Scheme 프로그래밍 모바일 앱 설계 및 구현

김동섭[1], 송동수[2], 우균[3]
[1,2]부산대학교 정보융합공학과
[3]부산대학교 정보컴퓨터공학부
{seob614, song97, woogyun}@pusan.ac.kr

Design and Implementation of a Scheme Mobile Programming Application

Dongseob Kim[1], Dongsu Song[2], Gyun Woo[3]
[1,2]Dept. of Information Convergence Engineering, Pusan National University
[3]School of Computer Science and Engineering, Pusan National University

요 약

프로그래밍 모바일 앱은 장소에의 제약성 해소와 장비의 경량화가 가능하다. 프로그래밍 실습을 위한 환경으로 PC나 서버 연결 온라인 환경 등이 주로 사용되나 모바일 앱은 거의 사용되지 않는다. 이 연구에서는 모바일 앱을 활용하여 프로그래밍할 수 있는 환경을 설계하고 구현한다. 프로그래밍 언어로는 LISP의 파생어인 Scheme을 사용하였다. Scheme 언어는 다중 패러다임 언어로서 프로그래밍 교육에서 다양한 관점으로 문제 해결 방식을 제공할 수 있다. 이를 통해 Scheme 언어에 대한 인터프리터를 서버리스 프로그래밍 앱으로 설계하고 구현하는 과정을 기술한다. 구현 결과에 대한 인터프리터 처리 성능 실험으로 재귀 함수로 피보나치 수열을 계산하였을 때 PC 수행 시간에 대한 모바일 버전 수행 시간 백분율 기하 평균은 0.96으로 모바일 환경에서도 일반 컴퓨터 환경에 버금가는 처리 성능을 얻음을 확인하였다.

1. 서론

컴퓨터 프로그래밍 교육의 필요성과 관심으로 인해 현재 전국 대부분의 초·중·고교에서 컴퓨터 관련 수업을 진행한다[1]. 하지만 실제 수업을 하는 교사의 전문성이 부족하며, 무엇보다도 컴퓨터가 노후화하여 최신 코딩 환경 구축에 어려움을 겪는 일이 종종 발생한다. 이러한 상황을 해결하기 위한 환경 구축 방안이 필요하다.

실제로 이러한 점을 보완하기 위해 온라인이나 모바일 앱을 통한 프로그래밍 환경이 대안으로 제시되고 있다. 또한, 여러 프로그래밍 언어가 온라인 및 모바일 환경에서 프로그래밍 실습을 할 수 있도록 지원하기도 한다. 그러나 대상 언어가 복잡하기도 하며 널리 알려지지 못해서 사용이 저조하다. 본 연구에서는 함수형 프로그래밍 언어 LISP의 교육용 버전인 Scheme의 프로그래밍 환경을 제안한다.

본 논문에서는 인터넷 연결 없이도 모바일 환경에서도 Scheme 프로그래밍을 할 수 있는 앱을 제안한다. 이 앱을 설치하면 누구나 쉽고 편리하고 장소와 관계없이 Scheme을 사용할 수 있다. 제안한 앱의 사용성을 입증하기 위해 안드로이드 환경과 PC 환경에서 같은 프로그램을 구현한 후 이를 바탕으로 한 비교 실험을 수행한다.

본 논문은 다음과 같이 구성된다. 2장에서는 제안 시스템 개발과 관련된 여러 연구에 대해 살펴본다. 3장에서 모바일 환경에서 JNI를 활용한 네이티브 언어로 컴파일하기 위한 과정과 프로그래밍 앱을 구현하는 방법을 설명한다. 4장에서는 제안 시스템과 Windows 환경에서의 성능 비교를 수행한다. 5장에서는 고찰을 수행한 후, 6장에서 결론짓는다.

2. 관련 연구

2.1 Scheme

Scheme[2]은 1975년 MIT의 Guy Steele과 Gerald Sussman이 함께 개발한 LISP 계열 함수형 언어이다. Scheme은 입력 변수로 다른 함수를 사용하거나 결과값 반환이 가능한 고차 함수를 지원한다. 고차 함수를 통해 하나의 함수로 다수의 기능을 수행하고, 함수 합성을 통해 다수의 함수로 하나의 복잡한 기능의 함수를 간단히 만들 수 있다.

2.2 RScheme

RScheme[3]은 Scheme의 객체지향 확장형 오픈소스이며 R^4RS(Revised^4 Report on Scheme)와

Dylan의 개념을 합친 것이다. RScheme은 REPL (read-eval-print loop) 환경을 제공하여 소스코드 실행 결과를 바로 확인할 수 있다. RScheme으로 작성된 코드는 일반 C 컴파일러로 컴파일되어 기계어 코드를 생성할 수 있다. RScheme의 런타임 시에는 가상머신에 의해 해석되는 바이트 코드로 컴파일되어 컴파일 속도가 빠르고 코드 크기가 줄어든다.

2.3 JNI

JNI(Java Native Interface)[4]는 C나 C++ 등을 이용하여 네이티브 언어로 작성된 계층과 Java로 작성된 계층이 상호작용할 수 있게 하는 인터페이스이다. 먼저 컴파일 환경처럼 실행 속도와 처리 속도가 중요한 부분을 네이티브 언어로 작성한다. 다음으로 앱 제작을 위한 안드로이드 프레임워크 환경은 Java를 사용해 구현한다. 이를 JNI로 연결하면 각각의 언어의 장점을 활용하여 성능을 높일 수 있다.

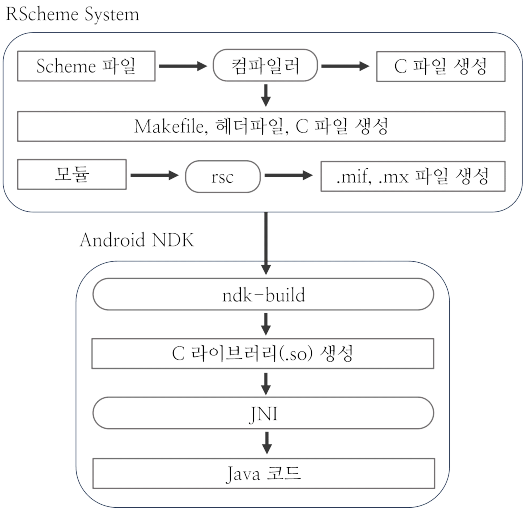
2.4 NDK

NDK(Native Development Kit)[5]는 안드로이드 프레임워크 환경에서 C 및 C++를 통해 작성된 네이티브 라이브러리를 컴파일할 수 있는 도구 모음이다. JNI를 활용한 NDK를 이용하면 기기의 성능을 최대한 활용하여 짧은 지연 시간 내에 프로그램 실행이 가능하므로 컴퓨팅 집약적 앱에 유리하다. 또한, 실시간 대량 데이터를 처리하는 상황에서 속도를 향상시킬 수 있다.

3. 설계 및 구현

본 논문에서 제안하는 시스템은 RScheme의 REPL 환경을 사용한다. C로 컴파일할 수 있는 RScheme을 모바일 환경에서도 사용하기 위해, 안드로이드 프레임워크 환경에서 C를 컴파일할 수 있도록 설계하였다. 그리고 JNI를 사용하기 위해 안드로이드 NDK를 통해 Java 애플리케이션과 연결할 수 있는 C 라이브러리를 작성하였다. 그림 1은 제안 시스템 구조를 나타낸다.

RScheme 시스템에서 컴파일러는 C 코드를 생성하거나 C 코드 및 바이트 코드의 혼합 형태를 생성한다. 바이트 코드로 컴파일하는 경우, rsc는 각 모듈에 대해 모듈 이미지 파일(.mif)과 모듈 인덱스 파일(.mx)을 생성한다. 다음으로 모듈 전체에 대해 컴파일러는 Makefile, 두 개의 헤더파일과 C 파일을 생성한다.



(그림 1) 제안 시스템 구조

foo 모듈을 생성하는 경우, Makefile은 foo 모듈의 객체 코드를 생성하는 데 사용한다. 다음으로 모듈에서 내보낸 Symbol에 접근하기 위해 클라이언트 모듈이 사용하는 헤더파일 생성과 RScheme 시스템에 연결하기 위한 링크 파일인 foo_l.c를 생성한다. 모듈과 관련된 파일 이외에도 컴파일러는 각 Scheme 파일(.scm)을 하나의 C 파일로 생성하는데, 이 파일은 컴파일러가 C 언어로 변환 시 사용한다.

Java 클래스에서 C 코드 연결을 위해 네이티브 메서드를 선언하면 Java 컴파일러는 클래스 파일의 메서드 정보에 네이티브 메서드 정보를 입력한다[6]. Java 클래스로 입력받은 RScheme 코드를 네이티브 코드로 전송하는 과정으로 공유 라이브러리에서 네이티브 코드 적재를 위해 정적 클래스 초기화 프로그램에서 System.loadLibrary() 메서드를 사용한다.

C 코드에서 Java 클래스의 네이티브 메서드의 정보를 기반으로 JNIEXPORT <return type> JNICALL Java_<class_name>_<method_name> (JNIEnv * env, <jobject or jclass>, <arg_type_list>)와 같이 구성한다[7]. RScheme 코드를 전송받고 컴파일하여 결과값을 반환하는 과정에서 GetStringUTFChars 메서드는 jstring을 UTF-8 형식으로 인코딩한 문자 배열로 바꾸고, 그 배열의 포인터를 반환해 준다.

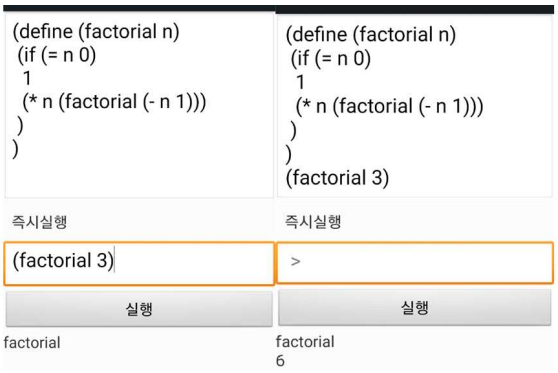
프로젝트 빌드를 위해 ndk-build 스크립트를 사용한다. NDK로 정적 라이브러리와 C 코드를 빌드하여 동적 라이브러리를 생성하는 환경 구성[8] 과정은 다음과 같다.

1) NDK 버전: android-ndk-r10b
2) jni 폴더에 네이티브 코드와 관련 파일 저장

3) Android.mk, Application.mk 파일 생성
4) 동적 라이브러리(.so) 생성

단계 2에서 프로젝트의 app/src/main 경로 아래에 jni 폴더를 생성하고 해당 폴더에 정적 라이브러리(.a), rsc, 모듈 이미지 파일(.mif), 모듈 인덱스 파일(.mx) 등 네이티브 코드와 관련된 파일을 저장한다. 단계 3에서는 app/src/main/jni 경로에 빌드 시스템의 네이티브 모듈을 설명하는 Android.mk 파일과 빌드 시 프로젝트 전체 설정을 하는 Application.mk 파일을 생성한다. 단계 4에서는 android-ndk-r10b 폴더 내부 ndk-build.cmd 파일을 통해 프로젝트를 빌드하면 app/src/main 경로에 libr 폴더와 NDK 플랫폼에 맞는 동적 라이브러리(.so)가 생성된다. 해당 동적 라이브러리는 System.loadLibrary() 메소드 실행 시 매개변수로 사용된다.

그림 2는 제안 시스템을 통해 RScheme 코드를 실행한 결과를 나타낸 것이다. RScheme으로 작성된 코드를 입력하면 C 코드로 컴파일하여 결과값이 출력된다. RScheme에 대한 결과값은 인터프리터 형식으로 한 줄씩 출력된다. 좌측 화면은 실행 버튼을 활성화하기 전 코드를 입력한 것을 나타내며, 우측 화면은 실행 버튼을 눌러 해당 코드에 대한 결과값을 출력한 것이다.



(그림 2) 제안 시스템 사용 예

4. 실험 및 평가

4.1 실험 환경 구성

환경에 따른 인터프리터 성능을 평가하기 위해 모바일, 컴퓨터 환경에 대해 피보나치 수열을 계산하는 함수를 작성한 후 실행 시간을 측정하여 인터프리터 성능을 파악하였다. 실험을 위해 RScheme 코드를 C 언어로 컴파일이 가능한 RScheme 언어의 인터프리터 실행 환경을 표 1과 같이 구성하였다.

<표 1> 실험 환경 구성

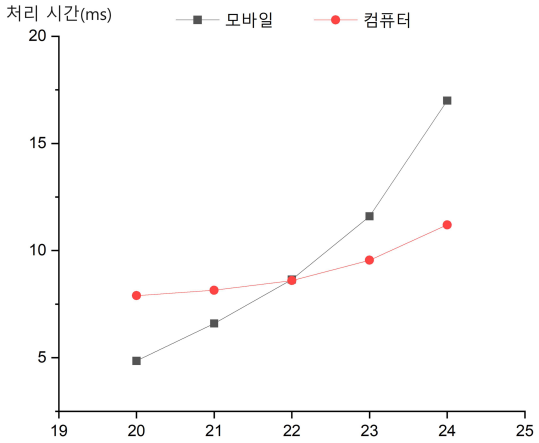
구분	모바일	컴퓨터
제품	Galaxy Z Fold2 5G	갤럭시북3 울트라
운영체제	Android 10	Windows 11 Home
RAM(GB)	12	32
프로세서	Snapdragon 865+	Intel Core i9 13900H

4.2 환경에 따른 인터프리터 성능 평가

이 절에서는 앞 절에서 구성한 환경에서 실제로 코드를 실행한 결과를 분석한다. 실험은 정확성을 위해 각 재귀 함수를 20회 실험하였고, 결과값은 20회의 인터프리터 처리 시간에 대한 평균값이다. 표 2는 실험 결과를 나타낸 것이다. 또한, 그림 3은 표 2를 바탕으로 실험 결과를 그래프로 나타낸 것이다.

<표 2> 환경에 따른 인터프리터 처리 시간(ms)

구분	모바일(M)	컴퓨터(C)	M/C 백분율
피보나치 20	4.85	7.90	0.61
피보나치 21	6.60	8.15	0.80
피보나치 22	8.65	8.60	1.00
피보나치 23	11.06	9.55	1.15
피보나치 24	17.00	11.20	1.51



(그림 3) 환경에 따른 인터프리터 처리 시간(ms)

실험 결과 피보나치 22까지는 모바일 환경이 컴퓨터 환경보다 실행 시간이 짧은 것을 확인할 수 있다. 하지만 피보나치 22 이상부터는 컴퓨터 환경이 모바일 환경보다 나은 성능을 보였다. 그리고 피보나치 24까지 계산한 결과를 모바일(M)/컴퓨터(C) 백분율 기하 평균으로 나타내었을 때 0.96으로 모바일과 컴퓨터 성능 차이가 크지 않았다. 따라서 모바일 환경이 프로그래밍 실습에 큰 제약이 되지 않는다는 것

을 알 수 있다.

5. 고찰

이 논문에서 제안하는 시스템은 앱 내부에서 RScheme 컴파일이 가능한 라이브러리를 구성하여 서버 없이 단독으로 동작하도록 구현하였다. 그리고 코드 실행 시 인터프리터 형식으로 결과값을 확인할 수 있다. 또한, 프로그램 수정과 디버깅 과정이 간단하여 프로그래밍 실습에 유용하다.

하지만 모바일 환경의 특성상 프로그래밍 실습 시 학생이 간편하게 시스템을 이용하는 방법을 제공하는 것이 필요하다. Scheme은 람다(lambda) 함수를 사용하여 정의하며 괄호를 통해 표현한다. 괄호가 중첩되면 여는 괄호와 닫는 괄호 개수를 착각할 수 있다. 따라서 코드 작성 시 중괄호를 자동 완성하는 기능을 제공할 필요가 있다.

다음으로 Scheme 언어의 난이도 문제가 있다. 보편적으로는 프로그래밍을 처음 배울 때 스크래치나 엔트리와 같은 블록 프로그래밍 언어 또는 Python이나 C와 같은 언어를 배우게 된다. 이렇게 보편적으로 습득하는 언어와의 비교를 통한 이해를 돕기 위해 Scheme 코드 작성 시 다른 언어로 번역하는 방법에 관한 연구가 필요하다.

제안 시스템에서 사용된 RScheme은 R^4RS 표준을 지원한다. R^5RS 표준 이후에서는 SRFI(Scheme Request for Implementation)을 통해 필요한 기능을 요청할 수 있다. R^6RS 표준에서는 SRFI 라이브러리 및 R^5RS 표준과의 호환성 문제로 교육용으로는 적합하지 않다는 의견이 있다[9]. R^7RS 표준은 이러한 점을 보완하고자 교육용과 업무용을 나누어 표준을 제정하였다. 교육용으로 다양한 기능을 구현하고 실습하기 위해 SRFI를 지원할 수 있는 표준 지원과 라이브러리의 호환성 해결에 대한 연구가 필요하다.

6. 결론

본 논문에서는 C 코드를 컴파일할 수 있는 RScheme 시스템을 활용하여 모바일 프로그래밍 앱을 설계하고 구현하였다. RScheme 시스템의 컴파일을 위해 필요한 파일 및 과정을 확인하고 이를 모바일 앱에 적용하고 JNI를 통해 Java API가 C 네이티브 코드를 실행하도록 하였다. 구성된 JNI는 안드로이드 NDK로 프로젝트를 빌드하여 동적 라이브러리를 생성하고 전체적인 구현이 가능하게 하였다.

학생들의 효율적이고 간소화된 프로그래밍 실습을 위해 인터넷을 통해 서버에 접속할 필요가 없는 프로그래밍 앱을 구현하였다. 또한, 우리는 환경에 따른 인터프리터 성능 평가 실험을 통해 모바일 환경도 일반 컴퓨터 환경에 못지않은 성능을 발휘한다는 것을 확인하였다. Scheme의 고차 함수 특성과 다중 패러다임 언어 특성을 통해 학생들이 다양한 관점의 문제 해결 능력을 기를 수 있을 것으로 기대된다.

향후 연구로 코드 작성 시 중괄호를 자동 완성하는 등 사용자 편의 기능을 제공할 예정이다. 또한, Scheme 코드 작성 시 Python 등 일반적으로 널리 사용하는 코드로 즉시 번역하는 방안, SRFI를 지원할 수 있는 표준 지원과 라이브러리 호환성 해결 방안 등에 대한 연구도 진행할 예정이다.

참고문헌

[1] 김병호, “앱인벤터 오픈소스 수정·구현”, 한국정보통신학회논문지, Vol.22, No.2, pp.221 - 226, 한국정보통신학회, 2018

[2] Gerald Jay Sussman, “Scheme: A Interpreter for Extended Lambda Calculus,” Higher-Order and Symbolic Computation, Vol.11 No.4 pp.405 - 439, Kluwer Academic Publishers, 1998

[3] Donovan Kolbly, RScheme: Design and Implementation, https://www.rscheme.org, 2023. 09 접근

[4] 손기철, 안드로이드 NDK를 이용한 어플리케이션 속도 향상 예측기, 전북대학교 대학원 전자공학과 석사학위논문, 2012

[5] Google for Developers, Android NDK, https://developer.android.com/ndk, 2023. 09 접근

[6] 이창환, 오세만, “JPP(JNI 전처리기)의 설계 및 구현”, 정보처리학회논문지, Vol.9, No.1, pp.129 - 136, 한국정보처리학회, 2002

[7] 이창환, 자바와 C 언어 결합을 위한 자바 전처리기, 동국대학교 대학원 컴퓨터공학과 박사학위논문, 2002

[8] 고은별, “안드로이드 NDK(Native Development Kit)를 이용한 3D 큐브 게임 이식 및 구현”, 디지털콘텐츠학회논문지, Vol.14, No.3, pp.381 - 390, 한국디지털콘텐츠학회, 2013

[9] Kato Takashi, “Implementing R7RS on an R6RS Scheme system,” 2014 Scheme And Functional Programming Workshop, Scheme ‘14, Washington DC, 2014, pp.1 - 7