경영학부 경영학 전공
20132651 이 성재
2018년 9월 7일

Compiler Assignment

Chapter 1. Introduction to Compiler

1.1 다음 괄호에 알맞은 말을 쓰시오

(1) 번역기 (translator)

(2) 구조 (syntax)

(3) 크로스 컴파일러 (corss compiler)

(4) 코드 최적화 (code optimization)

(5) 스캐너 (scanner) / 파서 (parser)

(6) 추상 구문 트리 (abstract syntax tree)

(7) precode optimization / postcode optimization

(8) N * M 개

(9) 컴파일러-컴파일러 (compiler-compiler)

(10) 토큰 표현 (token description) / 어휘 분석기 (lexical analyzer)

(11) 문법 표현 (grammar description) / 파싱 테이블 (parsing table) + 파서 (parser)

(12) 기계 표현 (machine description)

(13) 렉스 (LEX)

(14) TCOL (Tree Common Oriented Language) / EM (Encoding Machine)

(15) MSIL (MicroSoft Intermediate Language) / CLR (Common Language Rutime)

1.2 다음 약어에 대한 원어를 쓰시오

(1) PGS : Parser Generating System

(2) YACC : Yet Another Compiler-Compiler

(3) PQCC : Production-Quaility Compiler Compiler / TCOL : Tree Common Oriented Language

(4) ACK : Amsterdam Compiler Kit / EM : Encoding Machine

(5) JIT : Just In Time

1.3 컴파일러의 일반적인 구조를 어휘 분석기, 구문 분석기, 중간 언어 생성기, 코드 최적화기, 그리고 목적 코드 생성기로 나누어 설명하고 특히, 각 단계의 입력과 출력을 설명하시오.

컴파일러는 크게 전단부(front-end)와 후단부(back-end)로 나누어지며, 전단부에서는 어휘분석기, 구문분석기, 중간 언어 생성기가 소스프로그램을 분석하고 중간 코드를 생성한다. 첫 단계인 어휘 분석기에서는 소스 프로그램을 읽어 들여 문법적으로 의미를 갖는 최소 단위, 토큰을 생성한다. 이 때 토큰은 지정어, 연산자 기호, 구분자 등의 특수 형태와 프로그래밍에서의 명칭 및 상수인 일반 형태로 구분된다. 두번째 단계인 구문분석기는 앞에서의 토큰을 입력받아 에러를 체크하고, 구문 구조를 만든다. 구문 구조는 파스 트리나 추상 구문 트리가 있으며, 최근의 대부분 컴파일러는 추상 구문 트리를 사용한다. 전단부의 마지막인 중간 코드 생성기는 추상 구문 트리를 입력으로 받아 의미 검사를 행하고, 이에 해당하는 중간 코드를 생성한다.

다음으로 후단부에서는 코드 최적화기, 목적 코드 생성기가 최종적인 목적 코드를 생성해낸다. 우선 코드 최적화기는 중간 코드를 입력으로 받아, 지역 최적화 및 전역 최적화, precode 최적화 및 postcode최적화 등으로 비효율적 코드를 효율적으로 개선한다. 최종적으로 목적 코드 생성기는 최적화된 중간 코드를 입력으로 받아, 이에 의미적으로 동등한 목적 기계에 대한 코드를 생성한다. 먼저 목적 코드를 선택 및 생성하고, 레지스터의 사용을 지정한 다음 기억 장소를 할당하고, 기계 의존적인 코드를 최적화한다. 이상이 컴파일러의 일반적인 구조이다.

1.4 코드 최적화는 관점에 따라 지역 최적화와 전역 최적화로 구분할 수 있다. 지역 최적화 (펍볼 최적화)에 의해 얻을 수 있는 효과를 열거하시오

코드 최적화기의 지역 최적화 방법에 의해서 주로 컴파일 시간 상수 연산 (constant folding), 중복된 load, store 명령문의 제거, 식(expression)의 대수학적 간소화 (algebraic simplification), 연산 강도 경감(strength reduction), 불필요한 일련의 코드 블록 (null sequence) 삭제등의 효과를 얻을 수 있다.

1.5 목적 코드 생성기가 목적 코드를 생성하기 위하여 행하는 작업을 크게 4가지로 나누어 설명하시오.

목적 코드 생성기는 목적 코드의 선택 및 생성, 레지스터의 운영, 기억 장소 할당, 기계 의존적인 코드 최적화를 통해 목적 코드를 생성한다.

1.6 렉스와 YACC 의 기능을 간단히 설명하시오. 그리고 렉스와 YACC를 이용하여 컴파일러의 전단부를 구현하는 방법을 설명하시오

렉스는 어휘 분석기 생성기중 하나로, 토큰의 형태를 묘사한 정규 표현들과 각 정규 표현이 매칭되었을 때 처리를 나타내는 액션 코드로 구성된 입력을 받아 어휘 분석의 일을 처리하는 프로그램을 출력한다. YACC는 파서 생성기중 하나로, 문법 규칙과 그에 해당하는 액션 코드를 입력으로 받아 파싱을 담당하는 프로그램을 출력한다. 이 둘을 이용하여 컴파일러의 전단부를 구현할 수 있으며, 그 과정은 다음과 같다. 우선 정규 표현과 액션 코드를 기반으로 렉스에서 lex.yy.c 이름의 어휘 분석 작업을 담당하는 C 프로그램을 생성한다. 이는 해당 언어의 어휘 분석기가 된다. 다음으로 문법 규칙과 액션 코드를 기반으로 YACC 에서 y.tab.c 이름의 구문 검사 작업을 담당하는 C프로그램을 생성한다. 이는 해당 언어의 구문 분석기가 된다.

1.7 컴파일 과정을 어휘 분석, 구문 분석, 중간 코드 생성, 코드 최적화, 목적 코드 생성 단계로 나누어 설명하고 각 단계를 자동적으로 구성하기 위한 컴파일러 제작 도구에 관하여 논하시오

컴파일은 사람이 작성한 고차원 언어의 소스 프로그램을, 기계가 실행 가능한 파일로 변환하는 과정이다. 가장 먼저 어휘 분석기에서 해당 소스 프로그램을 토큰 단위로 변환한다. 그 다음, 구문 분석기에서 연속적인 토큰에서 에러를 검사하고 추상 구문 트리로 변환한다. 생성된 트리는 중간 코드 생성기를 통해 중간 코드로 변환되게 된다. 여기까지는 컴파일러의 전반부이며, 후반부에서는 중간 코드의 최적화 및 목적 코드의 생성이 이루어진다. 생성된 중간 코드는 코드 최적화기를 거쳐 지역 / 전역 최적화가 진행된다. 최종적으로 최적화된 중간 코드는 목적 코드 생성기를 통해 기계가 직접 실행 가능한 목적 코드로 변환된다.

이러한 컴파일 전반에 걸친 과정을 자동화하기 위해서 만들어진 컴파일러 제작 도구는 대표적으로 PQCC 와 ACK 가 있다. PQCC 에서는 트리 형태의 중간 언어인 TCOL 을 사용하였으며, 그 시스템의 기능은 언어와 목적 기계에 대한 정형화된 표현을 입력으로 받아 PQC라고 부르는 기초 코드 생성기와 최적화기가 사용하는 테이블을 만들어 낸다. 그리고 PQC는 이 테이블을 이용하여 전단부에서 생성한 TCOL 을 목적 코드로 번역하는 작업을 한다. ACK의 경우 컴파일러의 후단부를 자동화하기 위한 도구의 하나로서, 이식성과 재목적성이 매우 높은 컴파일러를 만들기 위한 실질적인 도구이다. ACK에서는 언어에 대한 전단부가 중간 언어인 EM코드를 생성해 낸다. EM-코드는 가상적인 스택 기계에 근거를 둔 일종의 어셈블리어이다. 그리고 생성된 EM 코드에 대한 최적화 과정을 거친 후에, 후단부에 의해 목적 프로그램으로 변환된다.