1. Introduction to Compiler

2022학년도 2학기

컴파일러

강의 소개

- **CB23606: 컴파일러(Compiler)**
- **Course Home Page: PLATO - https://plato.pusan.ac.kr**
- **Classroom: 201-6516**

- **Instructor: 권 동 현**
  - e-mail: kwon at pusan dot ac dot kr
  - office: 313동 411호

- **강의 목표**
  - 전반적인 컴파일 과정에 대한 이해
  - 컴파일러가 사용하는 자료구조와 최적화 기법 학습
  - 간단한 컴파일러 구현

교재 소개

- **주교재**
  - 도서명 : 컴파일러의 이해(증보판)
  - 저자 : 박두순
  - 출판사 : 한빛아카데미㈜
  - 참조 : 강의 자료

- **보조 교재**
  - 만들면서 배우는 컴파일러 첫걸음
  - 저자 : 나카다 이쿠오
  - 출판사: 인사이트
  - 참조: 실습 자료

주차별 강의 내용

- 1주차: Introduction to compiler
- 2주차: Compilter structure
- 3주차: Formal language and finite automata
- 4주차: Lexical analysis
  - 실습 1
- 5주차: Context-free grammar and pushdown automata
- 6주차: Syntax analysis (1)
- 7주차: Syntax analysis (2)
  - 실습 2
- 8주차: 중간고사
- 9주차: Semantic analysis
  - 실습 3
- 10주차: Intermediate representation
- 11주차: Runtime-environment
- 12주차: Code generation
  - 실습 4
- 13주차: Code optimization (1)
- 14주차: Code optimization (2)
- 15주차: 기말고사

평가 방법

- **Attendance: 10%**
  - 결석 1회당 1점 감점
  - 3회 지각 = 결석 1회
  - 1/3 초과 결석시 F
- **Assignments: 20%**
- **Mid-term exam: 35%**
- **Final exam: 35%**

목차

- 컴파일러의 필요성
- 프로그래밍 언어
- 번역기의 종류

1.1 컴파일러의 필요성

- **컴파일러 필요성**

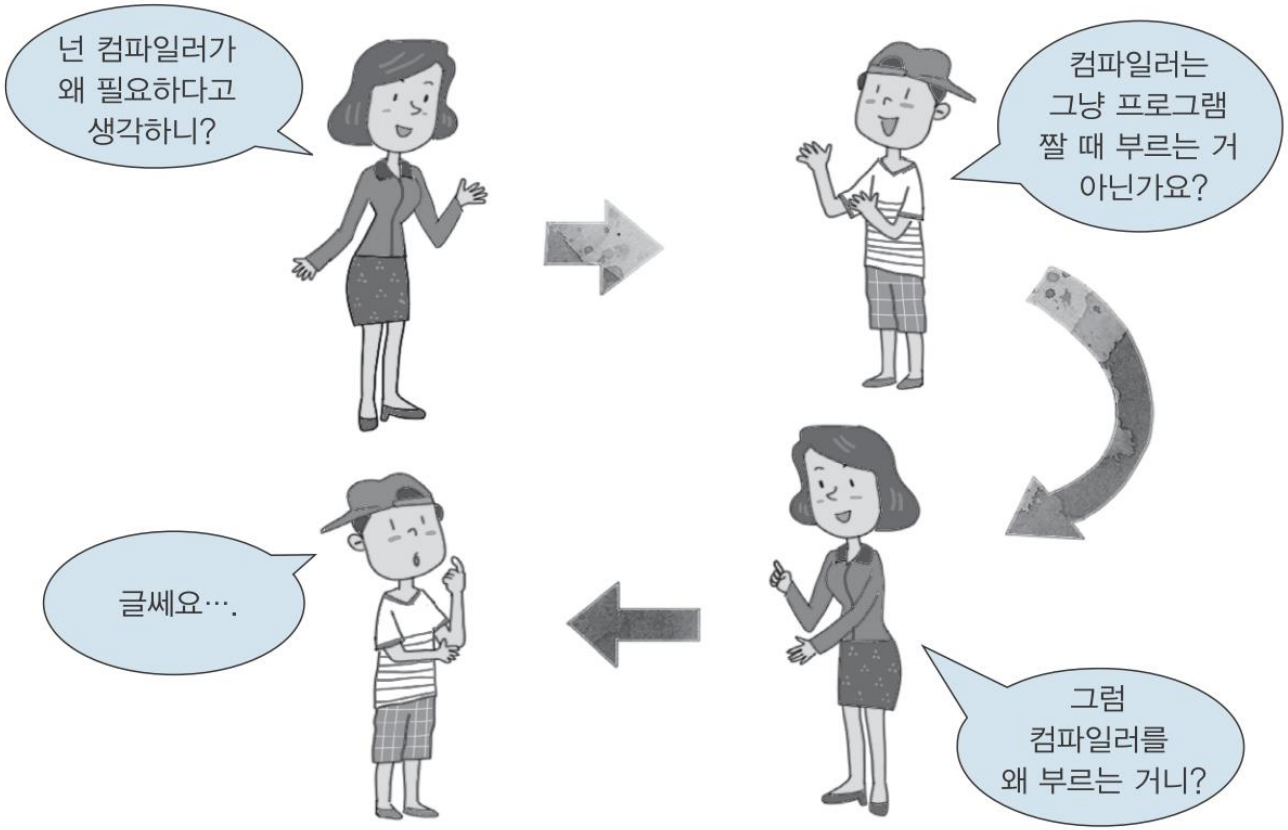


그림 1-1 컴파일러의 필요성

1.1 컴파일러의 필요성

- **언어란?**
  - 의사전달을 하기 위한 도구

- **언어의 종류**
  - 자연언어(natural language)
  - 형식언어(formal language)

- **형식언어의 종류**

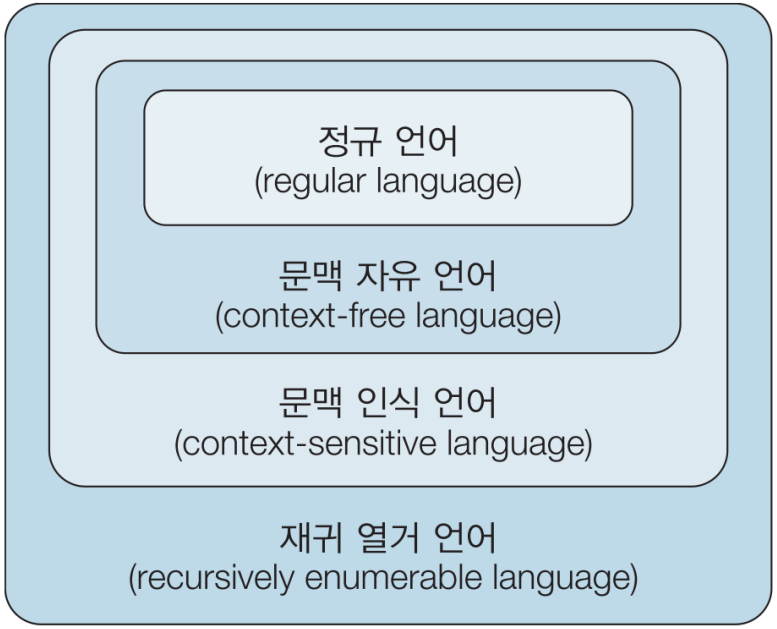


그림 1-2 형식 언어의 촘스키 계층 구조

1.1 컴파일러의 필요성

- **프로그래밍 언어**
  - 컴퓨터 언어 중에서 어떤 문제를 풀기 위해서 일련의 과정을 기술하는데 사용되는 언어
  - **기계어** - 프로그래밍 언어 중 에서 초기에 사용된 언어로 0과 1로 구성된 언어.
    - 기계어를 주로 사용한 이유는 컴퓨터가 0과 1로 만들어진 추상 기계(abstract machine)이기 때문
    - 그러나 기계어로 프로그래밍을 하기란 상당히 어렵고 매우 복잡하다는 단점.
  - **어셈블리어** – 기계어의 단점을 보완하기 위해 나온 언어가 어셈블리어.
    - 어셈블리어는 0과 1로 구성된 기계어 대신 더하기에 ADD, 빼기에 SUBT 등 대응하는 명령 기호를 사용함으로써 프로그래밍 작업에서 기계어의 단점을 조금 보완.
    - 인간이 어셈블리어를 사용하니 컴퓨터가 이것을 이해하지 못했다. 서로 이해할 수 있는 언어가 달랐기 때문이다.
  - 이를 쉽게 이해하기 위해 자연 언어에 빗대어 살펴보자.
    - 예를 들어 프랑스어를 모르는 한국 인과 한국어를 모르는 프랑스인이 만나 인사를 한다고 하자. 서로 말을 알아듣지 못하니 통 역사가 필요한데 이러한 통역사는 [그림 1-3]과 같이 번역기(translator)의 역할을 수행한다.
    - 어셈블리어를 기계어로 번역해주는 번역기는 어셈블러(assembler)

1.1 컴파일러의 필요성

그림 1-3 번역기의 필요성

- 고급언어
  - 어셈블리어도 저급 언어의 수준을 벗어나지는 못했으며, 그 후 저급 언어의 단점을 보완하기 위해 C, 파스칼, 알골, 포트란, 코볼 등 사람 중심의 고급 언어가 탄생
  - 고급 언어도 저급 언어와 마찬가지로 기계어로 변환해주는 번역기가 필요한데 이를 컴파일러

1.1 컴파일러의 필요성

- **컴파일러가 필요한 이유(정리)**
  - 인간은 문제를 해결하기 위해 컴퓨터를 사용하며 컴퓨터와 의사소통을 하는 데 '언어'가 필요하다.
  - 컴퓨터는 기계어를 사용하지만 인간이 기계어를 사용하여 문제를 표현하기란 무척 어렵기 때문에 인간은 사람 중심 언어인 고급 언어를 사용한다.
  - 그런데 인간이 사용하는 고급 언어는 컴퓨터가 이해하지 못한다.
  - 따라서 인간이 사용하는 고급 언어를 기계어로 변환해주는 번역기인 컴파일러가 필요하다

1.2 프로그래밍 언어

- **고급 언어의 장점**
  - 특정한 컴퓨터의 구조에 대한 지식이 없어도 프로그래머의 생각을 자연스럽게 표현할 수 있도록 만들어져 있다
  - 저급 언어에 비해 배우기가 쉽고, 프로그래머의 생산성(productivity)을 높임.
  - 이식성(portability) 우수 : 한 기종에서 다른 기종으로 프로그램을 쉽게 가져감.
  - 디버깅이 용이함.
  - 기계독립적 – 특정한 컴퓨터의 구조를 모르고도 프로그래밍을 할 수 있음.

- **고급 언어의 분류 방법**
  - 특성별/연대별
  - 절차 언어(procedural language)와 비절차 언어(non-procedural language)
  - 블록 언어(block language)와 비블록 언어(non-block language)
  - 절차 언어는 명령형 언어라고도 하며 다음과 같은 특징이 있음
    - 명령의 순차적 실행, 기억장소를 표시하기 위한 변수 사용, 값을 저장하기 위한 치환문 사용, 반복 연산을 구현하는 반복문 사용

1.2 프로그래밍 언어

- **식별자(Identifier)와 변수(variable)**
- **블록(Block) 구조**
  - 프로그램을 여러 단계의 블록으로 나누어 작성할 수 있도록 해주는 언어 구조
- **재귀법(Recursion)**
  - 수학적 귀납법이라고도 불리는 재귀법은 자신을 정의하거나 자기 자신을 재참조하는 방법
- **BNF**
  - BNF(Backus-Naur Form)는 문법을 표현하는 방법

1.2 프로그래밍 언어

- **기계독립적(machine independent)**
  - 특정한 기계에 국한하지 않고 실행할 수 있는 프로그램으로 작성하는 것.
- **문서화(documentation)**
  - 사람들을 이해시키기 위해 문장, 도식 등을 사용하여 문서화하는 것으로, 프로그램을 작성할 때 설계, 제조 공정, 작업 결과를 나타내어 유지·보수를 위한 정보를 주는 것이다.
- **레코드(record)구조**
  - 구조형(structured type)의 대표적인 형으로는 배열(array)과 레코드가 있다. 배열은 첫 번째 원소의 상대적 위치인 첨자로 원소를 식별하는 동질성 자료(homogeneous data)의 집합이고, 레코드는 원소를 식별자로 구별하는 이질형 자료(heterogeneous data)의 집합이다.
- **함수언어**
  - 함수형 프로그래밍(functional programming)은 계산을 수학적 함수의 평가로 취급하는 프로그래밍 패러다임

1.2 프로그래밍 언어

- **폴란드식 표기법(polish notation -prefix형태)**
  - 폴란드의 얀 루카시에비치(Jan Lukasiewicz)가 1920년대에 제창한 논리식의 표기법으로 산술식에도 적용할 수 있다. 산술식을 표기할 때 피연산자를 연산자 뒤에 놓는 표기법
    - 예를 들면 산술식 a×(b + c)를 ×a + bc로 나타내는 것이다.
  - 폴란드식 표기법인 전위(prefix) 표기 방법, 역폴란드식 표기법(reverse Polish notation)이라 불리는 후위(postfix) 표기 방법, 그리고 중위(infix) 표기 방법이 있다.
- **예제: 이진 트리를 선형으로 표기하는 방법**
  - 전위 순회한 결과인 전위 표현은 +*ab/cd, 후위 순회한 결과인 후위 표현은 ab*cd/+, 중위 순회한 결과인 중위 표현은 a*b+c/d이다.

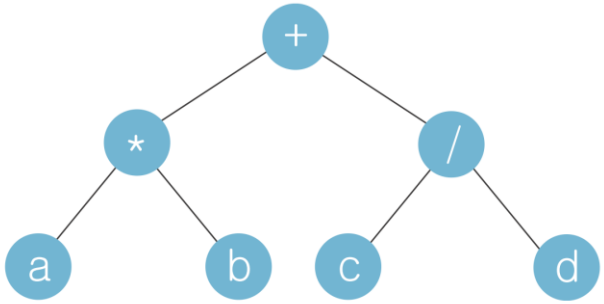


그림 1-4 이진트리

1.2 프로그래밍 언어

- **쓰레기 수거(garbage collection)**
  - 메모리 관리 기법 중의 하나로, 프로그램이 동적으로 할당했던 메모리 영역 중에서 필요 없게 된 영역을 해제하는 기능이다.
- **OOPL(Object-Oriented Programming Language)**
  - 객체지향 프로그래밍은 C, 파스칼과 같은 절차형 언어가 크고 복잡한 프로그램을 구축하기 어렵다는 문제점을 해결하기 위해 탄생 되었고, 시뮬레이션 언어인 시뮬라(Simula)에서 유래했다.
  - 객체(object)라는 작은 단위로 모든 처리를 기술하는 프로그래밍 방법으로, 이 방법으로 프로그램을 작성 하면 프로그램이 단순해지고 생산성과 신뢰성이 높은 시스템을 구축할 수 있다.

1.2 프로그래밍 언어

- **매개변수 전달**
  - 매개변수에는 함수 정의 구문에서 기술되는 매개변수인 **형식 매개변수**(formal parameter)와 함수를 호출할 때 기술되는 매개변수인 **실 매개변수**(actual parameter)가 있다. 실 매개변수를 기술할 때는 함수의 헤더에 정의된 자료형과 일치되게 해야 한다.
  - 형식 매개변수와 실 매개변수 사이에 값을 어떻게 주고받느냐에 따라 여러 가지 **매개변수 전달 방법**(parameter passing)이 있음
    - **참조 호출**(call by reference, call by address, call by location): 실 매개변수의 주소를 대응되는 형식 매개변수로 넘겨주는 것
    - **값 호출**(call by value): 실 매개변수와는 별도로 형식 매개변수에 대한 기억 장소를 별도로 할당하는 방법
    - **이름 호출**(call by name): 형식 매개변수의 이름이 사용될 때마다 그에 대응하는 실 매개변수 자체가 사용된 것처럼 매번 다시 계산하여 시행하는 방법

1.2 프로그래밍 언어

[예제 1-1] 참조 호출, 값 호출, 이름 호출

다음과 같은 알골 형태의 프로그램에서 참조 호출, 값 호출, 이름 호출을 한 경우 출력되는 값을 구하시오.

```
Begin Integer A, B;
  Procedure F(X, Y, Z); Integer X, Y, Z;
    Begin Y := Y + 2;
          Z := Z + X;
    End F;
  A := 3;
  B := 5;
  F(A+B, A, A);
  PRINT A
END
```

[풀이]

① 참조 호출의 경우 :

처음에 A = 3, B = 5이다. 프로시저 F가 호출되면 F(A+B, A, A)에서 addr(A+B) = addr(X), addr(A) = addr(Y), addr(A) = addr(Z)이다. Y := Y + 2에서 Y = 3 + 2 = 5, Z := Z + X에서 Z = 5 + 8 = 13 PRINT A에서 A의 주소는 Z의 주소와 같으므로 13을 출력한다.

② 값 호출의 경우 :

처음에 A = 3, B = 5이다. 프로시저 F가 호출되면 F(A+B, A, A)에서 X, Y, Z에 대한 기억 장 소를 별도로 할당한 다음 X = 8, Y = 3, Z = 3을 치환한다. Y := Y + 2에서 Y = 3 + 2 = 5, Z := Z + X에서 Z = 3 + 8 = 11 그런데 프로시저가 리턴되면서 X, Y, Z에 대한 기억 장소가 삭제된다. PRINT A에서 3을 출력한다.

③ 이름 호출의 경우 :

처음에 A = 3, B = 5이다. 프로시저 F가 호출되면 F(A+B, A, A)가 호출된 다음 Y := Y + 2에서 A = A + 2 = 3 + 2 = 5, Z := Z + X에서 A = A + (A + B) = 5 + 5 + 5 = 15 PRINT A에서 15를 출력한다.

1.3 번역기의 종류

- **번역기란 ?**
  - 하나의 프로그래밍 언어로 작성된 프로그램을 입력으로 하여 그와 동등한 의미를 갖는 다른 프로그래밍 언어로 된 프로그램을 출력하는 일종의 프로그램
  - 소스프로그램은 사용자가 작성한 프로그램이라 하고, 목적 프로그램은 소스 프로그램이 번역기에 의해서 번역된 프로그램.
  - 번역기의 종류와 기능

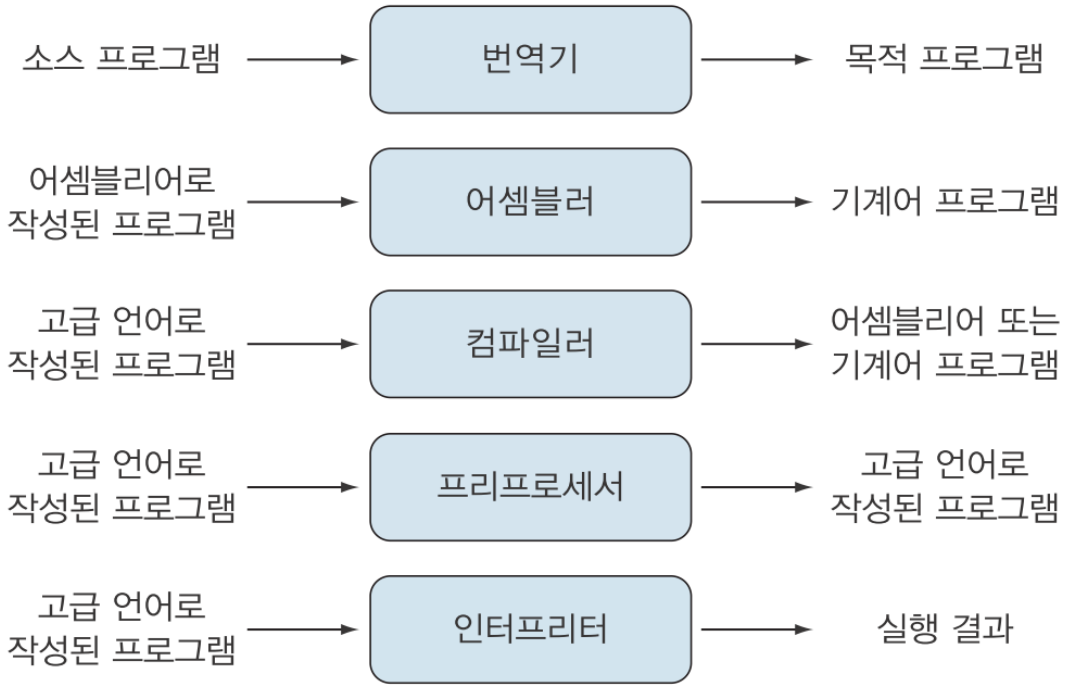


그림 1-5 번역기의 종류와 기능

1.3 번역기의 종류

- **어셈블러**
  - 어셈블리어로 작성된 프로그램을 그에 대응하는 기계어로 번역하는 번역기
  - 어셈블리 코드(assembly code)
    - 사람이 좀더 이해하기 쉽게 기호화한 것이다.
  - 어셈블리 명령어(instruction)의 전형적인 예

LOAD R_1, a (변수 a에 있는 값을 레지스터 R_1 에 적재)

ADD R_1, #2, R_2 (레지스터 R_1에 있는 값과 상수 2를 더해서 다시 레지스터 R_2에 저장
이때, R_1과 R_2가 같다면 ADD R_1, #2로 표시)

STORE b, R_1 (마지막으로 레지스터 R_1에 저장되어 있는 값을 변수 b에 저장)

  - 이렇게 해서 b = a+2를 계산

1.3 번역기의 종류

- 대부분 2-패스 어셈블러로 구성
  - 패스(pass) : 하나의 입력 파일(file)을 단 한번만 읽는 것으로 구성되는 단위로 하나의 패스는 일반적으로 여러 개의 단계(phase)들로 구성됨
  - 단계(phase) : 소스 프로그램을 하나의 표현에서 다른 표현으로 변환하는 것
  - 첫 번째 패스(first pass) : 어셈블리 코드를 가지고 기호표(symbol table) 작성

식별자 이름	주소
a	0
b	4

단, 하나의 단어(word)는 4 바이트로 구성되고, 각 식별자의 주소는 0 바이트로부터 시작한다고 가정

  - 두 번째 패스(second pass) : 연산(operation) 코드를 기계어로 표현하는 비트들의 sequence로 표시

*0001 01 00 00000000**

0011 01 10 00000010

*0010 01 00 00000100**

    - 첫 번째 4개의 비트는 명령어로서 0001은 LOAD, 0010은 STORE, 0011은 ADD를 나타낸다. 그 다음 두 개의 비트는 레지스터를 표시하는 것으로 여기서 01은 레지스터 1을 의미한다. 이어서 그 다음 두 개의 비트는 태그(tag)를 나타낸다. 00은 다음의 8비트가 메모리 주소를 가리키는 일반적인 주소 방식(ordinary address mode)이고 10은 다음의 8비트가 피연산자(operand)인 직접 번지 지정 방식(immediate address mode)일 때 사용된다.
    - *는 각각의 피연산자가 재배치(relocatable) 가능 기계어에서 재배치 비트임을 알려주는 것이다.

1.3 번역기의 종류

- **프리프로세서**(preprocessor, 전처리기)
  - 프로그래밍 언어에 유용한 기능들을 추가하여 언어를 확장시켜 주는 역할을 하는 것으로 원시언어와 목적언어가 모두 고급언어인 번역기

  - C 프리프로세서의 3가지 기능

    1) 헤더 파일이(header file)이 포함된 파일 포함(file inclusion) 기능
       - 프로그램에 헤더 파일(header file)들을 포함
       - 예) C 프리프로세서는 파일에 #include <stdio.h>라는 문장이 포함되어 있다면 그 자리에 표준 입출력과 관련된 함수 stdio.h의 내용으로 대체.

    2) 매크로(macro) 처리 기능
       - 매크로로 정의된 부분들에 대해서 필요할 때마다 확장
       - #define max 45=> max가 나타날 때마다 45로 바꾸어 줌

    3) 조건부 컴파일
       - 조건에 따라 소스 프로그램의 일부분을 선택적으로 삽입 혹은 삭제하는 가능

```
#if SYSTEM == WINDOWS
  #include "stdio.h"
#elif SYSTEM == UNIX
  #include "unix.h"
#else
  #include "etc.h"
#endif
```

1.3 번역기의 종류

- **인터프리터와 컴파일러**
  - 인터프리터와 컴파일러는 서로 상반 관계trade-off(두 변수가 서로 반대 방향으로 움직이는 것)이다.
  - 일반적으로 컴파일러 기법은 인터프리터 기법보다 실행 속도가 10배 이상 빠르다.
  - 인터프리터의 경우 고급 언어로 작성된 프로그램을 한 줄 단위로 번역과 실행을 하여 특히 반복문일 때 실행 시간이 많이 늘어나기 때문이다.
  - 반면에 한 줄 단위로 번역과 실행을 하여 매번 같은 기억 장소를 사용하므로 기억 장소를 줄일 수 있다는 것이 장점이다.

- **간단한 인터프리터 처리 과정**

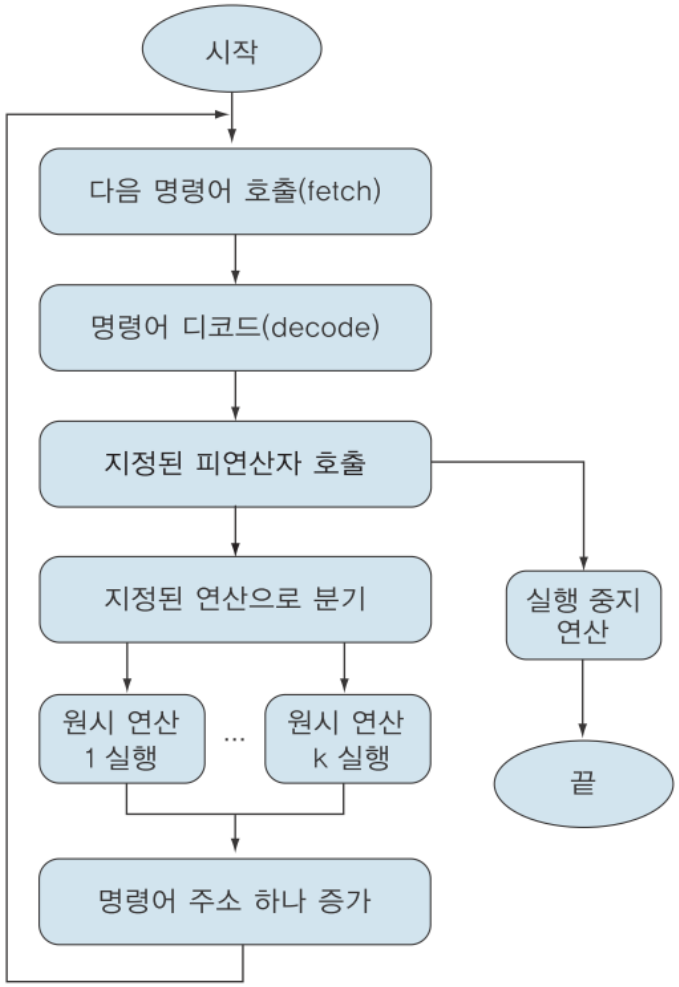


그림 1-9 인터프리터의 처리 과정

1.3 번역기의 종류

- **인터프리터와 컴파일러**
  - 반복문이나 계속 호출되는 부프로그램처럼 많은 횟수를 반복 처리하는 프로그램의 경우에는 컴파일러 기법이 큰 도움이 될 수 있다.
  - 인터프리터 기법에서는 반복 처리할 때마다 다시 디코딩해야 하지만, 컴파일러 기법에서는 전체적으로 한 번 디코딩하면 그 다음부터는 실행만 하므로 실행 시간 측면에서 효율적이기 때문이다.
  - 그러나 컴파일러 기법은 때로 몇 줄의 소스 프로그램이 몇 백 줄의 기계어로 번역되어 큰 기억 장소를 필요로 한다는 단점이 있다.

- **간단한 컴파일러 처리 과정**

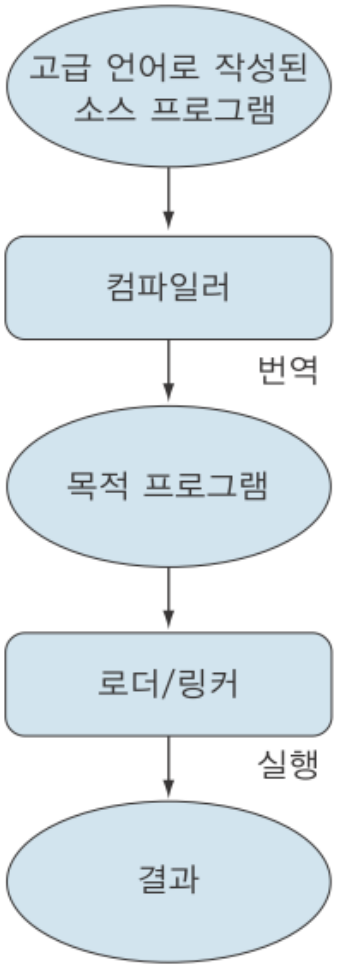


그림 1-8 컴파일러의 간단한 처리 과정

1.3 번역기의 종류

- **기타 번역기**
  - 크로스 컴파일러(cross compiler)
    - 원시 프로그램을 컴파일러가 수행되고 있는 컴퓨터의 기계어로 번역하는 것이 아니라 다른 컴퓨터의 기계어로 번역.
    - 예 - 목적 프로그램이 수행되어지는 컴퓨터의 용량이 커야 할 경우 용량이 작은 컴퓨터에서 크로스 컴파일러에 의하여 번역만 하고 번역된 목적 프로그램은 용량이 큰 컴퓨터에서 실행.
  - 실리콘 컴파일러(silicon compiler) :
    - 기존의 언어와 유사한 소스 언어를 가지고 있으나 이 언어에서 사용되는 변수는 기억 장소의 위치를 나타내지 않고 스위치 회로에서의 논리적 신호(0 혹은 1)나 논리 신호의 군(group)을 나타낸다. 또한 실리콘 컴파일러의 출력은 어떤 적당한 언어로 표현된 회로 설계도(circuit design)이다.
  - 바이트코드 컴파일러bytecode compiler :
    - 자바 언어가 대표적인 예이다. 바이트코드 컴파일러는 자바 소스 프로그램을 바이트코드라 불리는 중간 코드로 컴파일한다. 그런 다음 바이트코드는 자바 가상 기계에서 인터프리트되어 실행된다. 그래서 자바 언어 처리기를 컴파일러 방법과 인터프리터 방법을 결합한 혼합형 컴파일러라 하며, 이를 [그림 1-10]에 나타냈다

1.3 번역기의 종류

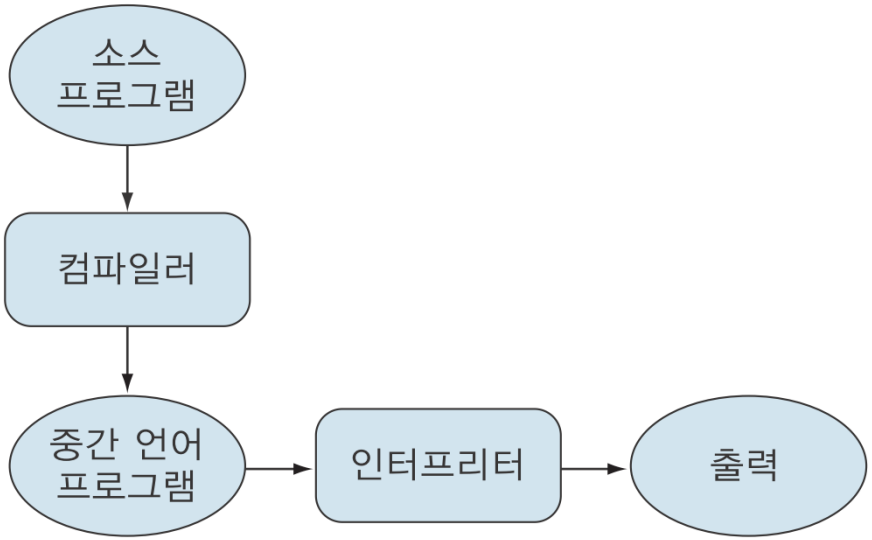
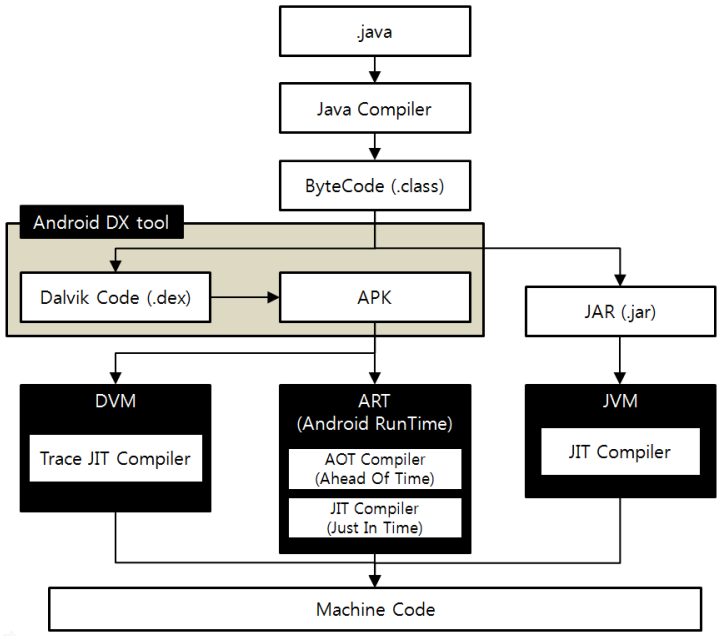


그림 1-10 혼합형 컴파일러



- 질의 인터프리터(query interpreter) : 관계 연산자나 부울 연산자 Boolean operator를 포함하는 술어predicate를 하나의 데이터베이스에서 그 술어를 만족하는 레코드를 찾기 위한 명령어로 번역하는 것이다. 여기서 술어는 한 객체의 성질이나 객체와 객체 사이의 관계를 표현하는 것을 말한다.

1.3 번역기의 종류

- **언어 처리 시스템**

  문제가 주어지고 그 문제에 대한 알고리즘을 작성해서 소스 프로그램으로부터 실행 가능한 목적 프로그램을 생성하는 것을 언어 처리 시스템이라 하며, 이때 컴파일러 이외에 여러 가지 다른 번역기가 필요할 수도 있다.

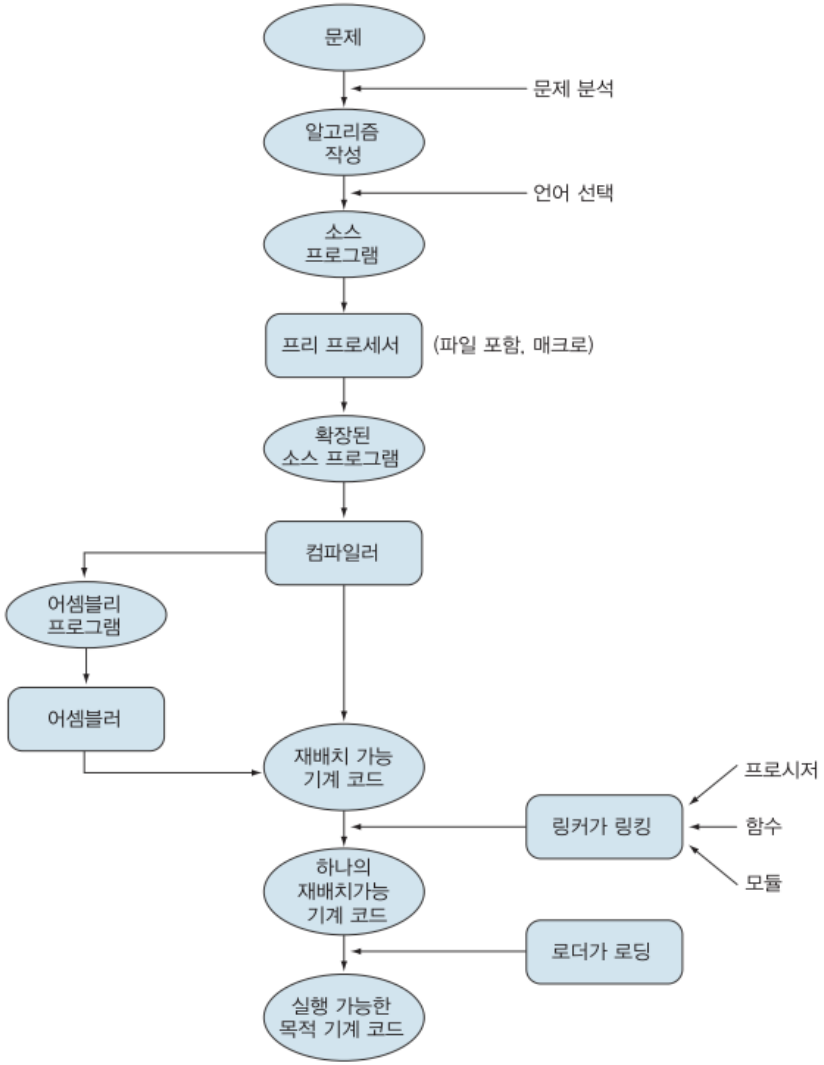


그림 1-11 언어 처리 시스템

Q&A