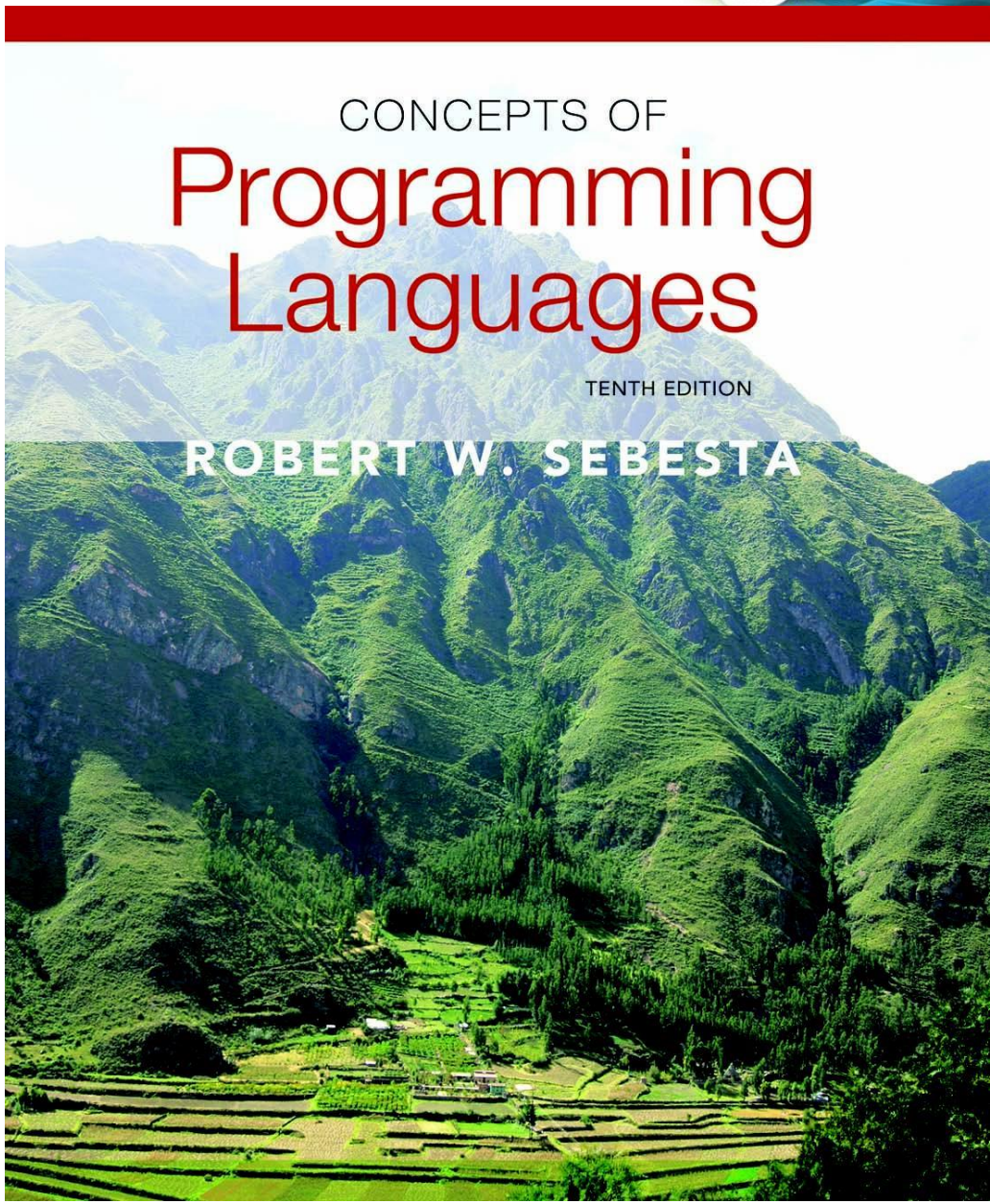
프로그래밍 언어론 (Programming Languages)

2018년 2학기

담당교수 : 박두순

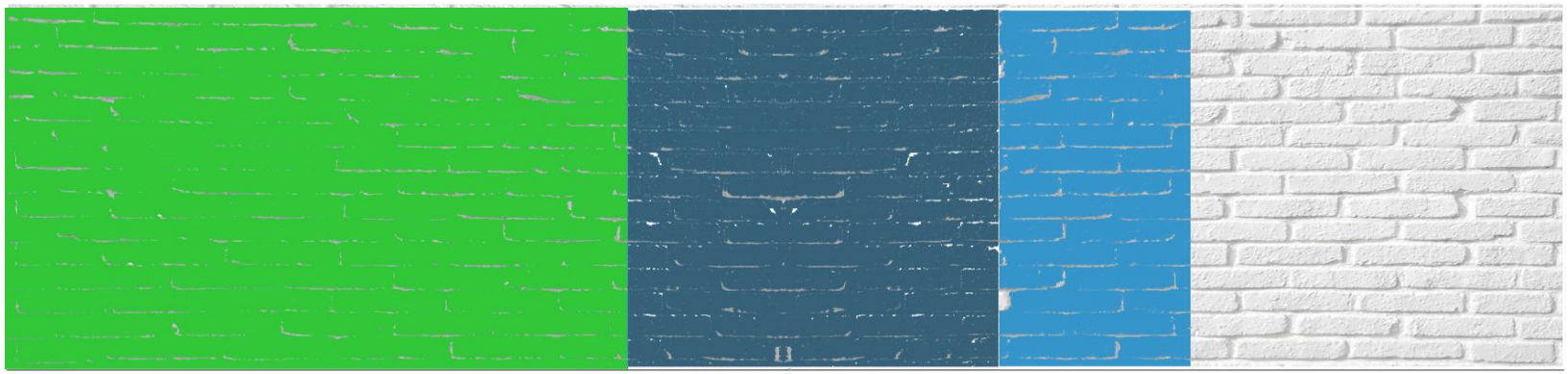


첫 시간 학습 내용

- **목적 :** 현재까지 나와있는 프로그래밍 언어가 많아서 모두 공부하기는 어렵다. 그래서 많은 프로그래밍 언어에서 다루고 있는 개념들을 공부하여 생각을 표현하는 능력 함양, 프로그래밍 언어를 선택할 수 있는 능력을 증대, 새로운 프로그래밍 언어를 배우기 쉽게 해주며, 유용한 프로그래밍을 구사할 수 있는 능력을 증대시켜 컴퓨팅에 대한 전체적인 개념을 이해하기 위한 것이다.
- **교재 :** 프로그래밍 언어론(10/E), 유원희, 하상호 역, 퍼스트북, 2012
- **참고문헌 :**
  - 1. 생각하는 프로그래밍, 윤성준,조상민 역, 인사이트, 2014
  - 2. 프로그래머처럼 생각하기, 김무향 역, 에이콘, 2014
- **성적 :** 중간고사 30%, 기말고사 40%, 과제물 20%, 출석 10%
- **강의노트 :** http://myclass.sch.ac.kr(학교 LMS 시스템)
- **선수과목 :** C 프로그래밍, 이산수학, 자료구조
- **강의 방법 및 리포트 :** 교재 따라감, 여러 가지 언어 다루기(5가지 언어 프로그래밍 숙제),연습문제풀이 – 절대 Cheating은 불가
- **리포트 제출 :** skk4248@naver.com

블로그에서 인용

❖ 페인트 공 알고리즘 (조엘 온 소프트웨어 中, 조엘 스폴스키)



❖ **생각 없이 strcat을 사용한 게 부끄럽게만 느껴진다.**

❖ 컴퓨터에서 가장 중요한 것은 OS, 자료구조, 컴파일러, 알고리즘, 네트워크 등 눈에 보이지 않는 것들이다.

❖ 우리는 진정한 프로그래머를 원한다.

컴퓨터학과 교육 과정

❖ACM/IEEE CS 교육과정(1978, 1988, 1991, 2001, 2004 CE, 2005 computing, 2008년, 2013년)

❖전산학과 학부과정 : 11개 subject→13개 subject →18개 knowledge

Knowledge area \ model	Curricula 91	Curricula 01	Curricula 08	Curricula 13	
				Tier1	Tier2
Algorithms & Complexity	47	31	31	19	9
Architecture and Organization	59	36	36	0	16
Intelligent Systems	9	10	10	0	10
Information Management	9	10	11	1	9
Human and Computer Interaction	8	8	8	4	4
Programming Fundamentals(software development fundamentals)	12	38	47	43	0
Computational Science	7	0	0	1	0
Operating Systems	31	18	18	4	11
Programming Languages	46	21	21	8	20
Software Engineering	44	31	31	6	22
Social and Professional Issues	11	16	16	11	5
Discrete Structures		43	43	37	4
Networking and Communications		15	15	3	7
Graphics and Visual Computing		3	3	2	1
Information Assurance and Security				3	6
Platform-based Development				0	0
Parallel and distributed computing				5	10
Systems Fundamentals				18	9
Total number of hours	283	280	290	165	143

컴퓨터학과 교육 과정

1/1	1/2
C 프로그래밍, C 프로그래밍 실습	C 프로그래밍 활용, C 프로그래밍 활용 실습
컴퓨터 개론	공업 수학
창의공학 설계	소프트웨어기초설계
	인터넷 프로그래밍
2/1	**2/2**
자료구조1, 자료구조1 실습	자료구조2, 자료구조2 실습
이산수학, 이산수학연습	소프트웨어공학
C++프로그래밍, C++프로그래밍실습	JAVA 프로그래밍, JAVA 프로그래밍 실습
컴퓨터 아키텍쳐	정보 이론
	프로그래밍 언어론
	오퍼레이팅시스템
3/1	**3/2**
데이터통신	컴퓨터보안
오토마타	컴파일러
데이터베이스	데이터베이스프로그래밍
컴퓨터그래픽스	게임프로그래밍
시스템분석 및 설계	설계 패턴
엔터프라이즈프로그래밍	캡스톤디자인1
4/1	**4/2**
네트워크 프로그래밍	유비쿼터스 컴퓨팅
모바일 프로그래밍	융합 소프트웨어
IOT기초이론	지능형 소프트웨어
임베디드 소프트웨어	휴먼 IOT 응용
캡스톤디자인2	

프로그래밍 언어론

- **1. Overview of programming languages :** 초기 언어들(Algol, Fortran, Cobol); procedural language(Algol, PL/I, Pascal, Euclid, Modula-2, Ada); non-procedural language(functional(Lisp), logic(Prolog), object-oriented(Smalltalk), parallel)
  - ☞교재 1,2장, 12장(object-oriented programming), 15장(functional), 16장(logic)
- **2. Virtual machine :** The concept of a virtual machine; Hierarchy of virtual machines; Intermediate languages; Security issues arising from running code on an alien machine
  - ☞ 교재 1장
- **3. Introduction to language translation :** Comparison of interpreters and compilers; Language translation phases(lexical analysis, parsing, code generation, optimization); Machine-dependent and machine-independent aspects of translation
  - ☞ 교재 3,4장
- **4. Declarations and types :** The conception of types as a set of values with a set of operations; Declaration models (binding, visibility, scope, and lifetime); Overview of type-checking; Garbage collection;
  - ☞ 교재 5,6장

프로그래밍 언어론

- **5. Abstraction mechanisms :** Procedures, functions, and iterations as abstraction mechanisms; Parameterization mechanisms (reference vs. value); Activation records and storage management; Type parameters and parameterized types; Modules in programming languages;
  - ☞ 교재 11장
- **6. Object-oriented programming :** Object-oriented design; Encapsulation and information-hiding; Separation of behavior and implementation; Classes and subclasses; Inheritance(overriding, dynamic dispatch); Polymorphism (subtype polymorphism vs. inheritance) ; Classes hierarchies; Collection classes and iteration protocols ; Internal representations of objects and method tables
  - ☞교재 12장
- **7. Functional programming :** Overview and motivation of functional languages; Recursion over lists, natural numbers, trees, and other recursively-defined data; Pragmatics(debugging by divide and conquer; persistency of data structures); Amortized efficiency for functional data structures; Closures and uses of functions as data(infinite sets, streams);
  - ☞교재 15장

프로그래밍 언어론

- **8. Language translation systems :** Application of regular expressions in lexical scanners; Parsing; Symbol table management; Code generation ; instruction selection and register allocation; Optimization techniques; Building syntax-directed tools;
  - ☞4장, 컴파일러, 오토마타
- **9. Type systems :** elementary data type(integer, real); structure data type(array, record)
  - ☞교재 6장
- **10. Programming language semantics**
- **11. Programming language design**

프로그래밍 언어론(교재)

- 1. 기본적인 사항(프로그래밍 언어를 배우는 이유, 프로그래밍 영역, 언어평가기준, ..)
- 2. 프로그래밍 언어의 발전사(FOTRAN, COBOL, ALGOL, PASCAL, ..)
- 3. 구문과 의미론
- 4. 어휘 분석과 구문 분석
- 5. 이름, 바인딩, 영역
- 6. 데이터 타입
- 7. 식과 배정문
- 8. 문장 수준 제어 구조
- 9. 부프로그램

- C 언어 ⇒자료구조 ⇒프로그래밍언어론 ⇒형식 언어 및 오토마타 ⇒ 컴파일러

제 1 장

기본적인 사항

제 1 장 기본적인 사항

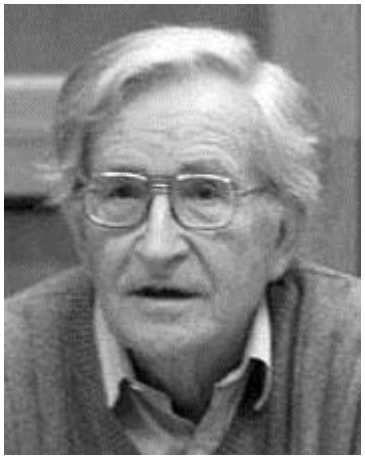
❖목적 :

- 1. 프로그래밍 언어란?
- 2. 프로그래밍 언어론을 배우는 이유
- 3. 프로그래밍 영역
- 4. 언어 평가기준
- 5. 언어 설계에 미친 영향
- 6. 언어 부류
- 7. 언어설계 절충
- 8. 구현 방법
- 9. 프로그래밍 환경

프로그래밍 언어란?

- **프로그래밍 언어(programming language)란 -**
  - 사람과 컴퓨터가 의사 소통을 하기 위한 수단으로 알고리즘과 자료구조를 서술하기 위한 표기 체제
- **프로그래밍 언어가 나오게 된 이유 -**
  - 언어란 의사소통의 수단
- **언어의 종류 – 의사 소통의 대상이 누구냐?**
  - 자연 언어(natural language) – 사람과 사람
  - 형식 언어(formal language) – 사람과 기계
- **형식 언어의 종류**
  - 촘스키(Chomsky) 분류 :
    - ✓ type 0(재귀 열거 언어(recursively enumerable language))
    - ✓ type 1(문맥 인식 언어(context-sensitive language))
    - ✓ type 2(문맥 자유 언어(context free language))
    - ✓ type 3(정규 언어(regular language))
  - Type 2의 범위가 프로그래밍 언어

프로그래밍 언어란?



❖ **촘스키(Chomsky, Noam)**

- 1928년 출생, 미국 MIT 교수
- 현대 언어학자로서 변형 생성 문법의 창시자이다. 인간은 유한개의 규칙에 따라 무한개의 문장을 만들 수 있다고 주장하였으며 그러한 규칙을 수학적인 엄밀성을 가지고 정식화하고자 하였다.
- 경력사항
  - ✓ 변형 생성 문법 창시

[네이버 지식백과] 촘스키 [Chomsky, Noam] (철학사전, 2009., 중원문화)

프로그래밍 언어란?

- **프로그래밍 언어의 종류 – 기계어, 어셈블리 언어, 고급언어**
  - 기계어(machine language) :
    - ✓ 컴퓨터가 직접 해독할 수 있는 2진 숫자(binary digit)로 표현된 언어.
    - ✓ 그러나 이해하기 어렵고, 컴퓨터 구조에 대한 충분한 지식이 없으면 프로그램 작성을 할 수 없기 때문에 범용성이 부족하고, 숫자(0과 1)를 사용함에 따라 프로그래머의 프로그램을 작성하는 데 시간이 많이 걸린다.
  - 어셈블리(assembly) 언어 :
    - ✓ 0과 1로 구성된 기계어 대신 더하기에 ADD, 빼기에 SUBT 등 대응하는 기호를 사용한 언어, ADD R, A
    - ✓ 프로그래밍 작업이 어려움
    - ✓ 어셈블러(assembler) 필요.
  - 고급언어 :
    - ✓ 사람 중심 언어, C JAVA
    - ✓ 컴파일러(compiler) 필요
- **명령적 언어의 특징 – 명령어를 순차적으로 수행, 폰 노이만 구조 따름**
  - 기억 장소를 표시하는 변수 사용
  - 변수의 값을 변경시키기 위한 배정문 사용
  - 반복문 사용
  - 명령의 순차적인 실행

1.1 프로그래밍 언어를을 배우는 이유

- **현재까지 나와있는 프로그래밍 언어가 많아서 모두 공부하기는 어렵다. 그래서 많은 프로그래밍 언어에서 다루고 있는 개념들을 공부하여**
  - 1. 생각을 표현할 수 있는 능력이 향상된다.
  - 2. 적합한 언어를 선택할 수 있는 능력이 향상된다.
  - 3. 새로운 언어를 배울 수 있는 능력이 향상된다.
  - 4. 구현의 중요성에 대해서 보다 많이 이해한다.
  - 5. 이미 알고 있는 언어에 대한 더 나은 사용
  - 6. 컴퓨터 분야의 이해가 향상된다.

1.1 프로그래밍 언어론을 배우는 이유

- **1. 생각을 표현할 수 있는 능력이 향상된다.**
  - 여러 가지 언어나 단어가 풍부하면 표현력이 좋음
  - 이해하는 것과 사용하는 것(아는 것)의 차이
    - ✓ 언어를 많이 사용한다고 그 언어의 특징을 아는 것은 아니다.
    - ✓ [예] 재귀법(recursion) - 잘 사용하면 근사하고 효율적인 알고리즘 구성, 잘못 사용하면 실행시간 기하급수적으로 증가
      - ♣ 정의 - procedure that can call itself (자신을 정의할 때 자기 자신을 재참조하는 방법)
        - factorial(int n)
        - { **if** (n <= 1) **return** 1;
        - **else return** n * factorial(n-1); }
      - ♣ 종류 - direct recursive , indirect recursive
      - ♣ 허용 언어 - C, Pascal 허용 않는 언어 - Fortran, Cobol
      - ♣ iterative와 recursion에 대해서 정의부터 장단점 비교
      - ♣ Iterative와 recursion과 함수는 어떤 공통점과 차이점은 있는가?
      - ♣ Activation record, call by value, stack
    - ✓ [예] array와 record가 어떻게 만들어지고 다루어 지는가 이해 -
      - ♣ Array와 record에 대해서 정의부터 장단점 비교
      - ♣ Array에 대한 부연 설명
        - 열 우선(column major) – Fortran
        - 행 우선(row major) - C, Pascal

1.1 프로그래밍 언어론을 배우는 이유

♣ 행렬의 곱과 관련된 프로그래밍 기법

```
Algorithm product(A, B, C)
  Begin
    for i = 1 to m
      for j = 1 to s
        begin
          c_ij = 0
          for k = 1 to n
            c_ij = c_ij + a_ik b_kj
          endfor
        end
      endfor
    endfor
  end
```

♣ 실행 속도를 빠르게 하기 위한 방법 : i, j, k 루프를 교환해 보는 것 – 결과는 같고 지역성(locality) 효과(공간적, 시간적)

✓ [예] coroutine - C나 Fortran에 coroutine이 필요하면 이 개념을 사용

♣ subprogram과의 차이

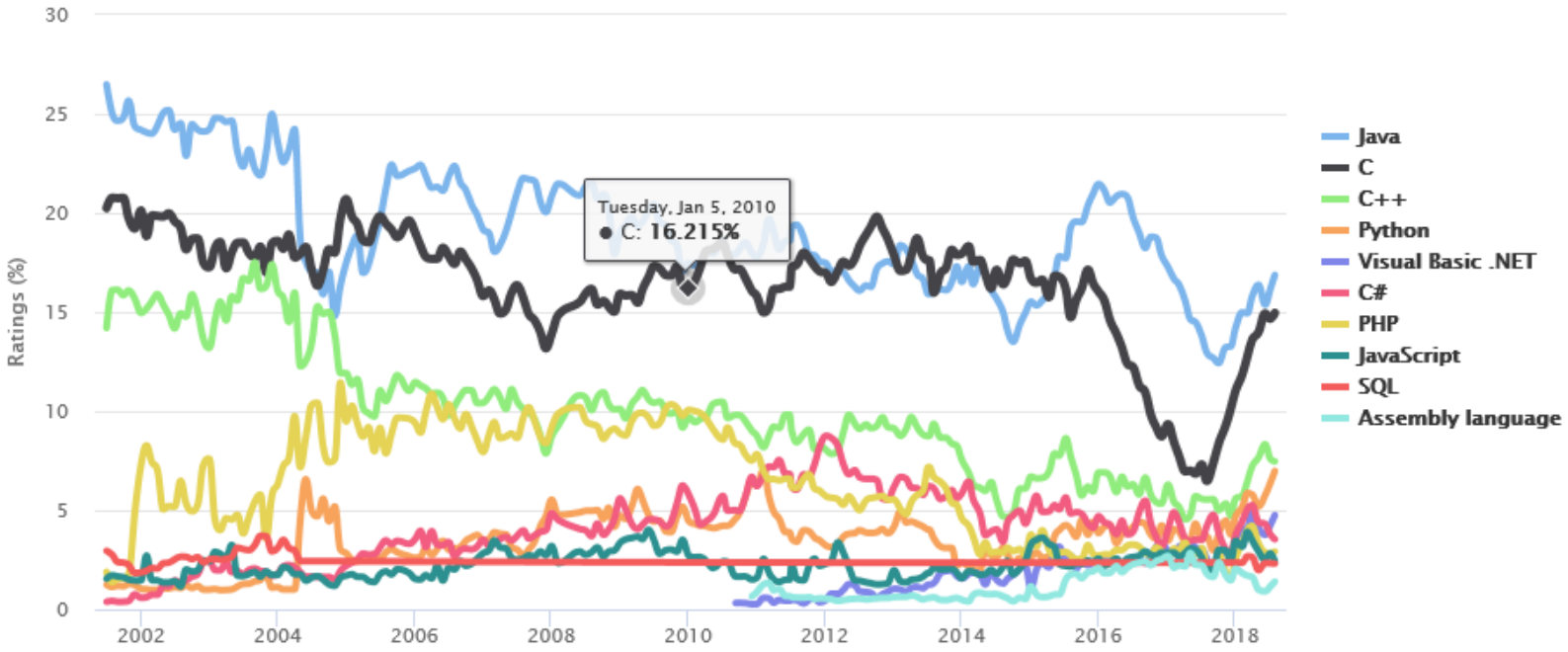
✓ [예] 연상 배열(associative arrays) – C 언어에서 연상 배열이 필요하다면 Perl, Php 등에서 사용되는 개념을 사용(참조번호와 문자열에 의해서 indexing 됨) – 첨자변수 사용 못함.

1.1 프로그래밍 언어론을 배우는 이유

- **2. 적합한 언어를 선택할 수 있는 능력이 향상된다.**
  - 과학기술계산 - C, Fortran, Ada
  - 사무처리 - Cobol
  - 문자처리 - Snobol 4
  - 인공지능 - Lisp, ML, Prolog

- **3. 새로운 언어를 배울 수 있는 능력이 향상된다.**
  - TIOBE – 프로그래밍 언어의 상대적 사용 지수를 나타내는 인덱스 (http://www.tiobe.com/tiobe_index/index.htm)
    - ✓ The ratings are based on the number of skilled engineers world-wide, courses and third party vendors.

1.1 프로그래밍 언어를을 배우는 이유



- 2018년 8월 기준으로 보면 Java, C, C++, Python, Visual-Basic.net, C#, PHP, Javascript, SQL, Assembly language

1.1 프로그래밍 언어를을 배우는 이유

Very Long Term History

To see the bigger picture, please find below the positions of the top 10 programming languages of many years back. Please note that these are *average* positions for a period of 12 months.

Programming Language	2018	2013	2008	2003	1998	1993	1988
Java	1	2	1	1	14	-	-
C	2	1	2	2	1	1	1
C++	3	4	3	3	2	2	5
Python	4	7	6	11	22	17	-
C#	5	5	7	8	-	-	-
Visual Basic .NET	6	12	-	-	-	-	-
JavaScript	7	10	8	7	19	-	-
PHP	8	6	4	5	-	-	-
Ruby	9	9	9	18	-	-	-
R	10	24	45	-	-	-	-
Perl	13	8	5	4	3	11	-
Objective-C	16	3	40	53	-	-	-
Ada	28	20	18	14	12	5	3
Fortran	30	25	21	12	6	3	15
Lisp	31	11	16	13	7	6	2

1.1 프로그래밍 언어튠을 배우는 이유

- **IEEE Spectrum ranking—(https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages)**
  - **1.** Python(A scripting language that is often used by software developers to add programmability to their applications, such as engineering-analysis tools or animation software.)
  - **2.** C++
  - **3.** C(C is used to write software where speed and flexibility is important, such as in embedded systems or high-performance computing.)
  - **4.** Java(Designed to allow the creation of programs that can run on different platforms with little or no modification, Java is a popular choice for Web applications.)
  - **5.** C#
  - **6.** PHP
  - **7.** R(통계 계산 그래픽을 위한 언어)
  - **8.** JavaScript
  - **9.** Go(구글 개발언어)
  - **10.** Assembly

1.1 프로그래밍 언어론을 배우는 이유

- **4. 구현의 중요성에 대해서 보다 많이 이해한다.**
  - 구현상의 고려사항들을 이해함으로써 언어가 왜 이렇게 설계되었는지를 이해할 수 있다.
- **5. 이미 알고 있는 언어에 대한 더 나은 사용**
  - 프로그래밍 언어의 특징을 알면 더 나은 사용 가능.
- **6. 컴퓨터 분야의 이해가 향상된다.**

1.2 프로그래밍 영역

- 분류 방법은 여러 가지가 존재 –
  - 특성별, 연대별, 절차와 선언, block과 non-block
- 특성별 분류(**1.2 프로그래밍 영역 분류**)
  - **1. 과학 응용분야(Scientific applications)** – 큰 부동소수점(floating number) 연산
    - **Fortran - I, II, III, IV, 77, 86, 90, 95, 2003, 2008, HPF, FORTRAN D, CEDAR FORTRAN, KSR**
      - **달에 가는 아폴로 프로젝트**
        - 이것은 한 명의 인간에게는 작은 발걸음이지만, 인류에게는 위대한 도약이다. That's one small step for (a) man, one giant leap for mankind.) - 닐 암스트롱, 1969년 7월 20일 달에 최초의 발자국을 남기며
      - **매개변수 전달 방식 – 참조호출(call by reference)**
        - 매개변수 전달 : 매개변수 사이에 값을 주고받는 것
          - 매개변수에는 함수 정의 구문에서 기술되는 매개변수인 **형식 매개변수**(formal parameter)와 함수를 호출할 때 기술되는 매개변수인 **실 매개변수**(actual parameter)가 있다.
          - 형식 매개변수와 실 매개변수 사이에 값을 어떻게 주고받느냐에 따라 여러 가지 **매개변수 전달 방법**(parameter passing)이 있는데, 여기서는 참조 호출(call by reference, call by address, call by location), **값 호출**(call by value)을 간단히 살펴보고 자세한 것은 9장에서 다룬다.

1.2 프로그래밍 영역

- **참조 호출**은 실 매개변수의 주소를 대응되는 형식 매개변수로 넘겨주는 것이며 대표적으로 포트란, PL/I(Programming Language/One)과 C 언어의 포인터 변수에 대한 매개변수 전달 방식에 사용. 하지만 이 방법은 2개 이상의 단위 프로그램에서 2개 이상의 변수가 기억 장소를 공유함으로써 지역 변수 이외의 변수 값을 변화시키는 **부작용**(side effect)이 발생, 이는 프로그램을 읽고 이해하는 데 어려움을 줌. 부작용을 일으키는 대표적인 예로 포트란의 COMMON 문장. 하나의 단위 프로그램에서 2개 이상의 변수가 기억 장소를 공유할 때 발생하는 **별칭**(alias). 별칭의 대표적인 예는 포트란의 EQUIVALENCE 문장이다
- **값 호출**은 실 매개변수와는 별도로 형식 매개변수에 대한 기억 장소를 별도로 할당하는 방법으로, 부작용이 발생하지 않지만 기억 장소가 추가로 필요하다는 단점. 블록 구조 언어인 C나 파스칼에서 기본적으로 값 호출 방법을 사용.

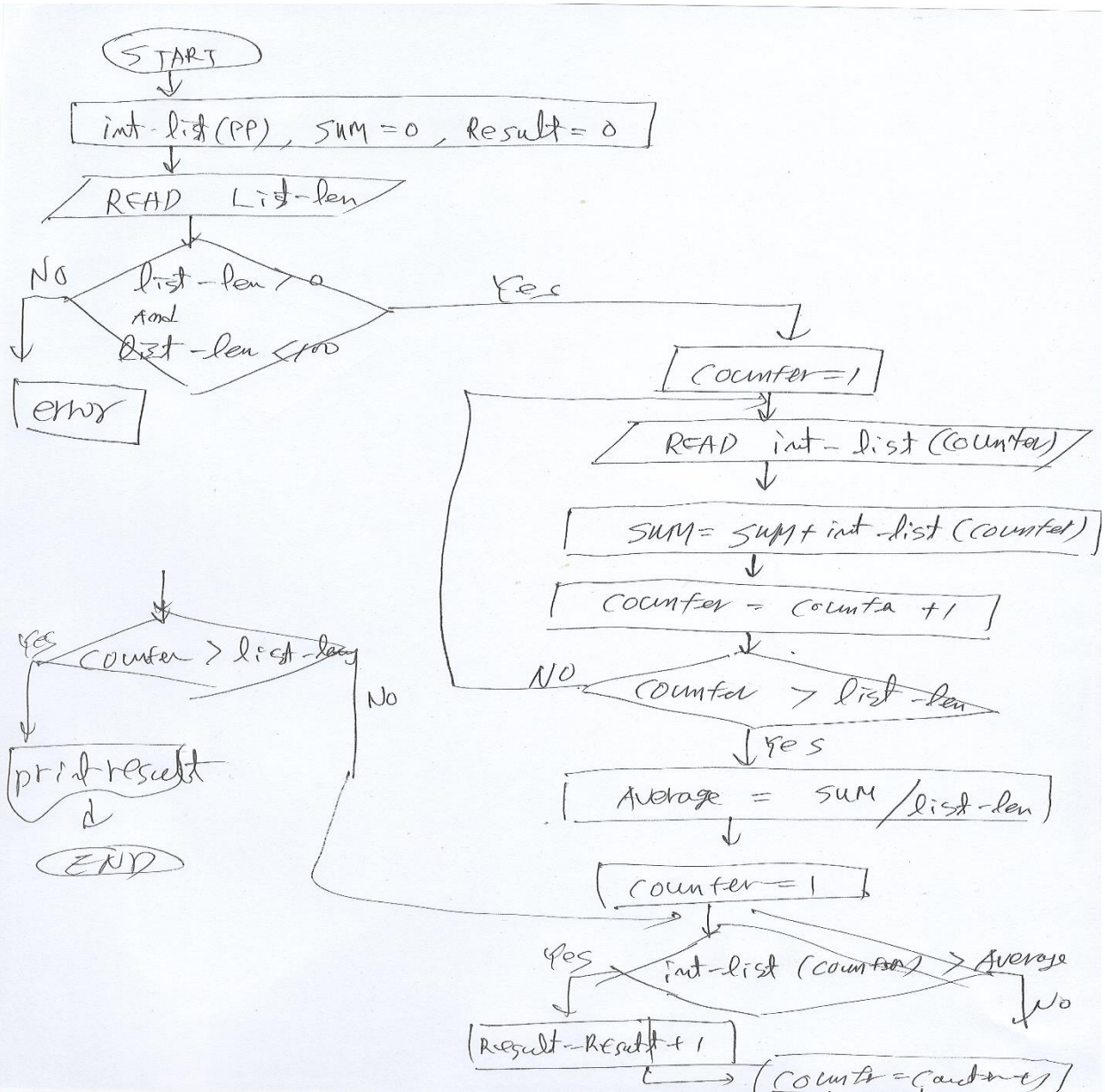
♣ **예제 p49(2장)**

♣ **똑같은 예제에 대해 각각의 언어로 프로그램 설명**

- **Flowchart로 먼저 그려보고 프로그램 설명**
- **100개 이하의 데이터를 배열로 선언하고 데이터들의 전체 합과 평균을 구한다. 평균보다 큰 데이터는 몇 개인지를 세어서 출력.**

1.2 프로그래밍 영역

합과 평균을



1.2 프로그래밍 영역

- ✓ Algol – 언어 구조가 명확하고 다른 언어들에 많은 영향을 줌.
  - ♣ free-format, 블록(block) 구조, 변수의 type선언, 재귀법(recursion)-가장 먼저 사용한 언어는 LISP, 이름 호출(call-by-name), 스택 환경, BNF(Backus Naur Form)
  - ♣ 예제 p61(2장)
- ✓ Simula67 - 객체지향기법
  - ♣ GPSS, SIMSCRIPT, SLAM => simulation 언어
- ✓ PL/I - 예외처리 기능(ON) : 0으로 나눈 나눗셈, 산술연산 overflow
  - ♣ multi-tasking : 병렬처리
  - ♣ 예제 p76(2장)
- ✓ Basic – 초보자
  - ♣ Visual basic(윈도우용 소프트웨어 개발 Gui)
  - ♣ Time sharing(시분할)
  - ♣ 예제 p70 (2장)

1.2 프로그래밍 영역

- **2. 사무 응용분야(Business applications) – 주로 I/O에 관한 문제**
  - ✓ **Cobol** – 은행이나 병원에서 사용하는 프로그램
    - ♣ 기계독립적 , Documentation, record구조(C 언어에서 구조체) 도입, 자료구조와 실행부문 분리
    - ♣ 예제 p65(2장)
  - ✓ **Flowmatic**

- **3. 인공지능 분야(Artificial intelligence) – 숫자보다는 기호(symbol)을 다룸, 강력한 리스트 처리 기능**
  - ✓ **Lisp** - 함수 언어(COMMON LISP은 함수 언어가 아님), prefix형태의 연산(polish notation)-(+ a b), garbage collection, recursion
    - ♣ IPL, Scheme, COMMON LISP
    - ♣ 예제 p54 (2장에서 설명)
  - ✓ **Snobol** - 문자일 처리
  - ✓ **Prolog** - 논리 언어

- **4. 시스템 프로그래밍(Systems programming) - 계속 사용하므로 효율성 필요**
  - ✓ **C** – p85
  - ✓ **C++** -
  - ✓ **Object –C** –

1.2 프로그래밍 영역

- **5. 스크립트 언어(Scripting languages)** - 스크립트라고 불리는 명령들의 리스트를 한 개의 실행될 파일로 작성
  - ✓ **sh(shell의 축약)** - 파일 관리나 간단한 파일 필터링 등과 같은 유틸리티 함수를 수행하는 시스템
  - ✓ **Perl** - Larry Wall이 개발, sh와 awk의 결합 형태로 시작하여 언어로 발전, CGI 프로그래밍에 이상적, p105
  - ✓ **JavaScript** - Netscape, 웹 서버와 브라우저에서 사용
    - ♣ 여기에 변수, 제어 구조, 함수 등이 추가되어 언어로 발전

- **6. 특정 목적 언어(Special purpose languages)**
  - ✓ **Web 프로그래밍 언어 – markup(XHTML), scripting(PHP), 일반목적언어(JAVA)**
    - ♣ **JAVA** - C++로부터 시작, 규모는 더욱 작아지고, 더욱 단순, 더욱 안전한 언어
    - ♣ - 내장 가전 제품을 위한 프로그래밍 언어, 신뢰성 추구
    - ♣ - Web 프로그래밍에 유용한 도구, 이식성이 강한 언어, p118
  - ✓ **APL** - 강력하고 수학적인 개념 사용, 일반적인 프로그래밍 언어의 규칙을 위배
  - ✓ **Pascal** - 분리 컴파일 기능, 유용한 문자열 조작 , p98
  - ✓ **Modula 2** - 운영체제
  - ✓ **Delphi** - 명령 언어에 객체지향 추가

Report 1

- C, JAVA, Fortran, Cobol, LISP으로 행렬의 곱을 구하는 프로그램 작성.
  - 먼저 C로 행렬의 곱을 구하는 프로그램 작성. i, j, k 등 제어 변수를 바꿔가면서 실행할 때의 실행 시간과 메모리 사용량 비교
  - C가 끝난 후 그 프로그램을 JAVA, Fortran, Cobol, LISP으로 같은 방법으로 작성해서 비교해보기.
  - 1단계 - 5개의 언어로 작성해서 실행 시간과 메모리 사용량 비교 구현하시오.
  - 2단계 – 5가지 언어에 대해서 재귀법과 반복법, 배열과 레코드, 함수를 사용할 때와 사용하지 않을 때, 매개변수를 call by value와 call by reference 등등
  - 제출기간 : 3주

1.3 언어 평가기준

❖ 언어 평가 기준 :

- 가독성, 작성력, 신뢰도, 비용
- 표1.1 평가 기준에 영향을 주는 요소

특성	평가 기준		
	가독성	작성력	신뢰도
단순성	■	■	■
직교성	■	■	■
데이터 타입	■	■	■
구문 설계	■	■	■
추상화 지원		■	■
표현력		■	■
타입 검사			■
예외 처리			■
제한된 별칭			■

1.3 언어 평가기준

- **Readability(가독성) :** 프로그램을 얼마나 쉽게 읽고 이해할 수 있느냐?
  - Overall simplicity (전반적인 단순성, 간결성) :
    - ✓ 언어가 복잡하지 않고 간결해야 함. – Pascal과 c 언어의 특성
    - ✓ 언어가 단순하지 않고 복잡해지는 특성
      - ♣ 언어의 규모가 크면 많은 기본 구조를 가지고 기능이 많아서 배우기도 어렵다.
        - 모든 기능을 배우기보다는 언어의 일부분만을 배우고 다른 특징을 무시한다. => (예) 핸드폰의 기능
      - ♣ Multiplicity(다중성) : 특정 연산을 한 가지 이상 방법으로 수행할 수 있는 특징 – 다중성이 있으므로 해서 언어가 복잡해짐
        - Java에서 1을 증가시킬 수 있는 4가지 방법
          - count=count+1, count +=1, count++, ++count
      - ♣ operator overloading (연산자 중복) : 한 연산자 기호가 하나 이상의 의미를 갖는 연산자
        - 유용하지만 사용자가 이 기능을 분력력있게 사용하지 않는다면 가독성은 감소
        - + 연산자가 정수와 부동소숫점 연산

1.3 언어 평가기준

- Orthogonality (직교성)
  - ✓ (예) 집을 지을 때 가장 공통적인 기둥을 어떻게 세울 것인가?
  - ✓ (국가나 회사)는 시스템적으로 돌아가야한다. 직교성이 좋은 것임.
  - ✓ C 언어의 매개변수 전달에서 배열은 참조 전달, 반면에 나머지 매개 변수들은 값 전달 방식이면 직교성이 부족.
    - ♣ 매개변수에 관계없이 일률적으로 적용된다면 직교성이 좋음.
  - ✓ 컴퓨팅에서 이 용어는 일종의 독립성independence이나, 결합도 줄이기decoupling를 의미한다. 하나가 바뀌어도 나머지에 어떤 영향도 주지 않으면서로 직교한다고 할 수 있다.
- 데이터 타입
  - ✓ 충분한 데이터 타입을 제공하면 가독성이 높아짐.
    - ♣ 블리안 타입이 허용되지 않는 언어에서는 timeout = 1 이라고 배정문을 사용하지만 의미가 분명하지 않다. 반면에 블리안 타입을 허용하면 timeout=true 이라고 하면 의미가 분명하다.
- 구문 설계
  - ✓ 구문은 프로그램의 가독성에 상당한 영향을 미친다.
    - ♣ Fortran 95에서는 지정어인 Do, End도 올바른 변수 이름이다. 이런 이름을 사용하면 변수인지 지정어인지를 구별하기가 어렵다. 즉, 가독성 떨어짐
  - ✓ 3장에서 자세히

1.3 언어 평가기준

- **Writability(작성력) :** 언어가 쉽게 사용될 수 있는 정도
  - 단순성과 직교성
  - 추상화 지원 (Support for abstraction)
    - 추상화(abstraction) : 많은 세부 사항이 무시될 수 있는 방식으로 복잡한 자료 구조나 연산을 정의하여 사용할 수 있는 능력 즉, 여러 가지 사물이나 개념에서 공통되는 특성이나 속성 따위를 추출하는 것
    - 추상화 종류 : 프로세스 추상화와 자료 추상화
      - 프로세스 추상화(process abstraction) : 프로그래밍 언어에서 가장 오래된 개념 중의 하나
        - 프로그램에서 여러 번 요구하는 정렬 알고리즘을 구현하기 위한 부프로그램의 사용. 부프로그램들은 프로세스가 어떻게 실행하는지에 관해서는 상세하게 제공하지 않고 어떤 프로세스가 실행되어야 한다고 다른 프로그램에게 제공하기 때문에 모든 부프로그램은 프로세스 추상화이다.
      - 자료 추상화(data abstraction) : 복잡하고, 어렵고, 귀찮은 자료들을 간단한 변수 하나로 표현하는 것.
        - 배열(array)이나 구조화된 변수(structured variable)가 자료를 추상화한 예이다. 즉 여러 개의 변수를 하나의 변수에 넣어 캡슐화 함으로서 간단한 변수 하나로 표현.
      - 프로세스 추상화와 자료 추상화를 합하면 class로 abstract data type 이다.

1.3 언어 평가기준

- 표현력(expressivity)
  - ✓ 언어로 표현할 때 상대적으로 편리한 방법
  - ✓ 예) c 언어에서 count++는 count=count +1보다 편리한 방법

❖ **Reliability(신뢰도)** : 프로그램이 모든 조건하에서 주어진 명령에 따라 수행되느냐?

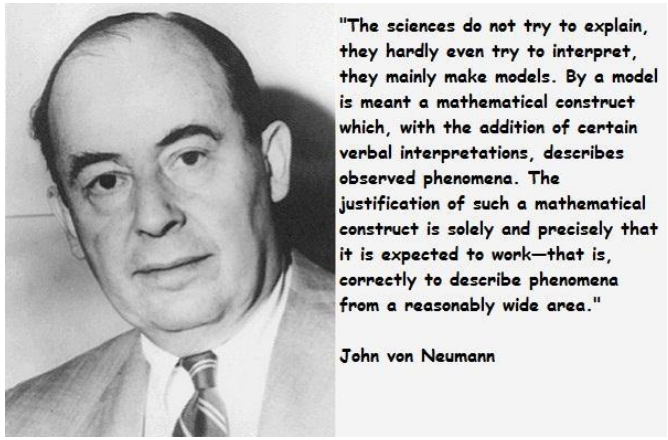
- 타입 검사(Type checking) – 컴파일 시간이나 실행 시간에 타입 오류를 검사
  - ✓ 타입 검사는 언어 신뢰성에 중요한 요소
  - ✓ 실행 시간 타입 검사는 비용이 비싸므로 컴파일 시간 타입 검사가 더 바람직.
  - ✓ 2+3.5 설명
  - ✓ 6장에서 자세하게 설명
- 예외 처리(exception handling)
  - ✓ 일반적으로 프로그램이 **처리**되는 동안 특정한 문제가 일어났을 때 **처리**를 중단하고 다른 **처리**를 하는 것.
  - ✓ 0으로 나누는 연산
- 별칭(aliasing)
  - ✓ 두 개 이상의 변수들이 하나의 기억 장소를 공유하는 것
  - ✓ 신뢰성을 저해하는 요소이므로 신뢰성을 높이기 위해서는 별칭을 제한

1.3 언어 평가기준

- **비용(Cost) :** 극한적인 상황의 전체 cost
  - 프로그래밍 언어의 최종 비용
    - ✓ 언어를 사용하는 프로그래머를 교육시키는 비용
    - ✓ 언어를 사용하여 프로그램을 작성하는 비용
    - ✓ 언어로 작성된 프로그램을 컴파일하는 비용이다.
    - ✓ 프로그램을 실행시키는 비용
    - ✓ 언어 구현 시스템의 비용
    - ✓ 신뢰성 부족에 따른 비용
    - ✓ 프로그램을 유지보수하는 비용
  - 비용에 대해 가장 중요한 요소
    - ✓ 프로그램 개발, 유지보수, 신뢰성 ➔ 이들은 작성력과 가독성에 의해 결정

1.4 언어 설계에 영향을 주는 요인들

- **프로그래밍 언어의 기본 설계에 영향을 주는 요소 –**
  - **컴퓨터 구조와 프로그래밍 설계 방법론**
- **컴퓨터 구조 :**
  - 폰 노이만 구조(그림 1.1(20페이지))
  - Stored program 방식
  - **폰 노이만 구조**(Von Neumann
  - architecture)는 현재와 같은 CPU,
  - 메모리, 프로그램 구조를 갖는 범용
  - 컴퓨터 구조의 확립이다.

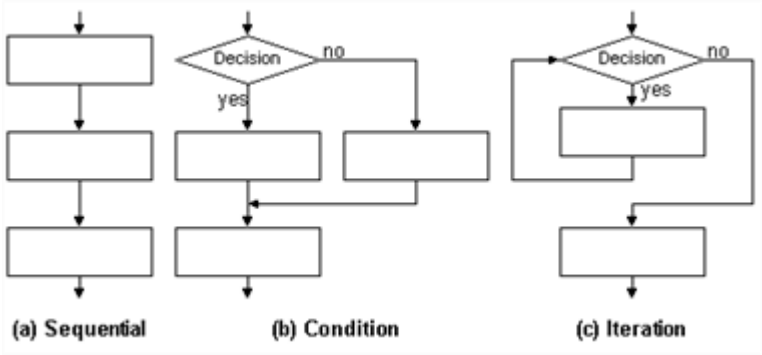


  - **존 폰 노이만**(1903년 - 1957년)은 헝가리 출신 미국인 수학자이다. 양자 역학, 함수 해석학, 집합론, 위상수학, 컴퓨터 과학, 수치해석, 경제학, 통계학 등 여러 학문 분야에 걸쳐 다양한 업적을 남겼다.
  - 폰 노이만은 맨해튼 프로젝트에 참여할 당시 발표한 논문 <전자계산기의 이론 설계 서론>에서 CPU, 메모리, 프로그램 구조를 갖는 프로그램 내장 방식 컴퓨터의 아이디어를 처음 제시하였고, 7년 후 케임브리지 대학교의 의뢰로 세계 최초의 프로그램 내장 방식 컴퓨터 EDSAC을 제작한다. 이후에 나온 컴퓨터는 모두 폰 노이만의 설계를 기본 구조로 디자인되고 있다.

1.4 언어 설계에 영향을 주는 요인들

- **컴퓨터 구조 :**
  - 명령형 언어(imperative languages) : 폰 노이만 구조에 기반한 언어
    - ✓ 기억 장소를 표시하는 변수 사용
    - ✓ 변수의 값을 변경시키기 위한 배정문 사용
    - ✓ 반복문
    - ✓ 명령어를 순차적으로 실행
  - 함수 언어(functional languages)
    - ✓ 명령형 언어에서의 변수, 배정문, 반복문을 사용하지 않고서 함수형 언어로 프로그래밍 가능
    - ✓ 특정 컴퓨터 구조에 독립
    - ✓ 수학적 함수에 기반한 인터프리터 언어로 함수에 매개변수를 적용하여 사용
- **프로그래밍 설계 방법론 :**
  - 60년대: **프로세스 지향 설계 방법론(process-oriented)**
    - ✓ 구조화 프로그래밍(structured programming) – **다익스트라**(Dijkstra)
      - ♣ 절차적 프로그래밍의 하위 개념.
      - ♣ GOTO문을 되도록이면 사용하지 말자.
      - ♣ 프로그래밍은 순차, 선택, 반복 등 3가지 구조만 있으면 가능하다.

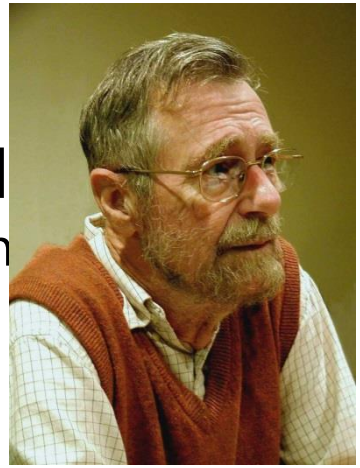
1.4 언어 설계에 영향을 주는 요인들



- ♣ 순차는 구문 순서에 따라서 순서대로 수행된다는 것이다.
- ♣ 선택은 프로그램의 상태에 따라서 여러 구문들 중에서 하나를 수행하는 것이다. 주로 if..then..else..endif, switch, case와 같은 키워드로 표현한다.
- ♣ 반복은 프로그램이 각각의 원소들에 대해 어떤 구문을 반복 수행하는 것이다. 보통 while, repeat, for, do..until 같은 키워드로 표현한다.

- ✓ 하향식 설계(top-down design), 단계적 세분화(step-wise refinement)
- ✓ 타입 검사의 불완전성과 제어문의 부적절성 때문에 goto 문을 과도하게 사용하도록 요구.

1.4 언어 설계에 영향을 주는 요인들



- **다익스트라**(Edsger W. Dijkstra,
- 1930~2002)는 네덜란드의 전산학자이다. 29살이던
- 1959년 'Numerische Mathematik' 이라는 학술지에
- 'A note on two problems in connection with graph
- '라는 3페이지 짜리 논문을 발표하였는데 **지금까지**
- **9600번 이상 인용**되면서 컴퓨터과학 역사상 가장
- 유명한 논문 중의 하나가 되었다.
- 1000회 이상 인용되는 논문도 거의 없는 상황.
- 이 논문이 최단 거리 논문
- 그래서 1972년 컴퓨터 분야의 노벨상인 튜링상 수상

1.4 언어 설계에 영향을 주는 요인들

- **70년대 후반: 자료 지향 설계 방법론(data-oriented)**
  - ✓ 프로세스 지향 프로그램 설계방법론에서 자료 지향 프로그램 설계 방법론(추상 자료 타입의 사용에 중점)
  - ✓ 추상 자료 타입 사용을 통한 자료 지향 문제 해결 방법
  - ✓ 자료 추상화 지원
  - ✓ 최초 언어: SIMULA 67
- **80년대: 객체지향 설계 방법론(object-oriented)**
  - ✓ 객체지향 설계(자료 추상화에서 시작, 자료 추상화는 객체를 캡슐화(encapsulation), 자료에 대한 접근을 은폐, 상속(inheritance)과 동적 타입 바인딩(dynamic type binding)을 추가
  - ✓ 자료 추상화(캡슐화, 자료 은폐), 상속, 동적 바인딩
  - ✓ Smalltalk, C++, Java, Ada95, C#

1.5 언어 부류 (language categories)

- 프로그래밍 언어 분류 – 명령형, 함수, 논리, 객체지향 언어
  - **명령형 언어**(imperative language)
    - 폰 노이만 구조를 기반으로 하는 언어
    - 특징 – 변수, 배정문, 반복, 순차 수행
    - 예 – C, Pascal
    - 비주얼 언어는 명령형 언어의 sub-category
    - 스크립트 언어는 한 부류가 되지 않는다 – 스크립트 언어인 Perl, JavaScript, Ruby 등은 모두 명령형 언어
    - C 언어의 예

```
int main(void)
{
  int x, y;
  x = 10;
  y = x +20;
  return 0;
}
```

  - **함수 언어(functional language)**
    - 수학 함수와 같은 원리로 프로그램 구성
    - 명령형 언어의 변수나 배정문을 사용하지 않고 프로그래밍
    - Lisp, Scheme

1.5 언어 부류 (language categories)

- ✓ Lisp의 예
  - ♣ (first (sort '(3 9 7 5)))

➢ **논리 언어(logic language)**

- ✓ 규칙-기반 언어(rule-based language)
- ✓ 개체에 대한 사실과 개체 사이의 관계 규칙을 이용해 원하는 결과 얻음
- ✓ 'how to do'보다는 'what to do'를 기술
  - ♣ 규칙이 특정 순서 없이 기술
  - ♣ 언어 시스템이 요구된 결과를 생성하는 실행 순서 선택
- ✓ Prolog

➢ **객체지향 언어(object-oriented language)**

- ✓ 명령형 언어 + 객체지향 개념
- ✓ 자료 추상화, 상속, 동적 binding
- ✓ Java, C++

➢ **마크업(markup) 언어**

- ✓ 프로그래밍 언어가 아니다.
- ✓ 계산을 명세하지 않지만, 웹 문서에서 정보에 대한 일반적인 형태를 기술
- ✓ XHTML, XML

1.5 언어 부류 (language categories)

- **비주얼 언어(visual language)**
  - ✓ 명령형 언어의 한 부속 부류
  - ✓ Drag-and-drop 기능
  - ✓ 그래픽 사용자 인터페이스를 생성하는 간단한 방법 제공
  - ✓ Visual BASIC, VB.NET, Visual C++ 등
- **특정 목적 언어(special-purpose languages)**
  - ✓ RPG(report Program Generator) – 사무보고서 생성
  - ✓ GPSS(General Purpose Simulation System)

1.6 언어 설계에 대한 trade-off

- 두 가지 평가 기준의 trade-off
  - 신뢰성과 실행 비용 (Reliability versus cost of execution)
    - ✓ Java에서는 인덱싱에 대한 checking을 위해서 배열 원소들에 대한 모든 reference들을 요구한다. 반면에 c 언어는 첨자 범위에 대한 검사를 요구하지 않는다. 그래서 Java 언어는 신뢰성이 증가하지만, c 언어는 Java 언어보다 더 빠르게 실행한다. 또한, Java 언어는 실행 비용은 증가한다.
  - 작성력과 가독성 (Writability versus readability)
    - ✓ APL은 compact 한 program으로 작성하기 위해서 복잡한 계산을 허용하는 많은 강력한 연산자들을 제공한다. 그래서 작성력은 매우 좋지만 가독성은 떨어짐. 즉, 함축되고 간명한 표현식은 수학적으로 아름답지만 프로그래머 이외의 다른 사람이 이해하기는 어렵다.
  - 작성력과 신뢰성
    - ✓ C++의 포인터는 매우 융통성 있는 자료 구조이지만 신뢰성에 문제가 있어서 Java에는 포함되지 않았다.

1.7 구현 방법

- Layered **View of Computer** : 그림 1.2(26페이지)
- **구현 방법** : 컴파일 기법, 해석 기법, 하이브리드 기법

- **컴파일(compile) 기법** :
  - 고급 언어로 작성된 프로그램을 컴퓨터가 바로 실행할 수 있는 기계어 프로그램으로 번역하는 방법
  - 컴파일하는 프로그램 : 컴파일러(compiler)
  - 매우 빠르게 실행 가능, 구현이 어렵고, 메모리 많이 사용
  - 번역은 어휘 분석, 구문분석,의미 분석, 코드 생성 등 여러 단계를 거침
  - Fortran, Pascal, C, COBOL, C++, Ada 등.
  - 그림 1.3(27페이지) ➔ 다음 페이지
  - Linking과 Loading

1.7 구현 방법

- 컴파일러의 구조에 대한 견해 :
  - 전단부(Front-end)-후단부(Back-end)

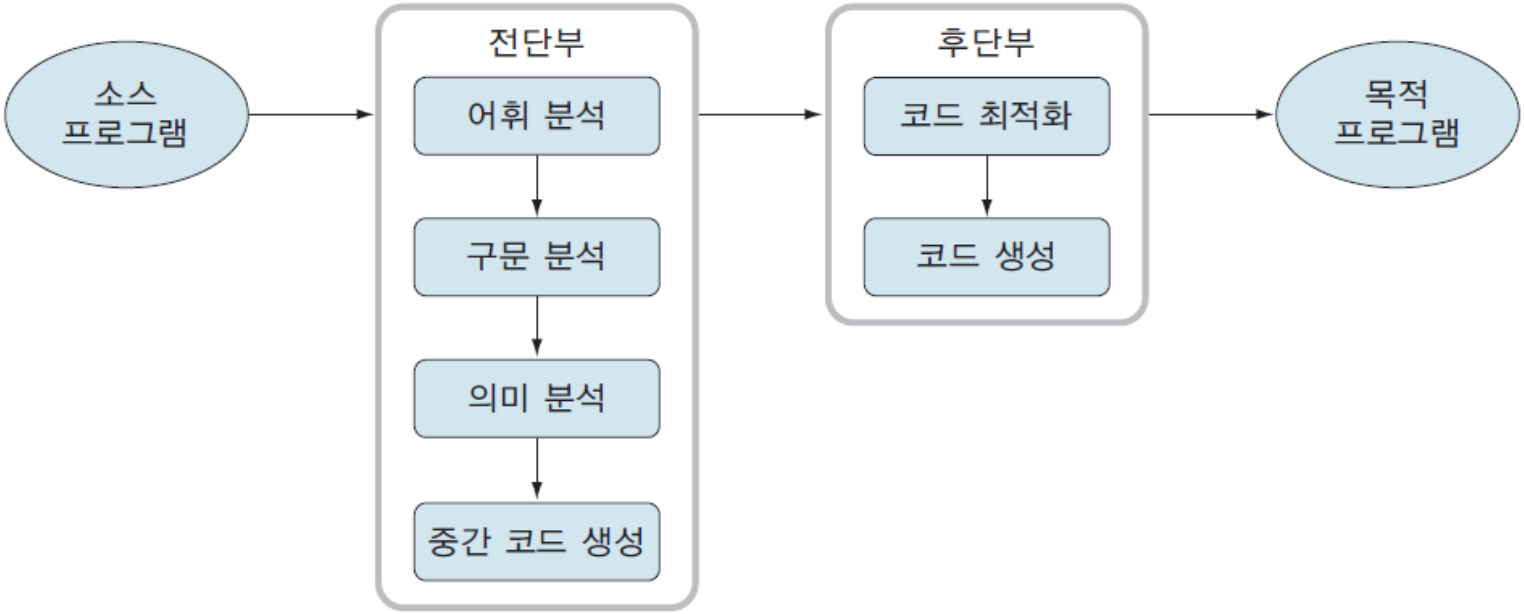


그림 2-2 컴파일러의 구체적 구조

1.7 구현 방법

❖

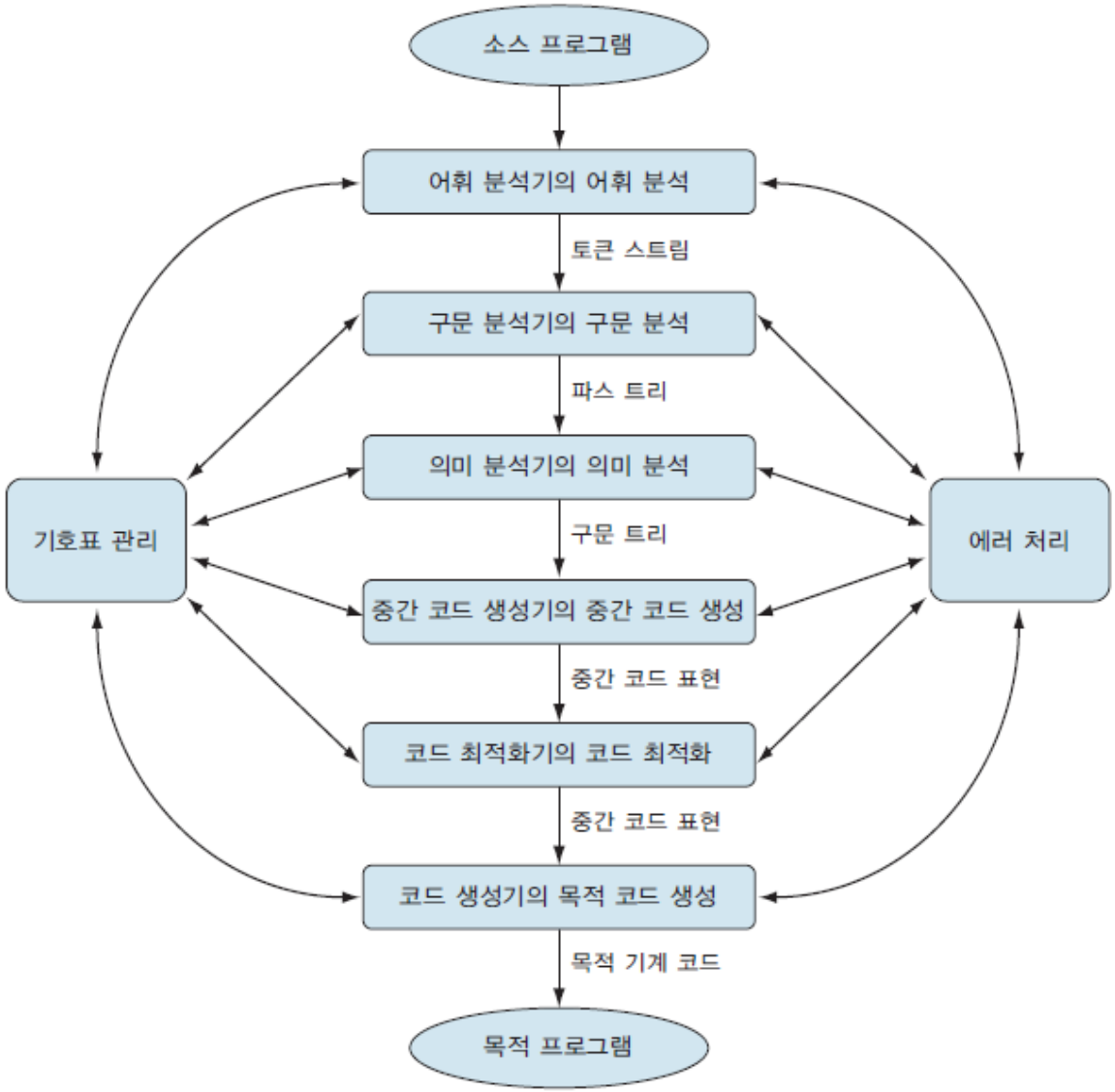
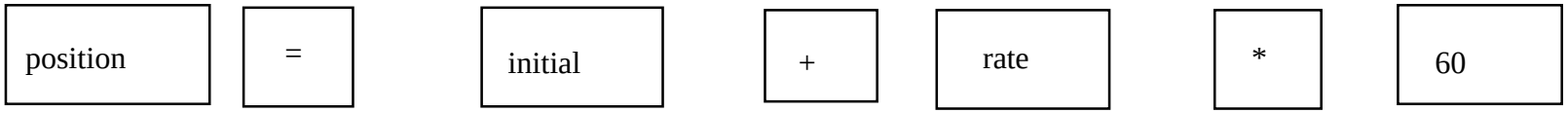


그림 2-3 컴파일러의 논리적 구조

1.7 구현 방법

- **어휘 분석(lexical analysis, scanning)**
  - 원시 프로그램을 읽어 들여 토큰(token)이라는 의미 있는 문법적 단위(syntactic entity)로 분리
  - 어휘 분석을 담당하는 도구(tool)를 어휘 분석기(lexical analyzer) 혹은 스캐너(scanner)라고 부른다.
  - 토큰 : 문법적으로 의미 있는 최소 단위로 문법에서 터미널 기호



■ 토큰의 종류

- **식별자(identifier)** ..... **(position, initial, rate, ...)**
- **예약어(key word)** ..... **(IF, WHILE, ...) : reserved word**
- **상수(constant)** ..... **(60, " KIM " ....) : numerical, literal**
- **연산자(operator)** ..... **(+, *, %, ...)**
- **구분자(delimiter)** ..... **([,], ; ....)**

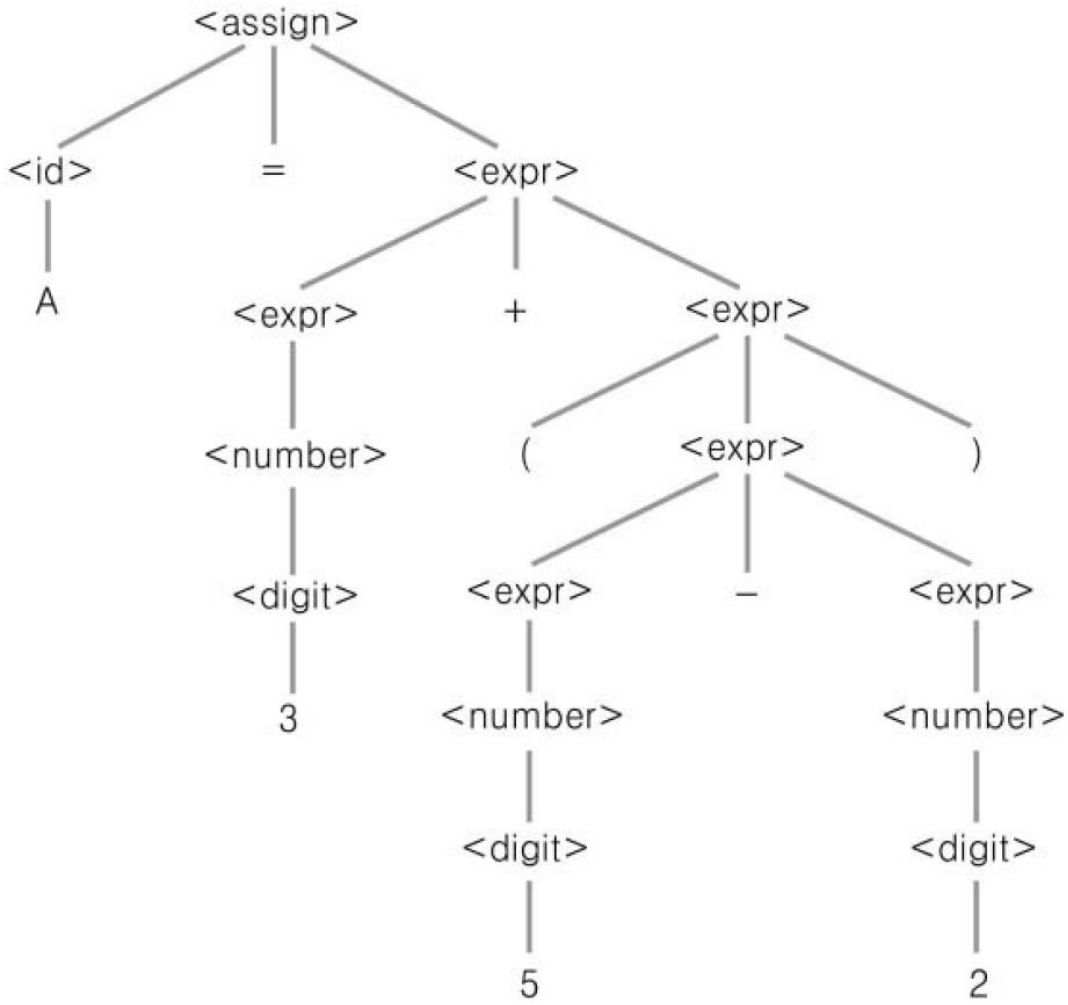
1.7 구현 방법

❖ 구문 분석(syntax-analysis, parsing)

- 어휘 분석 단계로부터 토큰들을 받아 이 토큰들이 주어진 문법(grammar)에 맞는지를 검사하고 올바른 문장에 대해서는 그 문장에 대한 파스 트리(parse tree)를 출력하고 올바르지 못한 문장에 대해서는 에러 메시지(error message)를 출력하는 과정을 구문 분석(syntax analysis) 혹은 파싱(parsing)
- 구문분석을 담당하는 도구(tool)를 구문 분석기(syntax analyzer) 혹은 파서(parser)라고 부른다.
- 파스 트리(parse tree)는 토큰들을 터미널 노드(terminal node)로 하는 트리(tree)

1.7 구현 방법

- A = 3 + (5 - 2)에 대한 파스 트리



1.7 구현 방법

- **의미 분석(semantic-analysis)**
  - 원시 프로그램의 의미적 오류를 검사하고 계속되는 코드 생성 단계를 위한 정보를 모으는 일
  - 의미 분석을 담당하는 도구(tool)를 의미 분석기(semantic analyzer)
  - 의미 분석 단계에서 가장 중요한 기능 중의 하나는 형 검사(type checking)
    - ✓ 각 연산자들이 원시 프로그램 규칙에 의해서 허용된 피연산자를 가졌는가를 검사(일반적 연산(generic operation)과 형 고정 연산(Type specific operation))
    - ✓ 정수와 실수의 일반적 연산을 허용하는데 이때의 의미 분석기는 연산을 수행하기 전에 정수를 실수로 바꾸어 주는 작업.(언어정의시간)
    - ✓ 예를 들어 a+b와 같은 산술식을 생각해보자.
      - a와 b의 자료형을 각각 int형과 float형이라면 형 고정 연산에서는 에러. 반면에 일반적 연산에서는 a나 b의 형을 변환하여 연산을 허용. 형 변환(type conversion)이란 주어진 자료형의 값을 다른 자료형의 값으로 변환하는 것을 의미, 이에는 시스템에서 자동으로 형을 변환하는 묵시적 형 변환(implicit type conversion)과 프로그래머가 명시적으로 형 변환을 요구하는 명시적 형 변환(explicit type conversion). 명시적 변환은 프로그래머가 명령문으로 요구한 형 변환으로 캐스트(cast) 연산. 묵시적 형 변환은 시스템에서 자동으로 형을 변환하는 것으로 자동 변환(automatic type conversion) 또는 강제 변환(coercion

1.7 구현 방법

- **중간코드생성(intermediate code generation)**
  - 구문분석 단계에서 만들어진 구문 트리를 이용하여 코드를 생성하거나 한 문법 규칙이 감축될 때마다 문법지시적 번역(syntax-directed translation)으로 이루어짐
  - 중간 코드 생성을 담당하는 도구를 중간 코드 생성기(intermediate code generator)

- **코드 최적화(code optimization)**
  - 같은 기능을 유지 하면서 코드를 좀 더 효율적으로 만들어 코드 수행 시 기억 공간 이나 수행 시간을 절약하기 위한 단계

- **목적 코드 생성(code generation)**
  - 연산을 수행할 레지스터를 선택하거나 자료에 기억 장소의 위치를 정해주며 실제로 목적 기계에 대한 코드를 생성하는 단계

1.7 구현 방법

- **순수 해석** : 프로그램이 번역 과정 없이 해석되면서 실행하는 기법으로 인터프리터 기법이라고 부른다.
  - 해석하는 프로그램 : 인터프리터(interpreter)
  - 쉽게 구현 할 수 있다
  - 언어에 대한 가상 기계 제공
  - 문장 해석에 따른 실행 속도가 느리다, 메모리 save.
  - 단순한 구조의 언어에 적합
    - ✓ APL, LISP, UNIX's Shell script, JavaScript 등
    - ✓ 그림 1.4(30페이지) ➔ 다음 페이지

1.7 구현 방법

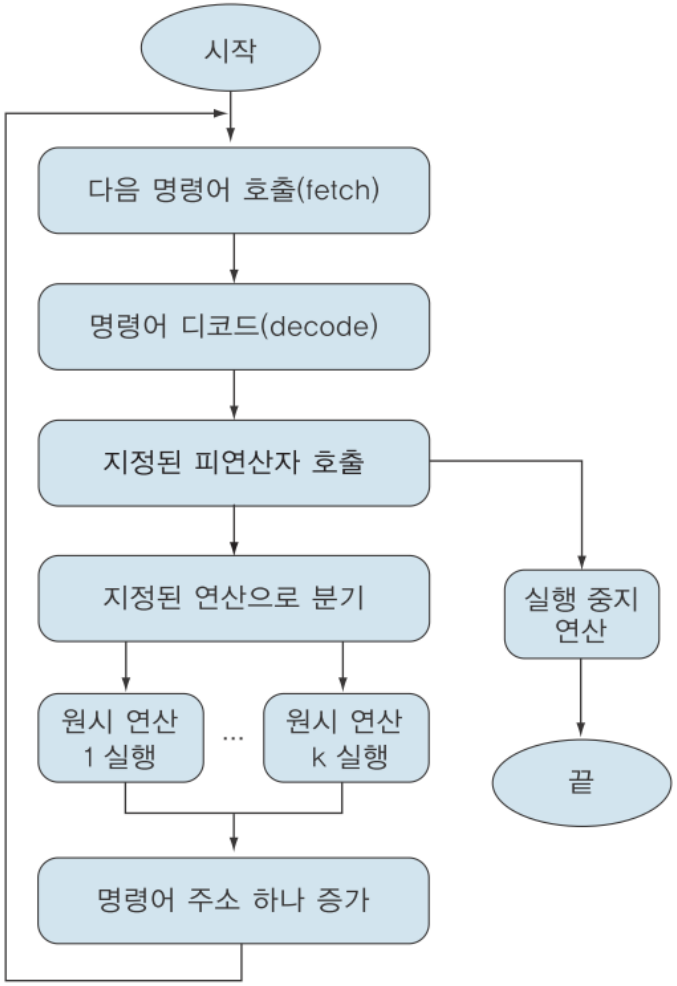


그림 1-9 인터프리터의 처리 과정

1.7 구현 방법

- **하이브리드 기법** :
  - 컴파일러와 인터프리터를 같이 사용하는 기법
  - 고급 언어로 작성된 프로그램을 쉽게 해석할 수 있도록 중간 코드 형태로 번역하고, 그 뒤에 번역된 중간 코드 형태의 프로그램을 해석하여 실행하는 기법
  - 원시 프로그램이 한번만 번역되기 때문에 인터프리터 기법보다 빠르다. 인터프리터 기법보다 융통성이 없다.
  - Java, Perl 등
  - 그림 1.5(31페이지)
  - **예** : Java
    - ✓ 바이트 코드(byte code)라고 하는 중간 코드로 번역
    - ✓ 운영체제마다 별도로 존재하는 자바 가상 기계(Java Virtual Machine)가 바이트 코드를 실행

- **프리프로세서 :** #include, #define, macro

1.8 프로그래밍 환경

- 통합 환경
- 소프트웨어를 개발하는데 사용되는 도구들의 집합
  - 파일시스템, 텍스트 편집기, 링커, 컴파일러, 디버거 등
  - 단일의 사용자인터페이스를 갖는 통합된 도구
  - Borland C++, Smalltalk, MS Visual C++, Visual Basic, Delphi, Borland사의 Jbuilder(java에 대한 통합 개발 환경), Visual Studio.NET(더 크고, 복잡한 visual 환경으로 C#, visual basic.net, jscript, j#, c++등 사용 가능)

Report 2

- 1장 복습 문제, 연습문제 풀이(34-36 페이지)
  - 복습 문제 2, 14, 15, 25
  - 연습문제 10번 풀어서 제출하기
  - 기간 : 1주