내공 있는 프로그래머로 길러주는

컴파일러의 이해

Chapter 01
컴파일러의 개요

목차

학습목표

- 컴파일러가 왜 필요한지 컴파일러의 필요성을 언어와 연관지어 이해하기
- 프로그래밍 언어에 대한 일반적인 개념들을 프로그래밍 언어들의 특성별로 나누어서 설명하고 이해하기
- 컴파일러를 포함한 여러 가지 번역기들에 대해서 이해하고, 언어 처리 시스템에 대해서 이해하기

1.1 컴파일러의 필요성

■ 컴파일러 필요성

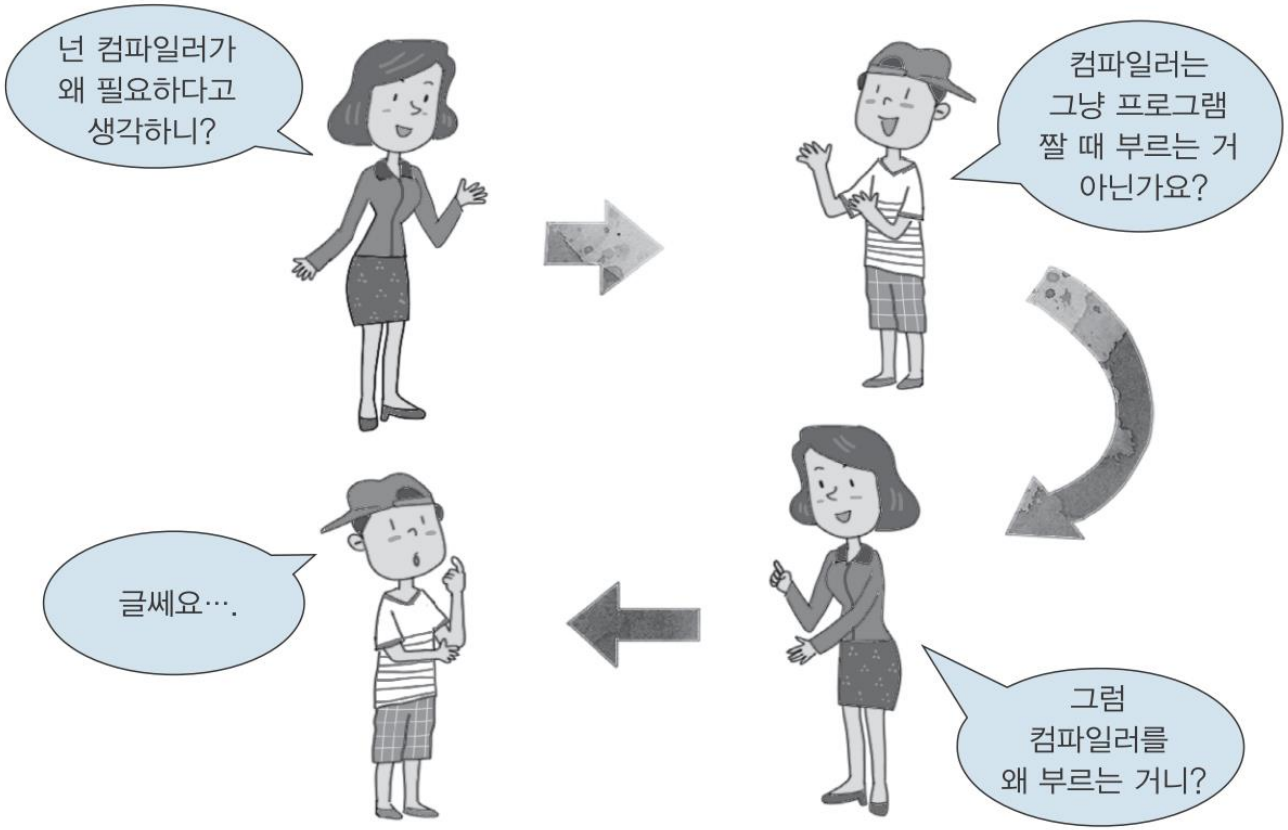

그림 1-1 컴파일러의 필요성

1.1 컴파일러의 필요성

언어란?

- 의사전달을 하기 위한 도구

언어의 종류

- 자연언어(natural language)
- 형식언어(formal language)

형식언어의 종류

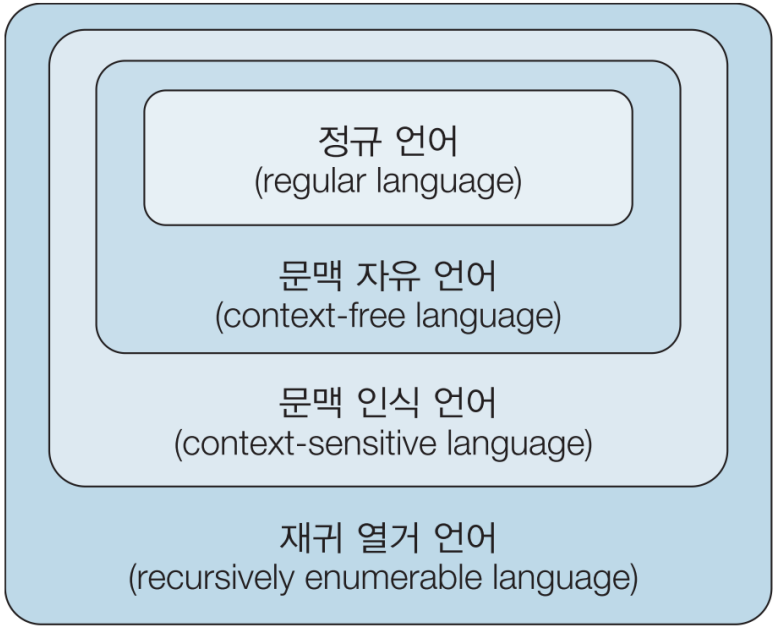

그림 1-2 형식 언어의 촘스키 계층 구조

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 언어와 비절차 언어
- 블록 언어와 비블록 언어
- 절차 언어는 명령형 언어라고도 하며 다음과 같은 특징이 있음
  - 명령의 순차적 실행, 기억장소를 표시하기 위한 변수 사용, 값을 저장하기 위한 치환문 사용

수치계산용 언어

- Fortran(**FOR**mula **TRAN**slation)
  - I, II, III, IV, 77, 86, 90, HPF, FORTRAN D, CEDAR FORTRAN, KSR
  - 매개변수 전달 방식 – 참조 호출(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. 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 있는데, 여기서는 참조 호출(call by reference, call by address, call by location), **값 호출**(call by value), **이름 호출**(call by name)을 간단히 살펴보자.

- **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. 여기서는 부작용과 비슷한 개념으로, 하나의 단위 프로그램에서 2개 이상의 변수가 기억 장소를 공유할 때 발생하는 **별칭**(alias)도 알아야 한다. 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.
- **이름 호출**은 형식 매개변수의 이름이 사용될 때마다 그에 대응하는 실 매개변수 자체가 사용된 것처럼 매번 다시 계산하여 시행하는 방법으로, 초기 프로그래밍 언어인 알골 60에서 사용.

[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 = 5 + 8 = 13 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.

1.2 프로그래밍 언어

- Algol(**ALGO**rithmic **L**anguage)
  - 언어의 구조와 의미가 명료하고 다른 언어에 영향
  - 블록 구조, 재귀법, BNF
  - **블록(Block) 구조** : 프로그램을 여러 단계의 블록으로 나누어 작성할 수 있도록 해주는 언어 구조
    - 프로그램 작성 시 일련의 문장을 하나의 프로시저나 함수로 묶고 이러한 프로시저나 함수를 계층적으로 조직하여 하나의 프로그램을 만들어내는 것.
    - 각 블록은 지역 변수와 다른 블록을 포함하며 변수의 유효 범위가 블록 내로 제한되므로 메모리가 절약되고 에러의 가능성이 작다.
    - 블록 구조를 사용하려면 시스템에서 스택(stack)을 제공해야 한다.
  - **재귀법**(Recursion) : 수학적 귀납법이라고도 불리는 재귀법은 자신을 정의하거나 자기 자신을 재참조하는 방법
    - 알골에서 처음으로 사용
    - 재귀법을 사용하면 프로그램을 간단하게 작성할 수 있지만, 실행 시간이 많이 걸리고 메모리를 많이 요구한다는 단점.
  - **BNF** : BNF(Backus-Naur Form)는 문법을 표현하는 방법으로 가장 많이 사용되는 것이다.
    - 메타 기호(meta-symbol)로 치환을 의미하는 ::=, 논터미널(non-terminal) 기호를 표시하는 < >, 선택을 표시하는 |를 가지고 문법을 표현하는데 더 자세한 내용은 3장에서 다룸.

1.2 프로그래밍 언어

- Basic(**Beginners All-purpose Symbolic Instruction Code)**
  - 프로그램을 잘 알지 못하는 사람들이 프로그래밍을 쉽게 할 수 있도록 개발된 절차형 언어
  - 교육용 언어로 개발되어 언어를 배우기가 쉽고, 문법이 쉬운 언어로 개발된 인터프리터 언어
  - 시분할 시스템(time sharing)
  - **비주얼베이직**(Visual basic) : 윈도용 소프트웨어를 개발하기 위한 프로그래밍 언어로, 그래픽 사용자 인터페이스(graphical user interface, GUI)를 구현하는 프로그램을 매우 쉽게 개발할 수 있다는 것이 특징

■ 사무 처리용 언어 : 코볼이 대표적

- Cobol(**CO**mmon **B**usiness **O**riented **L**anguage) –
  - 아직도 가장 널리 쓰이는 프로그래밍 언어이지만 대학에서는 외면당하는 편이다. 그 이유 중 하나는 복잡한 알고리즘을 프로그래밍하기가 상당히 어렵다는 단점.
  - **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. 코볼은 프로그램이 영어 문장과 유사하여 프로그램 자체가 문서화되는 언어이다
  - **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
  - **자료구조와 실행부분 분리** - 4개 division(identification, environment, data, procedure)

인공지능(Artificial intelligence) 언어

- Lisp(LISt Processing)
  - **함수언어** - 함수형 프로그래밍(functional programming)은 계산을 수학적 함수의 평가로 취급하는 프로그래밍 패러다임의 하나이다.
  - **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 표기법으로, 예를 들면 산술식 a×(b＋c)를 ×a＋bc로 나타내는 것이다.
  - **이진 트리를 선형으로 표기하는 방법**
    - 폴란드식 표기법인 전위(prefix) 표기 방법, 역폴란드식 표기법(reverse Polish notation)이라 불리는 후위(postfix) 표기 방법, 그리고 중위(infix) 표기 방법이 있다.
    - 전위 표기 방법, 후위 표기 방법, 중위 표기 방법은 트리(tree)에 저장되어 있는 자료를 트리 순회(tree traversal)한 결과를 선형으로 표현하는 방법으로, 전위(preorder) 순회한 결과를 전위 표기 방법, 중위(inorder) 순회한 결과를 중위 표현 방법, 후위(postorder) 순회한 결과를 후위 표기 방법이라 한다

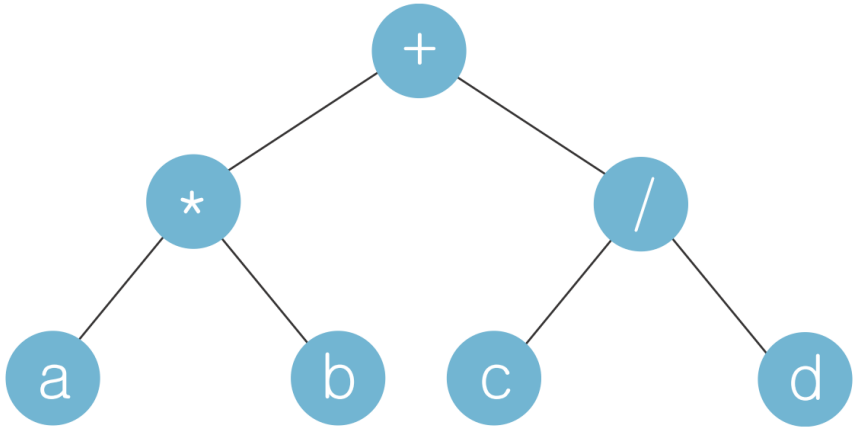

그림 1-4 이진트리

- 전위 순회한 결과인 전위 표현은 +*ab/cd, 후위 순회한 결과인 후위 표현은 ab*cd/+, 중위 순회한 결과인 중위 표현은 a*b+c/d이다.
- 쓰레기 수거(garbage collection) - 메모리 관리 기법 중의 하나로, 프로그램이 동적으로 할당했던 메모리 영역 중에서 필요 없게 된 영역을 해제하는 기능이다. 리스프의 문제를 해결하기 위 해 존 매카시가 개발했다.

- Snobol(StriNg Oriented symBOlic Language)
  - 문자열처리, 패턴 매칭
- Prolog(PRO-gramming in LOGic)
  - 논리 언어

System languages

- C
- C++(**OOPL** : class, inheritance, polymorphism) –
  - C 언어의 명령형 특징을 향상하고 객체지향 프로그래밍을 지원하는 구조를 추가한 것
  - C 언어의 특징을 대부분 포함하고 있어 시스템 프로그래밍에 적합할 뿐만 아니라 클래스, 연산자 중복 등과 같은 특징을 갖추어 객체지향 프로그래밍에 적합
  - **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.

기타

- APL(A Programming Language)
  - 하버드대학의 케네스 아이버슨(Kenneth E. Iverson)이 고안했으며, 배열과 행렬을 포함하는 수학적 연산을 쉽게 프로그래밍하기 위한 언어로 IBM에서 제작했다.
  - 많은 개념이 일반적인 프로그래밍 언어의 규칙에 위배되는 언어이다.
- Pascal
  - 취리히대학의 니클라우스 위스(Niklaus Wirth)가 고안한 프로그래밍 언어로, 프로그래밍에 대한 근본적인 개념을 명확하고 자연스럽게 그리고 체계적으로 가르칠 수 있도록 프로그래밍 언어론적인 관점에서 설계된 교육용 언어이다.
  - 특히 코드를 간결하게 작성할 수 있고, 여러 가지 자료형을 제공하여 자료 구조, 알고리즘 설계 등 컴퓨터 분야의 응용에 적합하다.
- Ada
  - 1980년 미국 국방성
  - 컴퓨터 내장 시스템(embedded computer system)을 지원하기 위한 언어
  - 실시간(Real time) 응용에 적합
  - 다중 처리(multi-tasking), 예외 처리 등의 특징이 있다.

1.2 프로그래밍 언어

- Java
  - 미국의 선마이크로시스템스에서 개발한 객체지향 프로그래밍 언어
  - C++를 바탕으로 언어 규격을 규정하여 더 작고 더 단순하며 더 안전한 언어로 발전
  - 임베디드 시스템을 위한 언어, 가전, 운영체제 및 하드웨어 플랫폼에 독립적인 언어
  - OOPL
- 자바와 자바스크립트의 차이
  - 자바스크립트는 HTML과 같이 사용자 컴퓨터에 의해 인터프리트되는 언어이다. 그러나 자바는 일단 서버 측에서 컴파일해야 하고, 프로그램의 실행이 사용자 측에서 이뤄진다.
  - 자바와 자바스크립트 모두 객체지향적 언어이다. 하지만 자바스크립트에는 상속성이나 클래스가 존재하지 않는다.
  - 자바스크립트는 HTML 코드에 끼워져서(embedded) 사용되지만 자바는 HTML과 독립적으로 사용 가능하다. 단, HTML을 이용해야 자바 프로그램에 접근할 수 있다.
  - 자바와 자바스크립트 모두 안전하다. 그런데 자바스크립트의 경우 HTML 코드에 직접 끼워져 있기 때문에 누구든지 볼 수 있지만 자바는 이와 다르다. 자바 소스코드를 컴파일하면 바이트코드라고 불리는 클래스 파일이 생성된다. 따라서 프로그램 작성자가 디렉터리 안의 소스코드를 지워도 HTML에서 부르는 것은 자바 클래스 파일이기 때문에 다른 사람이 그 소스코드를 보지 못한다는 데 차이가 있다.

1.2 프로그래밍 언어

- 인터넷 프로그램 – 웹 브라우저에서 실행될 수 있는 프로그램. 서버사이드 언어(PERL, PHP, ASP, JSP)와 클라이언트사이드 언어(HTML, DHTML, 자바스크립트, 자바 애플릿)로 구분
  - **펄(PERL)** : Practical Extraction and Report Language의 약자로, 1987년 래리 월(Larry Wall)이 개발했다.
    - » C 언어와 구문이 비슷하며, 초기에는 sh와 awk의 결합으로 만든 유닉스 기능을 포함하는 스크립트 프로그래밍 언어
    - » C 언어의 형태를 가지고 있으면서도 텍스트를 처리하는 기능이나 문자열의 일치 여부 검색, 치환 등의 기능이 뛰어날 뿐 아니라 코딩도 쉽다. 또한 인터프리터 언어이며, 특히 텍스트 처리 기능이 뛰어나 CGI(common gateway interface) 프로그램을 개발하는 데 많이 사용된다.
    - » 유닉스와 윈도 NT 등 여러 가지 운영체제에 사용할 수 있으므로 소스 파일을 이식하기도 쉬워 서버나 운영체제를 바꾸는 경우에도 부담이 되지 않는다.
  - PHP(Personal Hypertext Preprocessor) : HTML에 포함되어 동작하는 스크립트 언어로 처음에는 ' Personal Home Page'라고 불렸다.
    - » 공개된 무료 소스로 원래는 동적 웹 페이지를 만들기 위해 설계되었으며, 이를 구현하기 위해 PHP로 작성된 코드를 HTML 소스 문서 안에 넣으면 PHP 처리 기능이 있는 웹 서버에서 해당 코드를 인식하여 작성자가 원하는 웹 페이지를 생성한다. PHP 스크립트가 포함된 HTML 페이지에는 .php, .php3, .phtml이 붙는 파일 이름이 부여된다

1.2 프로그래밍 언어

- ASP(Active Server Pages) : 하나 이상의 작은 내장 프로그램을 가지고 있는 HTML 페이지를 사용자에게 보여주기 위해 서버에서 수행되는 것이다. 클라이언트가 요청하면 서버에서 응답해주는 방식의 서버 측 프로그램으로, 이를 사용하면 서버에서 웹을 프로그래밍할 수 있고 기존의 HTML 페이지와는 다른 동적인 구성을 할 수 있다. 서버에서 작동하므로 서버의 사양에 따라 속도가 다르며, 클라이언트 측은 인터넷을 사용하는 사용자의 사양에 따라 속도가 다르다.
- **JSP**(Java Server Pages) : HTML 내에 자바 코드를 삽입하여 웹 서버에서 동적으로 웹 페이지를 생성하고 웹 브라우저에 돌려주는 언어이다. JSP는 자바 서블릿으로 변환된 후 실행되므로 서블릿과 거의 유사하다고 볼 수 있으나, 서블릿과 달리 HTML 표준에 따라 작성되므로 웹 디자인을 하기에 편리하다
- **HTML**(HyperText Markup Language) : 월드 와이드 웹을 통해 볼 수 있는 문서를 만들 때 사용하는 언어로, 별도의 컴파일러가 필요하지 않고 웹 브라우저에서 해석이 가능하며 사용하기 쉽다. HTML은 문서의 글자 크기, 글자 색, 글자 모양, 그래픽, 문서 이동(하이퍼링크) 등을 정의하는 명령어 로서 홈페이지는 기본적으로 HTML을 사용하여 구현되며 제목, 문단, 리스트, 하이퍼링크 등이 모두 태그로 구조화된다. 그러나 프로그래밍 코드를 사용할 수 없고 데이터베이스를 연동할 수 없다는 단점이 있다.
- DHTML(Dynamic HTML) : 동적인 작용을 할 수 있는 HTML 문서이다.

1.2 프로그래밍 언어

- **자바스크립트**(Java Script) : 선마이크로시스템스와 넷스케이프가 개발한 스크립트 언어이다. 자바를 응용하여 사용하기 쉽게 만든 자바스크립트는 표준 HTML 문서에 사용되어 쌍방향 interactive 웹 페이지를 만들 수 있게 해준다.
- **자바애플릿**(Java applet) : 자바 언어로 작성된 작은 소프트웨어로서, 'applet'은 '응용'을 뜻하는 'application'과 '작다'는 의미의 접미사 ' let'을 조합한 합성어이다. 크기가 작아 네트워크에서의 전송에 적합하고, 월드 와이드 웹을 사용하여 배포할 수 있다. 독립 실행을 하는 자바애플리케이 션과 달리 웹 문서상에 포함되어 클라이언트(브라우저)의 요구 시 실행 코드를 내려 받아 사용자 의 마우스 동작, 문자 입력 등에 따라 작동하는 형태이다. 일단 브라우저에서 실행되면 동적이고 멀티미디어적인 기능을 발휘하지만 그 실행까지가 너무 느리다는 것이 단점이다.

프로그래밍 언어의 기본 개념

식별자(identifier) 와 변수(variable)

- 식별자와 변수는 만드는 방법이 이름(name)과 동일하다. 식별자는 어떤 대상을 유일하게 식별 및 구별할 수 있는 이름을 뜻하며, 이는 정보를 처리하려면 그 정보를 가리킬 방법이 있어야 하기 때문에 필요하다. 변수는 실행 시간에 저장된 값이 변경될 수 있는 객체를 의미하며, 값을 저장할 기억 장소의 특별한 위치를 나타낸다.
- 많은 프로그래밍 언어는 식별자로 사용될 수 있는 문자에 제한을 두는데, 예를 들어 C와 C++ 계열에서는 영문 대문자와 소문자, 숫자, 언더바(underscore)만 식별자로 쓸 수 있으며 a123, aaa는 모두 식별자와 변수가 될 수 있다. 변수가 되려면 값을 저장하는 기억 장소의 위치를 나타내야 한다.

1.3 번역기의 종류

번역기란 ?

- 하나의 프로그래밍 언어로 작성된 프로그램을 입력으로 하여 그와 동등한 의미를 갖는 다른 프로그래밍 언어로 된 프로그램을 출력하는 일종의 프로그램

- 번역기의 종류와 기능

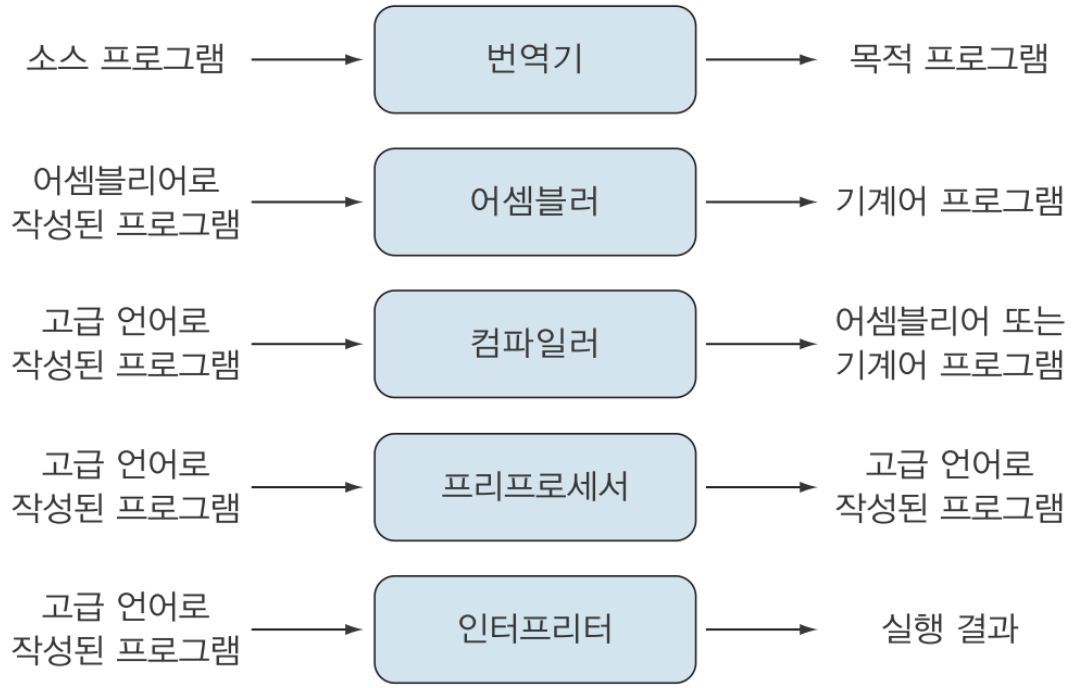

그림 1-5 번역기의 종류와 기능

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 (레지스터 R_1 에 있는 값과 상수 2를 더해서 다시 레지스터 R_1 에 저장)

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

- 이렇게 해서 b = a+2를 계산

1.3 번역기의 종류

- 대부분 two pass assembler로 구성
  - pass : 하나의 입력 파일(file)을 단 한번만 읽는 것으로 구성되는 단위로 하나의 pass는 일반적으로 여러 개의 phase들로 구성됨
  - phase : source program을 하나의 표현에서 다른 표현으로 변환하는 것
  - first pass : assembly code를 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

  - second pass : operation code를 기계어로 표현하는 bit들의 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
  - 코볼 프로그램은 있는데 코볼 컴파일러가 없을 때, C 컴파일러가 있다면 실행해 볼 수 있다. 코볼 언어로 작성된 프로그램을 C 언어로 작성된 프로그램으로 변환하고 C 언어로 변환된 프로그램을 C 컴파일러를 사용해서 실행할 수 있다. 이때 코볼을 C로 변환하는 코볼-C 프리프로세서가 필요하다.
- C 프리프로세서의 3가지 기능

  1) 파일 포함(file inclusion) 기능
  - 프로그램에 헤더 파일(header file)들을 포함
  - 예 - C 프리프로세서는 파일에 #include <global.h>라는 문장이 포함되어 있다면 그 자리에 <global.h> 파일의 내용을 대체.

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

1.3 번역기의 종류

- 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

- 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

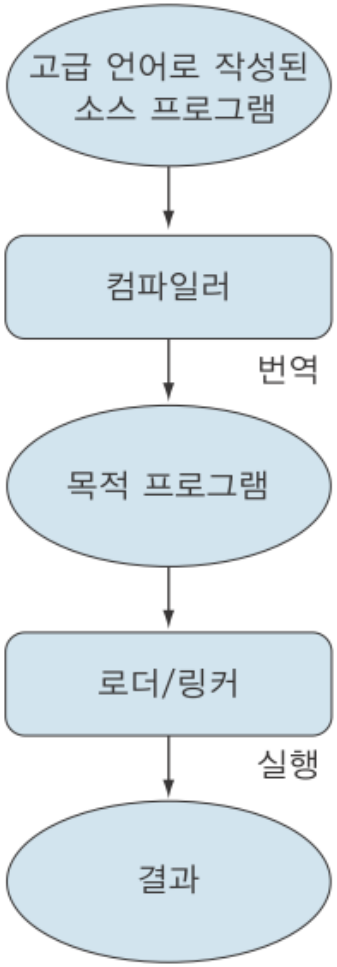

그림 1-8 컴파일러의 간단한 처리 과정

1.3 번역기의 종류

■ 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

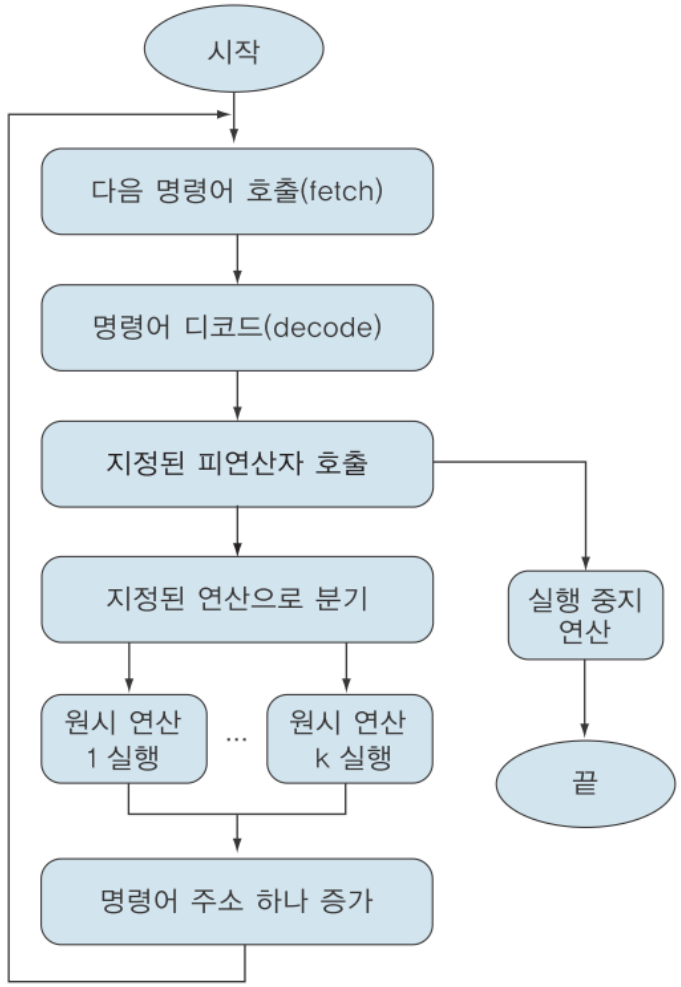

그림 1-9 인터프리터의 처리 과정

1.3 번역기의 종류

- **컴파일러와 인터프리터의 비교(반복문, speed, storage)**
  - 컴파일러 언어 - FORTRAN, ALGOL, PL/I, PASCAL, COBOL, C
  - 인터프리터 언어 - LISP, SNOBOL, APL

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 기억장소의 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

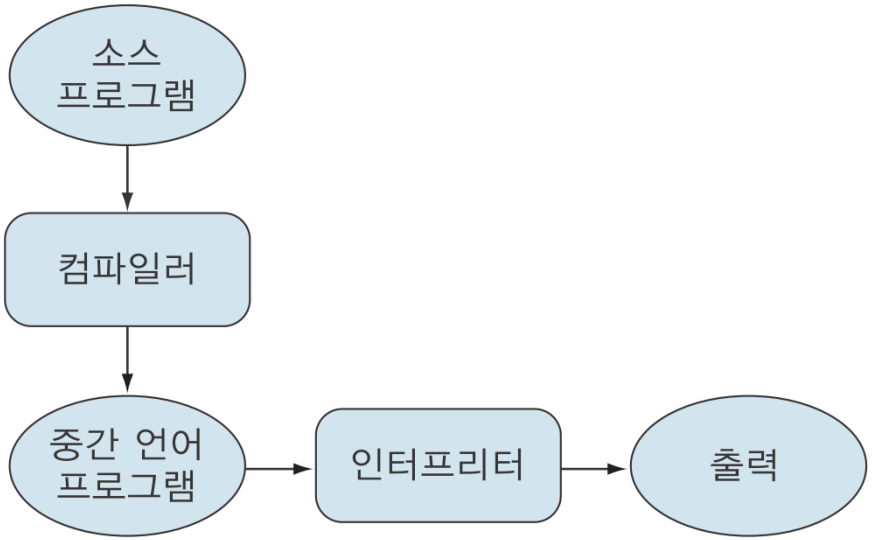

그림 1-10 혼합형 컴파일러

- 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.

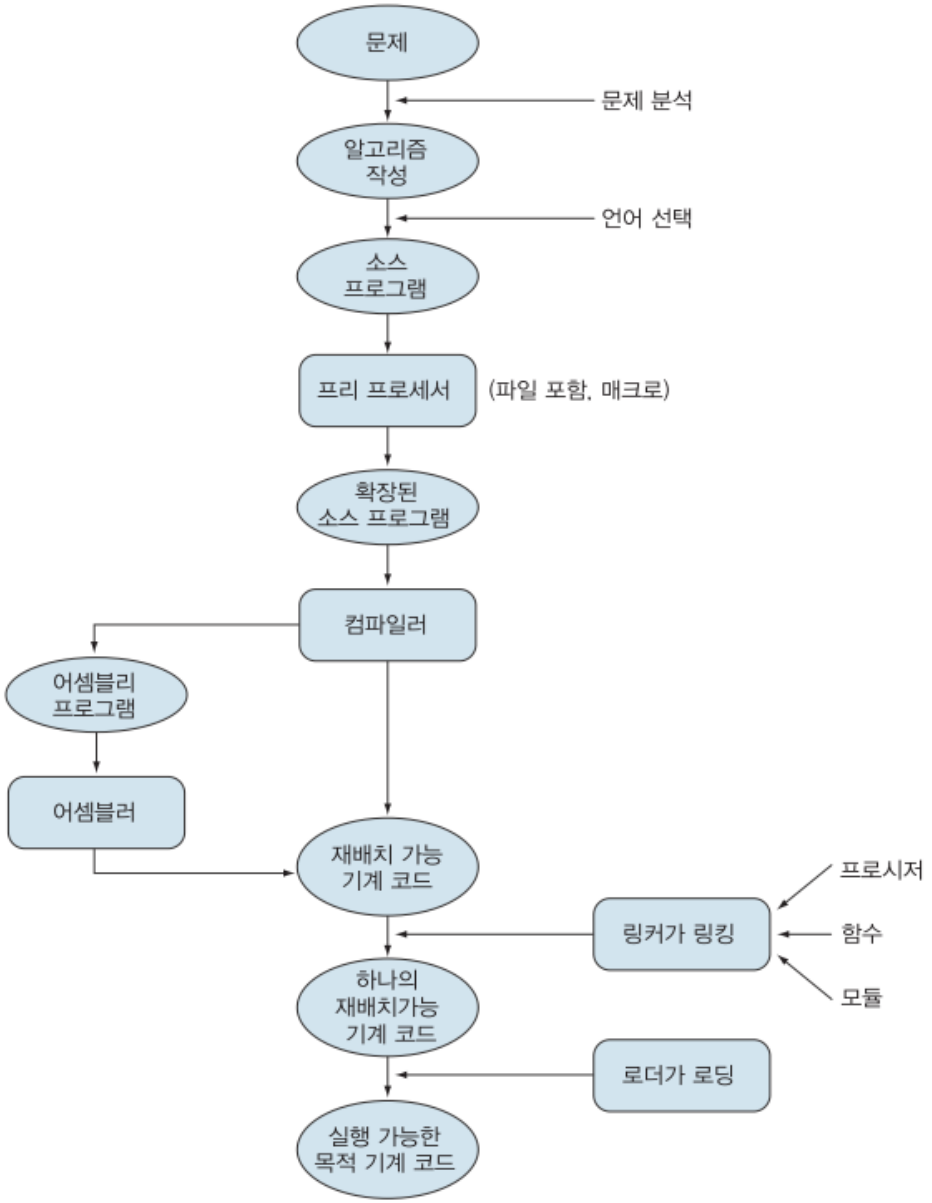

그림 1-11 언어 처리 시스템

Thank You