가독성과 작성력의 영향을 받음

□ 컴파일러 구현 방법(compiler)

- 소스 프로그램을 실행 가능한 기계 언어로 번역
- 기계 코드는 매우 빠르게 실행
- 링킹/적재

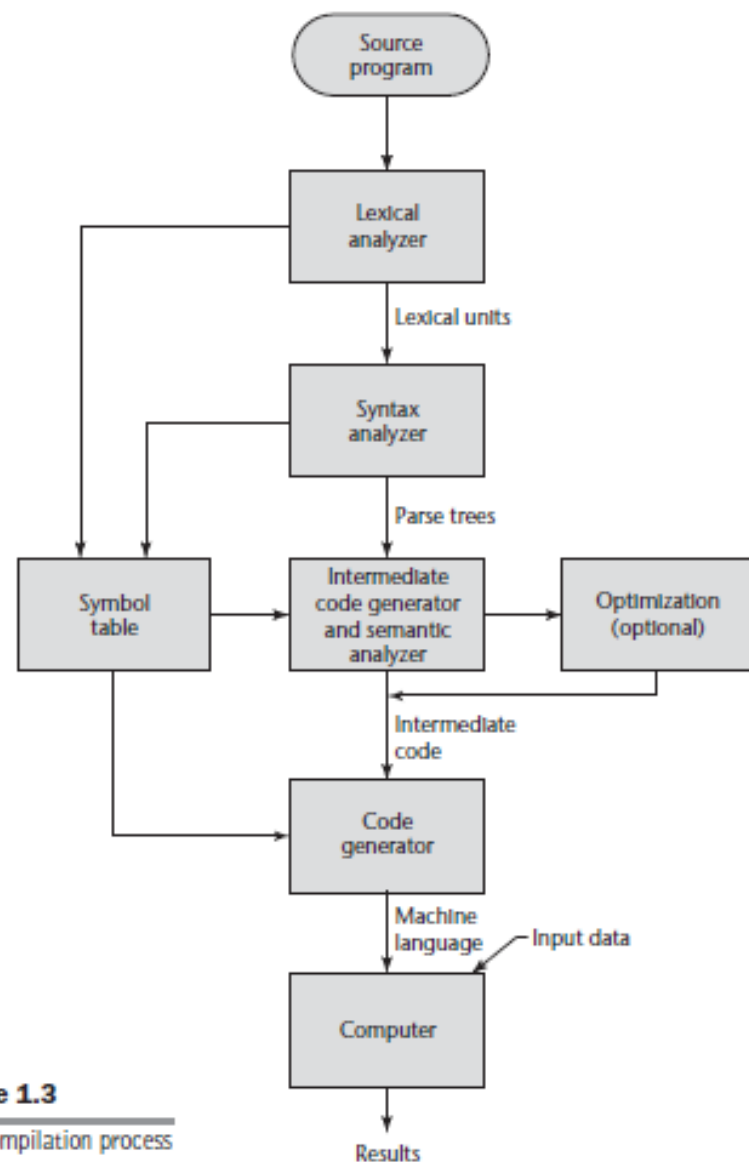


Figure 1.3
The compilation process

□ 순수 해석 (interpreter)

- 프로그램이 인터프리터에 의해서 해석되면서 실행
- 인터프리터는 언어에 대한 가상기계를 제공
- 단순한 구조의 언어에 적합
 - APL, LISP, JavaScript 등

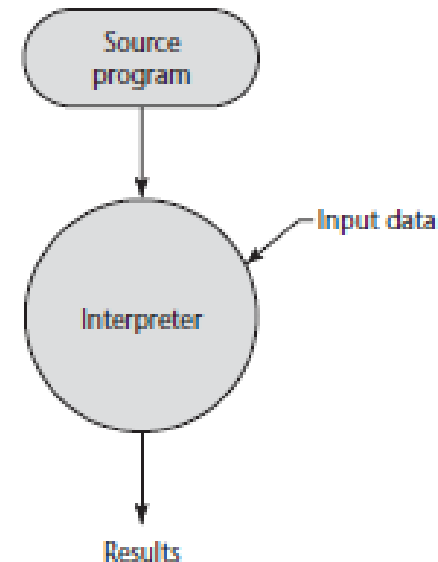


Figure 1.4

Pure interpretation

구현 방법

□ 컴파일러와 순수해석의 절충

- 프로그램을 중간 코드로 번역하고, 이 중간 코드를 해석
- JIT(Just-In-Time)
 - 메소드 호출시 기계코드로 번역하여 실행
 - Java, .NET 언어

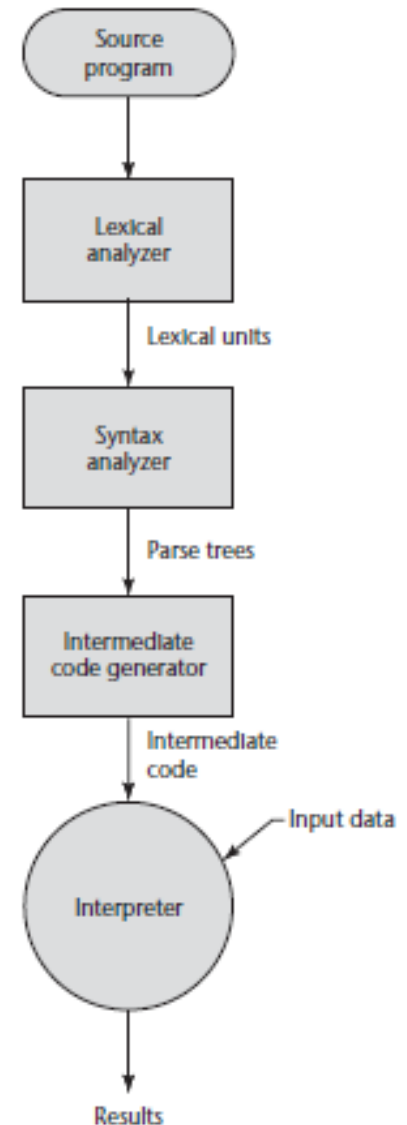


Figure 1.5

Hybrid implementation system

□ 전처리기 (preprocessor)

- 프로그램을 번역하기 직전에 프로그램을 처리
- 전처리기 명령어는 프로그램에 포함
- 기본적으로는 매크로 확장기
- Ex)
 - `#include "myLib.h"`
 - `#define max(A, B) ((A) > (B) ? (A) : (B))`

Thank you for your attention !!

Q and A