자바 클래스 파일을 생성하는 C 컴파일러 설계†

최 원 호°*, 정 민 수*, 김 도 우*, 진 민*, 윤 기 송**
* 경남대학교 컴퓨터공학과
** 한국전자통신연구원 실시간시스템팀

Design of C Compiler to Generate Java Class File

Won-Ho Choi* , Min-Soo Jung* , Do-Woo Kim*, Min Jin*, Ki-Song Yoon**
* Dept. of Computer Engineering , Kyungnam University.
** Realtime System Group. ETRI

요 약

기존의 고급 언어들로 작성된 응용 프로그램들은 인터넷이나 다른 플랫폼(Platform)에서 수행되는 것이 어렵다. 본 논문에서는 자바의 플랫폼 독립적인 특성을 C 언어와 같은 고급 언어에 적용해 보고자 한다. 즉, C 언어로 작성된 원시 코드(Source code)를 컴파일러를 이용해서 자바 가상 기계가 인식할 수 있는 목적 코드인 바이트 코드(Bytecode)로 변환하여 자바 가상 기계(Java Virtual Machine)가 탑재되어 있는 어떠한 플랫폼에서도 수행할 수 있도록 하는 것이다. 따라서, 본 논문에서는 C 언어로 작성된 응용 프로그램을 이기종 플랫폼 상에서 실행 가능 하도록 바이트 코드를 생성하는 컴파일러를 설계하고자 한다.

1. 서 론

자바는 분산 환경에 적합하고, 플랫폼 독립적이고, 코드 재사용이 용이하며, 프로그래밍이 단순하다는 장점을 가지고 있다. 이러한 이유로 자바는 인터넷 브라우저 환경, 데이터 베이스 연동, 디지털 가전기기의 내장 시스템에 이르기 까지 많은 분야에 적용이 되고 있다. 특히, 자바의 플랫폼 독립적인 특성으로 인하여 인터넷에서 더욱더 큰 진가를 발하고 있고, 현재 많은 응용 프로그램들이 자바 언어로 작성이 되고 있다.

가장 많이 사용되고 있는 C 언어와 같은 고급 언어들은 이식성은 좋으나 자바와 같은 플랫폼 독립적인 특성들을 가지고 있지는 않다. 이들 언어들을 이용 인터넷 프로그램이나 각종 플랫폼을 가진 디지털 가전기기의 내장 프로그램을 작성하는 것은 적당하지 않다. C 언어와 같은 고급 언어가 플랫폼에 종속되지 않는 프로그램을 구현하기 위해서는 자바 형태로의 변환이 필수적이다

이와 같은 변환을 하기 위해서는 먼저 C 언어로 작성된 소스 코드를 컴파일러를 통해 자바 가상 기계에서 인식될 수 있는 목적 코드인 클래스 파일(class 파일)로 변환하고, 그 변환된 목적 코드(Target.class)를 가지고 자바 가상 기계에서 실행을 하면 된다. 따라서, 자바 가상 기계를 목적으로 하는 컴파일러를 작성하면, C 언어에 익숙한 사용자들도 자바와 같은 플랫폼 독립적인 응용 프로그램을 작성할 수 있다.

본 논문은 C 컴파일러를 통해 플랫폼 독립적이고 자바 가상 기계에서 실행될 수 있는 바이트코드를 생성하고 고급 언어에서 객체지향 언어로의 변환과정을 이해하게 해준다

본 논문의 구성은 2 장에서 C 원시 코드를 컴파일 할 때 C 코드의 제약 사항과 생성 규칙들을 제시하고, C 컴파일러의 구성에 관해 알아보고, 3 장에서는 코드 생성 방법에 대해서 설명하고, 4 장에서는 결론 및 향후 연구 방향을 제시한다.

2. C 컴파일러의 생성 규칙과 구성

2.1 생성 규칙

생성할 C 원시 코드는 전역 변수 선언, 지역 변수 선언, 함수 선언, if-else 와 if 선언, while 루프 선언 등과 같은 간단한 문장들로 구성한다. 변수 선언과 함수의 매개 변수나 반환값의 형태는 정수(int)로 한다.[6]

아래와 같은 간단한 생성 규칙들을 제시한다.

```
Program          :  DeclarationList
DeclarationList  :  Declaration DeclarationList
                 |  Declaration
Declaration      :  VariableDeclaration
                 |  FunctionDeclaration
VariableDeclation:  TypeSpecID;
                 |  TypeSpecID[NUM];
```

† 본 연구는 한국학술진흥재단 우수 연구소 과제연구비 지원으로 수행되었습니다.

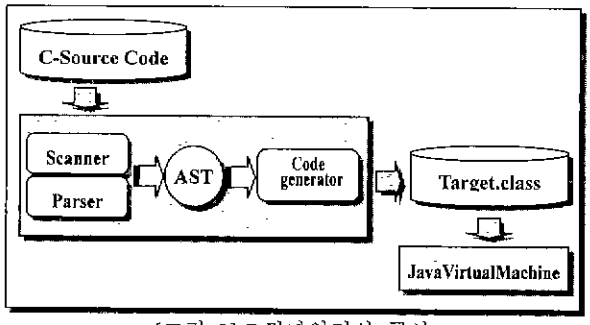
```
                    ··· <중략> ···
Call        :  ID(Args)
Args        |  ArgList
            |
ArgList     :  Expression, ArgList
            |  Expression
```

[그림 1] 생성 규칙

2.2 컴파일러 구조

C 컴파일러는 스캐너(Scanner : Lex), 파서(Parser : YACC), 코드 생성기(Code generator) 이들 세가지 부분으로 구성되어 있다

먼저 스캐너로 C 원시 코드로부터 필요한 토큰(token)을 구성하고, 생성된 토큰은 NUM 토큰과 ID 토큰으로 저장된다. 두번째, 저장된 토큰들을 파서에 전달해서 입력된 값들이 문법적인 오류를 벗어나는지를 검사하고 AST(Abstract Syntax Tree)를 생성한다. 마지막으로 생성된 AST 를 통해서 자바 가상 기계에서 올바르게 실행되는 목적 코드인 바이트코드를 생성한다. [5,6]



[그림 2] C 컴파일러의 구성

그림 2 에서는 고급 언어를 컴파일하는 과정에서와 같이 일반적인 중간 코드 생성은 이루어지지 않고 바로 목적 코드를 생성한다. 이는 자바 가상 기계가 바이트코드를 플랫폼에 맞게끔 이용하기 때문이다. 그리고 생성된 바이트코드에서 C 언어가 제공하는 라이브러리에 관한 것은 자바에서의 클래스에 대한 정보로 변환하여 자바 가상 기계가 동적으로 클래스 로더(Class loader)를 통해서 해당 클래스를 찾을 수 있게 하는 정보를 추가한다 [1,7,8]

3. 코드 생성기 설계

코드 생성기를 설계하기 위해서는 자바 가상 기계와 다른 스택 기반 기계 플랫폼과의 유사점과 차이점들을 알아야 한다. 두가지 기계에서 가장 큰 차이점은 자바 가상 기계는 컨스턴트 풀(Constant Pool)을 데이터 구조로 사용한다는 것이다. 클래스 파일 내의 컨스턴트 풀은 자바 가상 기계에서 사용되는 다양한 클래스와 메소드 그리고 변수들에 대한 정보를 저장하고 있다. 두번째, 변수와 함수를 참조하는 자바 가상 기계의 메소드들이다. 자바 가상 기계는 순수 객체지향을 따르고 있기 때문에 다른 스택 기반 기계에서 사용되는 전역변수를 지원하지 않는다.

따라서 자바 가상 기계가 생성된 바이트코드를 가지고 올바르게 수행하기 위해서 코드 생성기는 컨스턴트 풀, 변수와 함수의 참조를 반드시 사용하고, 전역변수 선언을 자바 가상 기계가 알 수 있게 처리해야 한다.

3.1 컨스턴트 풀 생성

자바 가상 기계는 클래스 파일, 즉 바이트코드 내에서 컨스턴트 풀에 저장된 스트링, 클래스 이름, 필드 이름, 메소드 이름 등을 참조해서 클래스 파일을 실행시킨다.

컴파일러는 바이트코드를 생성하기 전에 자바 가상 기계 실행에서 꼭 필요한 컨스턴트 풀을 생성하고 조직화 해야 한다. 컴파일러는 컨스턴트 풀내의 항목들에 대한 접근 권한이 필요하고, 또 새로운 항목들을 컨스턴트 풀에 첨가 해야 하기 때문에 동적으로 증가하기 쉬운 배열로서 코드 생성기에서 컨스턴트 풀을 표시하는 것이 가장 적합하다. 코드 생성기는 AST 를 통해서 컨스턴트 풀에 항목들을 첨가하고 변수와 함수들을 접근할 수 있다.[1,3]

3.2 변수와 함수 참조

자바 가상 기계와 다른 스택 기반 기계의 차이점은 객체지향적 관점에서 뚜렷하게 나타난다. 객체지향 언어는 클래스와 객체라는 개념만을 사용하고, 변수와 함수의 선언은 반드시 클래스나 객체 내에서만 이루어져야 한다. 클래스 내에 선언된 필드나 메소드를 참조할 때는 메소드를 이용할 수 있는 권한이 있어야 한다. 따라서 전역으로 선언된 변수나 함수는 자바 가상 기계 내에 표시하기가 불가능하다 하지만 클래스 내에 static 으로 선언된 필드나 메소드에 대한 참조는 접근 권한이 없어도 참조할 수 있다

따라서, 코드 생성기가 전역 변수나 함수를 참조하는 코드에 대한 바이트코드를 생성할 때, 코드 생성기는 전역 변수와 함수를 public 이나 static 으로 표시하고, 이를 컨스턴트 풀 항목에 추가하여 바이트코드를 생성할 수 있다.[9]

3.3 바이트코드 생성

C 언어로 작성된 원시 코드는 객체지향 언어와 달리 클래스를 갖지 않는다 그러나 자바 가상 기계는 객체지향 언어에 대해서 설계되었기 때문에 컴파일 과정에서 원시 코드가 객체지향 언어에 맞는 클래스를 생성해야 한다. 원시 코드 내에 모든 함수와 전역 변수들은 목적 코드에서 public 이나 static 으로 선언된 메소드나 필드가 되어야 한다.

코드 생성기에 의해 클래스의 필드와 메소드로 변환되는 전역 변수와 함수 선언에 대한 구문 분석과정을

살피보면, 코드 생성기는 모든 변수와 함수 식별자를 심볼 테이블 스택에 저장한다. 식별자명은 관련된 함수나 변수 정보에 의해 현재 심볼 테이블 영역 내에 첨가된다. 식별자를 심볼 테이블 내에서 찾을 때 식별자명과 같이 추가된 정보는 그 식별자의 변수나 함수에 접근하기 위해 사용된어진다.

3.3.1 세가지 형태의 식별자

컴파일러 설계에서 고려된 세가지 형태의 식별자는 전역 변수와 지역 변수 그리고 함수이다. 먼저 전역 변수에서, 코드 생성기가 전역 변수 선언을 가져오면 그 전역 변수에 접근을 요구하는 컨스턴트 풀 항목이 생성되고 컴파일러에 의해 컨스턴트 풀에 첨가된다. 그런 후에 전역 변수의 존재를 자바 가상 기계가 인식하도록 하기 위해 전역 변수에 대한 바이트코드를 임시 파일에 작성한다. 만약 전역 변수가 배열 포인터이면, 배열을 초기화하기 위해서 컴파일러는 생성자를 또 다른 임시 파일에 추가한다. 코드 생성자는 변수에 대한 식별자명을 심볼 테이블에 저장한다. 그 다음에 변수와 그 변수를 제공하고 있는 컨스턴트 풀 엔트리에서 참조 인덱스의 식별자명은 심볼 테이블에 저장된다.

두번째, 함수 선언도 전역 변수 선언과 비슷하다. 함수의 식별과 관련된 컨스턴트 풀 항목을 생성하고, 컨스턴트 풀에 저장한다. 또 함수의 동작과 관계된 바이트코드는 함수 선언을 위한 또 다른 임시 기억 장소에 작성된다. 그리고 함수는 자바 오퍼랜드 스택(Operand Stack)에 푸시(push)되고, 컨스턴트 풀 항목은 심볼 테이블에 추가된다.

마지막으로, 지역 변수 선언이 함수 내에서 해석되면, 코드 생성기는 이미 선언된 다른 지역 변수의 전체 수에 함수 인자의 수를 더하여 증가 시킨다. 코드 생성기는 참조 인덱스를 가지는 변수 식별자명을 심볼 테이블 내에 추가할 것이다.

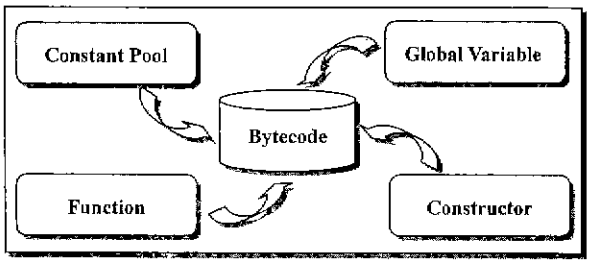
3.3.2 프로그램 카운터

자바 가상 기계에 대한 생성 코드와 다른 스택 기반 기계 플랫폼들에 대한 생성 코드 사이에 차이점은 메모리 관리에 대해 자바 가상 기계에서 메소드와 프로그램 카운터가 동작하는 방식에서 구별된다. 스택, 힙, 프레임에 대한 메모리 할당은 자바 가상 기계에 의해서 자동적으로 수행되기 때문에 코드 생성기가 함수 호출에 관하여 스택이나 프레임 포인터를 명확하게 이동 시키거나 저장하는 바이트코드를 생성할 필요는 없다 단지 코드 생성기는 메모리 할당하는데 있어 크기를 자바 가상 기계에 알려주는 코드를 생성하면 된다. 스택과 프레임 포인터를 저장하고 복귀하는 동작은 함수가 호출되거나 반환될 때 자바 가상 기계에 의해서 자동적으로 수행된다

프로그램 카운터는 코드 생성기에 의해서 직접적으로 변화되지 않고 자바 가상 기계에 의해 간접적으로 증가되고 관리된다. 그래서 조건문과 반복문 동안 프로그램의 다른 부분으로 점프하기 위하여 사용된 바이트코드는 프로그램 카운터 관리보다 바이트코드 카운트를 통해 수행된다. 코드 생성기는 함수 내의 코드 실행에서 특별한 코드로 점프하기 위해서 필요한 바이트코드를 생성한다.

3.3.3 생성자

전역 배열 변수의 선언때문에 요구된 생성자에 대한 임시 파일은 원시 코드를 통해 만들어지고, 컴파일러는 생성자가 전역 배열 변수를 해석할 때마다 바이트코드에 생성자를 추가 시키는 방법을 요청한다.



<그림 3> 네개의 임시 파일과 바이트코드

코드 생성이 완성될 때, 목적 클래스를 위한 컨스턴트 풀, 전역 변수, 함수, 생성자의 네 개의 임시 파일은 완전한 목적 코드 내로 조합한다.

4. 결론 및 향후 연구 방향

본 논문에서는 자바 가상 기계가 탑재되어 있는 여러 가지 다양한 플랫폼에서도 C 언어와 같은 고급언어로 작성된 프로그램을 컴파일 과정을 거쳐서 자바로 작성된 프로그램의 실행을 위해 설계된 자바 가상 기계에서도 실행이 가능한 바이트코드를 생성하는 C 컴파일러의 설계를 제안하였다. C 언어로 작성된 코드를 자바 프로그래밍에 의해 다시 자바로 번환하는 과정이 필요 없이 바로 이 컴파일러를 통해 자바 가상 기계에 실행이 가능하도록 하였다.

향후 연구 방향으로는 고급 언어에서 객체지향 언어로의 변환이 보다 효율적으로 이루어지도록 하는 것이다.

참고 문헌

[1] B. Venners, "Inside the Java Virtual Machine", McGraw-Hill, (1977)
[2] J. Gosling, "The Java Language Specification", Addison-Wesley, (1996)
[3] J. Meyer and T. Downing, "Java™ Virtual Machin", O'Relly, (1997)
[4] K. Louden, "Compiler Construction: Principles and Practice", (1997)
[5] J. R. Levine and T. Mason, "lex & yacc", O'Relly, (1993)
[6] 오세만, "컴파일러 입문", 정익사, (1996)
[7] 류동항, 정민수, "바이트 코드 분석기의 설계 및 구현", 한국정보과학회 제 25 회 춘계학술발표논문집, (1998)
[8] 옥재호, 정민수, "자바 클래스 파일 브라우저의 설계 및 구현", 한국정보과학회 제 25 회 추계학술발표논문집,(1998)
[9] 은준석, 정대교, "이 기종 컴퓨터 환경을 위한 제어 언어의 자바 언어 변환에 관한 연구", 한국정보과학회 제 25 회 춘계학술발표논문집,(1998)
[10] http://java.sun.com/, Sun Microsystems, Java Home Page
[11] http://www.suntest.com/Javacc, Sun Microsystems, Java Compiler Compiler – The Java Parser Generator