The Transactions of the Korea Information Processing Society
TKIPS Vol.13, No.3, pp.122~129

eISSN: 3022-7011
https://doi.org/10.3745/TKIPS.2024.13.3.122

Implementation of a Scheme Mobile Programming Application and Performance Evaluation of the Interpreter

Dongseob Kim† · Sangkon Han†† · Gyun Woo†††

ABSTRACT

Though programming education has been stressed recently, the elementary, middle, and high school students are having trouble in programming education. Most programming environments for them are based on block coding, which hinders them from moving to text coding. The traditional PC environment has also troubles such as maintenance problems. In this situation, mobile applications can be considered as alternative programming environments. This paper addresses the design and implementation of coding applications for mobile devices. As a prototype, a Scheme interpreter mobile app is proposed, where Scheme is used for programming courses at MIT since it supports multi-paradigm programming. The implementation has the advantage of not consuming the network bandwidth since it is designed as a standalone application. According to the benchmark result, the execution time on Android devices, relative to that on a desktop, was 131% for the Derivative and 157% for the Tak. Further, the maximum execution times for the benchmark programs on the Android device were 19.8ms for the Derivative and 131.15ms for the Tak benchmark. This confirms that when selecting an Android device for programming education purposes, there are no significant constraints for training.

Keywords : Scheme, RScheme, Programming Mobile App, JNI, Benchmarking

Scheme 프로그래밍 모바일 앱 구현과 인터프리터 성능 평가

김 동 섭† · 한 상 곤†† · 우 균†††

요 약

최근 프로그래밍 교육의 중요성이 강조되고 있지만, 초·중·고교 학생들은 프로그래밍 교육에 어려움을 겪고 있다. 대부분의 프로그래밍 환경이 블록 코딩을 바탕으로 이루어지고 있는데 이는 텍스트 코딩으로의 이행에 방해가 된다. 전통적인 PC 환경도 유지 관리 문제 등 어려움이 있다. 이러한 상황에서 모바일 앱은 대안적 프로그램 교육환경으로 생각해 볼 수 있다. 이 논문에서는 이동형 기기에서 프로그램을 작성할 수 있는 모바일 앱 설계하고 구현하였다. 첫 사례로 Scheme 인터프리터 모바일 앱을 구현하였는데, Scheme은 다중 패러다임 프로그래밍을 지원하는 교육용 언어로 MIT의 프로그래밍 교과에 사용되고 있다. 구현된 앱은 독립형 앱으로 설계되어 네트워크를 사용하지 않아도 된다는 장점이 있다. 벤치마크 수행 결과, PC 수행 시간에 대한 안드로이드 기기 수행 시간은 Derivative 벤치마크 131%와 Tak 벤치마크 157%로 나타났다. 또한, 안드로이드 기기에서 벤치마크 프로그램의 수행 시간 최댓값은 Derivative 벤치마크 19.8ms, Tak 벤치마크 131.15ms로 나타났다. 이는 안드로이드 기기를 프로그래밍 교육용으로 선택 시 실습에 큰 제약이 되지 않음을 나타낸다.

키워드 : Scheme, RScheme, 프로그래밍 모바일 앱, JNI, 벤치마크

1. 서 론

산업 현장의 수요에 비해 디지털 인력이 부족한 상황에서 초·중등 학생들에게 코딩 교육 활성화를 시도하고 있지만, 교육 여건은 녹록지 않은 상황이다. 프로그래밍 환경의 어려운 상황은 차치하고라도 프로그래밍 교육용 언어가 블록 프로그래밍 언어에 편중되어 있다. 엔트리, 스크래치, 엘리스, 앱인벤터 등 블록 언어는 프로그래밍 개념에 집중할 수 있다는 장점이 있지만[1], 이후 텍스트 언어로의 확장이 어렵다는 단점이 있다.

또한, 초·중·고교 학생들은 스마트폰 과의존으로 오히려 컴퓨터 활용이 익숙하지 않은 학생들도 늘어나고 있다. 컴퓨터도 주로 게임에 활용하고 있으며, 학력 단계를 나아갈수록 소프트웨어 활용 능력 격차가 벌어지는 상황이 발생한다. 따라

※ 이 과제는 2021학년도부산대학교교수국외장기파견지원비에 의하여 연구되었음.
※ 이 논문은 2023년 ACK 2023의 우수논문으로 "Scheme 프로그래밍 모바일 앱 설계 및 구현"의 제목으로 발표된 논문을 확장한 것임.
† 준 회 원 : 부산대학교 정보융합공학과 석사과정
†† 준 회 원 : 부산대학교 정보융합공학과 박사과정
††† 종신회원 : 부산대학교 정보컴퓨터공학부 교수
Manuscript Received : January 17, 2024
Accepted : February 26, 2024
* Corresponding Author : Gyun Woo(woogyun@pusan.ac.kr)

서 컴퓨터 활용 능력이 저조한 학생도 컴퓨터 사고력을 향상할 수 있도록 익숙한 스마트폰으로 소프트웨어 교육을 진행할 방안이 필요하다.

스마트 기기가 프로그래밍 실습에 더 적합한 측면도 있다. 일반 실습실에서 단체로 프로그램 교육이 진행되는 경우 교육환경 및 실습 기회가 제한적이라는 연구 결과가 있는데[2], 이는 스마트 기기의 실습 활용 가능성을 보여준다. 모바일 앱을 활용하면, 언어 처리기와 더불어 편집기 일체가 제공되므로 교육환경을 손쉽게 구성하고 통제할 수 있다. 또, 자신의 모바일 기기를 사용해서 실습을 진행할 수 있으므로 공간적, 시간적 제한 사항에서 벗어난다. 게다가 대부분의 프로그래밍 교육이 문제 해결을 중심으로 진행되기 때문에 학습자의 상황에 맞춰서 언제든 교육을 계속해서 진행할 수 있다.

모바일 환경에서 프로그래밍할 경우, PC 환경과 달리 작은 화면과 불편한 입력 기능으로 개발 및 디버깅이 어렵다는 단점이 있다. 그러나 2000년대 초·중반 이후 스마트폰과 함께 자란 학생들의 PC보다 스마트폰 키보드가 편하다는 의견이 나오고 있다. 2019년 연구[3]에 따르면 일반인들의 평균 모바일 타자 속도가 일반 키보드 타자 속도의 70%에 달한다. 소프트웨어 교육의 대상인 2000년대에 태어난 학생들에게 모바일 키보드의 사용이 편하며 고성능 작업이 아닌 교육을 위한 작업에선 모바일 환경이 큰 제한이 되지 않는다.

모바일 환경에 비해 PC 환경은 큰 부피와 무게로 설치를 위한 장소의 필요성과 코딩 환경 구축에 어려움이 있으므로, 코로나 19 확산과 같이 등교하지 않고 온라인으로 수업하거나 과제를 제출하는 상황을 위해 이동성이 뛰어나고 코딩 환경 구축이 쉬운 프로그래밍 교육환경이 필요하다. 이러한 상황에 프로그래밍 모바일 앱은 장소의 제약성 해소와 장비의 경량화가 가능하며 컴퓨터 활용 능력이 부족하더라도 소프트웨어 활용 능력을 향상할 수 있다.

현재 스마트폰에서 코딩할 수 있는 모바일 앱은 다양한 프로그래밍 언어로 출시되어 있다. 효율적인 환경적인 방안에 비해 대상 프로그래밍 언어의 복잡성과 낮은 접근성을 해소하고자 다중 패러다임 프로그래밍을 지원하는 교육용 언어인 LISP 파생어의 Scheme을 이용하여 연구를 진행한다.

Haskell이나 OCaml, Scheme, Eralng 등과 같은 함수형 언어를 교육에 활용할 때의 이점은 아래와 같다.

1) 인수와 함수 적용 등 기초적인 프로그래밍 개념을 명확하게 이해할 수 있다.
2) 함수형 언어의 간명하고 높은 수준의 표현력을 활용해서 난이도 있는 프로그래밍 문제를 수업 초반에 해결할 수 있다.
3) 부작용(side-effect)이 없는 함수를 이용하여 수학의 함수 개념을 프로그래밍 영역으로 확장할 수 있다.

이런 장점 덕분에 함수형 언어를 기반으로 한 프로그래밍 언어를 활용해서 교육을 진행하게 되면, 문제 해결력 향상을 위한 교육에 좀 더 집중할 수 있다[4].

교육용 프로그래밍 언어는 단순하면서도 표현력이 뛰어나야 한다. 현재 주로 사용되는 교육용 블록 기반 언어는 일반적인 텍스트 기반 언어에 비해 표현력이 제한되며 간단한 프로그램에는 적합하나 규모가 커질수록 가독성이 떨어진다. 블록 기반 언어를 학습하더라도 텍스트 기반 언어에서의 프로그래밍에 즉시 적용되지 않을 수 있다. 교육용 텍스트 기반 언어로 주로 사용되는 C 언어의 경우, 하드웨어와의 직접적인 상호작용을 중점으로 둔 저수준 언어로 표현력을 제한하는 문법 및 의미적 특성으로 코드가 상대적으로 복잡하다. 학생들은 C 언어를 학습하기 위해 문법 숙달, 의미적 제약에 대한 대응을 주요하게 생각하여 프로그래밍 교육의 주된 목적을 잃을 수 있다.

Scheme 언어는 Lisp 언어처럼 괄호 내부 구문을 사용하여 단순하고 가능한 일반적이고 적은 수의 추상화 메커니즘을 기반으로 설계하여 간결하고 뛰어나다. 학생들은 Scheme 언어 학습을 통해 추상화를 통한 복잡성 제어, 알고리즘 설계, 알고리즘 효율성의 주요 문제에 집중할 수 있어 C 언어와 대비하여 프로그래밍 교육의 핵심적인 부분을 수행할 수 있다[5]. 캘리포니아 대학교 버클리(U. of California, Berkeley), 브렌다이스 대학교(Brandeis Univ.), 컬럼비아 대학교(Columbia Univ.), 존스 홉킨스 대학교(Johns Hopkins Univ.), MIT(매사추세츠 공과대학), 노스이스턴대(Northeastern Univ.), 노스웨스턴대(Northwestern Univ.), 오벌린 대학(Oberlin College), 펜실베이니아 주립대학교(Penn. State Univ.), 프린스턴 대학교(Princeton Univ.), 라이스 대학교(Rice Univ.), 스탠포드 대학교(Stanford Univ.) 등 유수의 대학교에서 프로그래밍 교육을 위해 Scheme 언어를 사용하며 프로그래밍 교육 입문 과정에서도 다수의 대학교가 사용하고 있다.

본 논문에서는 모바일 환경에서 네트워크 접속 없이 Scheme 프로그래밍을 할 수 있는 모바일 앱을 제안한다. 이 앱을 통해 장소에 제약받지 않고 간편하게 Scheme을 접할 수 있다. 제안한 앱은 가독성을 위해 구문 강조(syntax highlighting) 기능을 제공하며 학습자의 편의를 위해 괄호 유효성 검사(parentheses validation), 괄호 생성, 소스 코드 저장 및 불러오기 등의 기능을 제공한다.

프로그래밍 교육용에 적합한지 확인하고자, 제안한 앱과 같은 기능의 프로그램을 안드로이드 기기와 PC 환경에서 각각 구현하여 비교 실험을 수행한다. 비교 실험을 위해 사용한 소스 코드는 연산 성능을 시험하고자 벤치마크를 사용한다.

본 논문의 구성은 다음과 같다. 먼저 2절에서는 본 연구와 관련된 연구인 Scheme과 JNI, NDK, 벤치마크 프로그램을 살펴본다. 3절에서는 안드로이드 환경에서 RScheme을 지원하기 위한 시스템을 설계하고, 4절에서는 JNI를 활용한 구현 방법을 설명한다. 5절에서는 제안 시스템과 PC 환경에서의 성능 비교를 수행한다. 6절에서 본 연구의 한계와 발전 방향을 살펴본 후, 7절에서 결론을 맺는다.

2. 관련 연구

2.1 Scheme 및 RScheme

Scheme[6]은 교육용 LISP 변종 언어로서 1975년 MIT에서 Guy Steele과 Gerald Sussman이 함께 개발하였다. Scheme은 고차 함수를 지원하며 다른 프로그래밍 언어와 비교해 최소한의 기본 명령만 기계어로 정의가 되어있어 크기가 작고, 다양한 기능을 요구하는 함수의 정의가 간편하다.

RScheme[7]은 Scheme의 객체지향 형태로 확장한 공개 소프트웨어로서 R^4RS(Revised^4 Report on Scheme) 표준과 Dylan을 혼합한 형태이다. RScheme도 여느 LISP 구현과 마찬가지로 REPL(read-eval-print loop)을 통해 작성된 코드의 결과를 즉시 확인할 수 있다. RScheme으로 작성된 코드는 C로 컴파일할 수 있으며 C는 일반 C 컴파일러로 컴파일하여 기계어 코드를 생성할 수 있다.

2.2 JNI 및 NDK

JNI(Java Native Interface)[8]는 C나 C++로 작성된 네이티브 코드와 안드로이드가 컴파일하는 Java로 작성된 바이트 코드가 상호작용하는 방법을 정의한다. JNI를 통해 네이티브 라이브러리를 활용할 수 있고 프로세스의 성능 향상할 수 있다.

NDK(Native Development Kit)[9]는 C나 C++ 언어를 사용하여 앱 일부를 구현할 수 있도록 하는 도구 모음이다. NDK를 통해 C 또는 C++ 라이브러리를 재사용할 수 있고 계산 집약적인 앱의 최적화와 성능 향상이 가능하다.

2.3 Derivative 및 Tak 벤치마크

Derivative 벤치마크는 함수에 대해 일반적으로 데이터 표현을 S-표현식을 사용하는 간단한 도함수를 수행한다. 해당 벤치마크는 DERIV를 주요 함수로 사용하며 다수의 CONS 및 함수 호출을 수행한다[10].

이 논문에서는 $3x^2+ax^2+bx+5$ 함수에 대해 주어진 S-표현식과 연산자 테이블을 기반으로 함수의 기호 미분을 수행하고 n번 반복하여 성능을 평가한다.

Tak 함수는 다케우치 이쿠오(竹内郁雄)의 이름을 따서 명명된 재귀 함수로서, Equation 1과 같이 정의된다.

$$\tau(x,y,z)=\begin{cases}\tau(\tau(x-1,y,z),\tau(y-1,z,x),\tau(z-1,x,y)) & \text{if } y<z \\ z & \text{o.w.}\end{cases} \quad (1)$$

Tak 함수는 재귀에 최적화된 프로그래밍 언어의 연산 능력이나 성능을 판별하기 위해서 사용되고 있다[11]. Tak 함수는 함수를 호출하는 속도를 중점적으로 평가한다. 최근의 프로그램이나 모바일 애플리케이션은 다양한 내장 라이브러리나 외부 API를 호출하는 데 많은 자원을 할당하고 소비하는 경향이 있지만, Tak 함수를 활용한 성능 측정 방식은 연산 성능을 기준으로 검증하는 데 유용하다.

3. 설 계

본 논문에서는 C로 컴파일되는 RScheme 인터프리터를 모바일 환경에서 구현하고자 모바일 앱을 제작할 수 있는 안드로이드 환경에서 JNI를 사용하여 C를 컴파일할 수 있도록 시스템을 구성한다. 전체 앱 구조는 Evilbinary의 연구[12]를 따랐다. Fig. 1은 제안 시스템 구조의 DFD(data flow diagram)를 나타낸다.

RScheme 시스템의 라이브러리(`.a`)를 `ndk-build`를 통해 RScheme 동적 라이브러리(`.so`)로 생성한다. Scheme 코드를 입력하면 Java의 `Rs` 객체를 통해 C 코드와 연결하고 `eval` 메소드를 통해 인터프리터를 수행한다. 이 과정은 JNI를 통해 C 함수를 호출하는 형태[13]로 구현한다. 제안 시스템의 동작 과정을 순차 다이어그램(sequence diagram)으로 나타내면 Fig. 2와 같다.

프로그래밍 실습 시 복잡한 연산에 취약한 React Native보다 최적화되어 있는 안드로이드를 사용하였는데, 최적화된 안드로이드는 메모리도 Flutter나 React Native 환경에 비해 절반만 사용한다. RScheme은 객체지향 접근 방식으로 코드의 재사용성과 유지보수성에 유리하다. RScheme의 또 다른 장점으로는 일반 C 컴파일러로 컴파일하여 기계어 코드를 생성할 수 있다는 점을 들 수 있는데, 그러므로 안드로이드의 JNI를 통한 상호작용을 통해 Scheme 인터프리터 앱을 구성하기 용이하다.

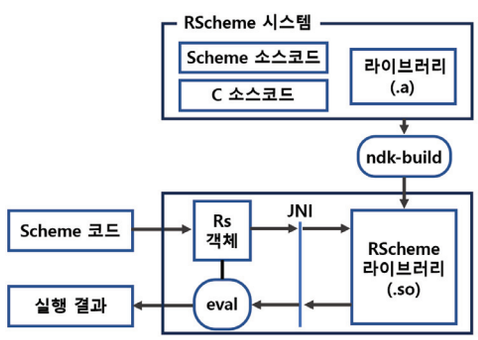

Fig. 1. The Structure of the Proposed System

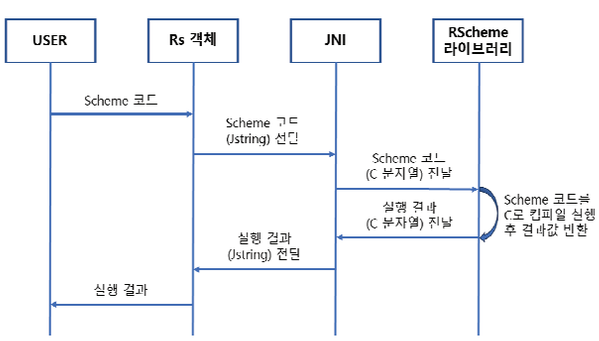

Fig. 2. The Sequence Diagram of a Typical Usage

원활한 프로그래밍 실습을 위해 코드를 읽고 분석하기 쉽게 하기 위한 기능도 구현하였다. 코드 가독성을 위해 구문 강조를 구현하였으며 코드 작성 시간 절감 및 오류 방지를 위해 괄호 유효성 검사와 괄호 생성 기능, 그 외 소스 코드 파일 저장 및 불러오기 기능 등을 설계하고 구현하였다.

4. 구 현

4.1 RScheme을 이용한 C 코드 생성

RScheme 시스템은 C 코드를 생성함으로써 Scheme 코드를 수행할 수 있도록 한다. RScheme에서 오프라인 모듈 컴파일러는 모듈별로 모듈 이미지 파일(module image file), 즉 `.mix` 파일을 생성한다. 링크 정보는 별도의 색인 파일(`.mx`)로 생성되는데, 색인 파일에는 동적 라이브러리 파일(`.so`)의 진입 지점(entry point) 등이 포함되어 있다.

모듈 이미지 파일과 모듈 색인 파일은 객체처럼 취급되는데, 이들 모듈에 포함된 심볼 정보는 모듈 컴파일러 `rsc`에 의해 헤더 파일(`.h`)로 생성된다. 모듈 링크 파일은 별도의 파일로 생성되는데 예컨대 모듈 이름이 `bingo`라면 `rsc`는 링크 정보를 `bingo_l.c`로 생성한다. Scheme 파일(`.scm`)은 별도의 C 파일로 생성된다.

4.2 네이티브 메소드의 C 코드 구현

Java 클래스에서 선언된 네이티브 메소드는 네이티브 라이브러리로 구현되어야 하는데, RScheme과 연결하기 위해 C 코드 형태로 네이티브 메소드를 구현한다[14]. 네이티브 메소드 구현 형태는 Fig. 3과 같다.

Fig. 3에서 볼 수 있는 것처럼, 반환 타입 이름과 함수 이름 앞에 각각 `JNIEXPORT`, `JNICALL`이라는 지시자를 명시한다. 함수명도 특별한 형태인데 `Java_<클래스명>_<메소드명>` 형태이다[15]. 매개변수는 `env`와 `obj`, `str`인데, `env`는 `JNIEnv` 타입이며 `obj`는 `jobject`나 `jclass` 타입, `str`은 `jstring` 타입이다. 여기서는 메소드 `eval`의 인수가 문자열 하나뿐이므로 매개변수가 `str` 하나이지만, 인수가 여러 개라면 매개변수를 추가해야 한다.

앱에서 입력된 RScheme 코드는 Java의 `String` 타입으로서 `eval_str`로 전달된다. `rs_eval`을 통해 이 코드를 인터프

```
JNIEXPORT jstring JNICALL Java_org_evilbinary_rs_Rs_eva
l(JNIEnv *env, jobject obj, jstring str){
 jstring ret;
 char *eval_str = (*env)->GetStringUTFChars(env, str, NULL);
 LOGI("eval=%s",eval_str);
 result=rs_eval(eval_str);
 LOGI("eval end.");
 ret=strToJstring(env,unicode_string_text(result));
 (*env)->ReleaseStringUTFChars(env,str, eval_str);
 return ret;
}
```

Fig. 3. Configuring Native Methods in the C Code

```
public class Rs {
…
public void init(String homePath, String[] argv) {
 int len = this.args.length;
 String[] newArgs = new String[argv.length + 1];
 System.arraycopy(this.args, 0, newArgs, 0, len);
 System.arraycopy(argv, 0, newArgs, len, argv.length);
 this.args = newArgs;
 this.homePath = homePath;
 System.loadLibrary("myrs");
 init();
}
public native void init();
public native String eval(String exp);
public native void exit();
}
```

Fig. 4. Configuring Native Methods in the Java Class

리터에서 계산하고 계산 결과를 `result`에 저장한다. `result` 값은 Java의 `String` 타입으로 변환되어 함수 결과로 반환된다. 반환 전에 `ReleaseStringUTFChars`를 사용하여 문자열 `eval_str`에 사용된 메모리를 해제한다.

4.3 Java 클래스의 네이티브 메소드 인터페이스

RScheme 컴파일러에서 생성한 C 코드에 대한 라이브러리를 Java 프로그램에서 사용하기 위해서는 네이티브 제한자(native modifier)가 적용된 메소드로 선언되어야 한다. Java 클래스에서 네이티브 메소드를 선언한 부분[12]을 요약하면 Fig. 4와 같다.

Fig. 4에서 볼 수 있는 것처럼 선언된 네이티브 메소드는 `init`과 `eval`, `exit` 세 가지인데, 이 중에서 `eval`이 Scheme 코드를 수행하는 주요 메소드다. 동적 라이브러리는 `eval` 수행 전에 초기화되어야 하는데, Fig. 4의 `init` 메소드를 보면 `System.loadLibrary`를 통해 동적 라이브러리 `myrs`를 적재하는 것을 볼 수 있다.

4.4 NDK 빌드

NDK는 메이크(`make`) 기반 빌드 시스템을 사용하는 프로젝트를 빌드할 수 있도록 하는 도구로서 `ndk-build` 스크립트를 이용한다. NDK를 이용하여 동적 라이브러리를 생성하는 과정[16]은 다음과 같다.

1) `jni` 폴더에 네이티브 코드 및 관련 파일 저장
2) 메이크 파일(`Android.mk`, `Application.mk`) 생성
3) 동적 라이브러리(`.so`) 생성

이 논문에서 사용한 NDK 버전은 android-ndk-r10b이다.

단계 2에서 애플리케이션 프로젝트 디렉터리의 `jni` 폴더 아래에 RScheme 시스템의 라이브러리와 같이 네이티브 코드와 관련된 파일을 저장하고, 단계 3에는 `jni` 폴더 아래에 네이티브 코드의 빌드 설정을 정의하는 `Android.mk` 파일과 네이티브 코드 빌드 환경과 관련된 전역 설정을 정의하는 `Application.mk` 파일을 생성한다. 단계 4에서는 네이티브 코

드 스크립트로 동적 라이브러리(.so)를 생성한다.

NDK 버전을 해당 버전으로 선택한 이유는 RScheme 시스템의 라이브러리(.a)에서 '__swbuf'의 기호를 사용하고 있기 때문인데, 안드로이드 NDK r10c 버전부터는 모든 아키텍처에서 해당 기호가 삭제되어 빌드할 수 없다. 따라서 NDK r10c의 하위 버전인 NDK r10b를 사용하여 빌드한다.

4.5 앱 구현 결과 및 부가 기능

본 논문에서 구현한 인터프리터에서 RScheme 코드가 수행되는 결과를 보면 Fig. 5와 같다. Fig. 5의 좌측 화면은 함수 정의 후 실행 전 상황이며, 우측 화면은 실행 후의 상황이다. Fig. 5의 우측 아래를 보면 (fac 3)의 실행 결과 6이 출력되어 있음을 볼 수 있다.

코드 작성 시 구문 강조를 적용하였을 때, 주관적으로 사용자들은 색상 코드가 읽기 쉽고 더 만족스럽다고 말한다. 분석에 따르면 프로그래밍 실습에서 색상 코드를 읽을 때 더 많은 어려움을 겪지 않는다[17]. 구문 강조를 구현하기 위해 적용이 필요한 키워드에 대해 색상을 배정하고 Edittext의 TextWatcher를 통해 실시간으로 코드 작성 이벤트를 처리한다.

괄호 유효성 검사를 통해 괄호 불일치 오류를 줄이고 괄호 생성 버튼을 통해 코드 작성 시간을 줄일 수 있다. 괄호 유효성 검사를 구현하기 위해 여닫는 괄호에 대한 스택 처리를 수행한다. 작성한 코드는 내부 저장소에 저장하고 불러올 수 있도록 기능을 구현하였으며 다른 앱에서 접근할 수 없도록 MODE_PRIVATE로 설정하여 코드 자체의 보안성을 부여한다. Fig. 6은 괄호 유효성 검사 기능을 보여준다.

Fig. 6의 좌측 화면은 함수 작성 시 나타나는 괄호 힌트를 보여준다. Fig. 6의 좌측 화면을 보면 닫는 괄호가 모자람을 볼 수 있는데, 닫는 괄호를 회색 힌트로 표시하고 있다. Fig. 6의 우측 화면을 보면, 닫는 괄호를 작성하지 않고 실행하여 발생하는 오류를 볼 수 있다.

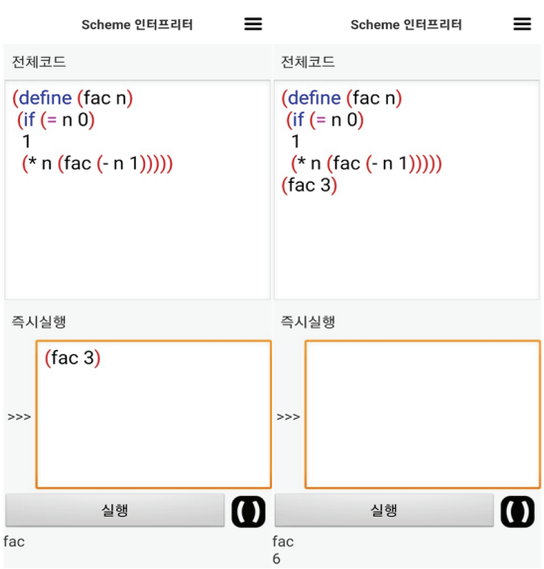

Fig. 5. The Screenshot of the Scheme Programming App

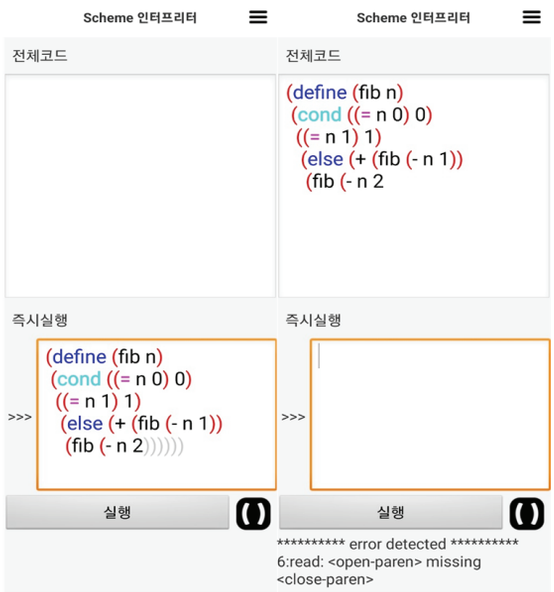

Fig. 6. The Features of the Scheme Programming App

5. 실험 및 평가

5.1 실험 설계 및 실험 환경 구성

프로그래밍 실습에서 인터프리터 성능 평가를 위해 안드로이드 기기와 PC 환경에서 벤치마크를 수행한다. 실험은 교육용에 초점을 두어 안드로이드 기기를 교육용으로 선택 시 문제 여부를 확인하고 적합성을 평가한다.

환경에 따른 인터프리터 성능을 평가하기 위해 안드로이드 기기와 PC에 RScheme 실행 환경을 구축하였다. 이 두 실행 환경에서 Derivative 및 Tak 벤치마크에 대한 실행 시간을 측정하여 인터프리터 성능을 파악하였다. 실험을 위해 구축한 인터프리터 실행 환경을 비교하여 나타내면 Table 1과 같다.

Table 1의 실행 환경을 보면 PC 환경이 안드로이드 환경에 비해 더 풍족한 메모리 용량을 지원하고 있다. 실제 현실적인 상황을 반영하기 위해 이처럼 실험 환경을 구축하였다.

5.2 환경에 따른 인터프리터 성능 평가

이 절에서는 5.1절에서 구성한 실험 환경에서 코드를 실제로 실행하여 실행 시간을 측정 · 분석한다. 정확성을 기하기

Table 1. The Experimental Environment

Category	Android Device	PC
Product	Galaxy Z Fold2 5G	Galaxy Book 3 Ultra
OS	Android 10	Windows 11 Home
RAM (GB)	12	32
Processor	Snapdragon 865+	Intel Core i9 13900H
Build System	ndk-build	GNU Make
Library Target Architecture	ARM	i386

위해 각 재귀 함수는 20회 반복 실험하여 평균을 측정하였다. Table 2의 실행 시간은 20회 처리 시간의 평균값이다.

벤치마크 프로그램 Derivative의 수행 시간을 나타내면 Table 2 및 Fig. 7과 같다.

Table 2 및 Fig. 7의 입력 데이터로는 $3x^2+ax^2+bx+5$ 함수를 사용하였는데, 이 함수에 대한 기호적 미분을 해당 수만큼 반복하여 수행하였다.

실험 결과 50,000 미만까지는 안드로이드 기기 환경의 실행 시간이 PC 환경보다 짧은 것을 확인할 수 있다. 하지만 50,000 이상부터는 PC 환경의 실행 시간이 안드로이드 기기보다 짧아지고 있음을 볼 수 있다. 90,000까지의 계산 결과 실행 시간 비율(안드로이드 기기/PC, Table 2의 A/P)의 평균(기하평균)은 131%로 산출되었다. 이 결과는 안드로이드 기기와 PC 성능 차이가 크지 않음을 나타낸다. 안드로이드 기기의 처리 시간은 90,000에서 19.8ms로서 연산에 많은 시간이 소요되지 않음을 볼 수 있다.

Table 2. The Run Time of the Derivative (ms)

Input	Android Device (A)	PC (P)	A/P Ratio (%)
10,000	7.70	11.45	93
30,000	10.45	12.60	114
50,000	13.55	13.45	134
70,000	16.60	15.05	158
90,000	19.80	16.09	178

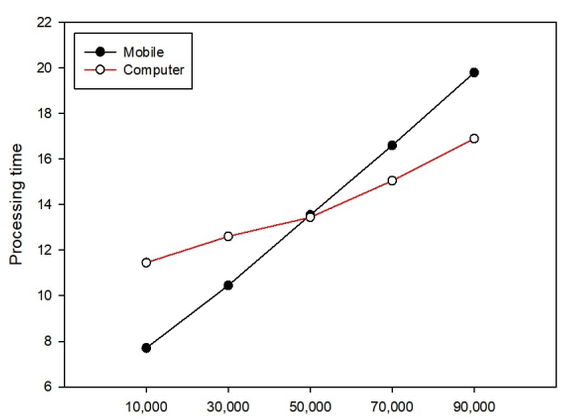

Fig. 7. The Run Time of the Derivative (ms)

Table 3. The Run Time of the Tak (ms)

Input	Android Device (A)	PC (P)	A/P Ratio (%)
18, 15, 9	3.25	8.60	52
18, 15, 8	5.15	9.60	70
18, 15, 7	12.25	11.10	116
18, 15, 6	38.65	20.10	278
18, 15, 5	131.15	50.60	814

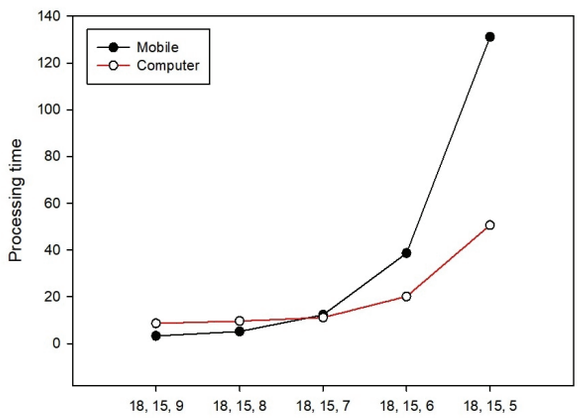

Fig. 8. The Run Time of the Tak (ms)

Tak 벤치마크에 대한 실험 결과를 나타내면 Table 3 및 Fig. 8과 같다.

Tak 함수는 한 호출에서 네 개의 재귀 호출을 수행하는 함수로서 함수 호출을 심하게 수행하는 함수이다. Table 3 및 Fig. 8에 나타낸 결과는 Tak의 세 개 인수 x, y, z 중에서 x와 y의 값은 일정하게 하고 z를 줄여가며 수행한 결과다.

실험 결과 z = 7 초과까지는 안드로이드 기기 환경이 PC 환경보다 실행 시간이 짧은 것을 확인할 수 있다. 하지만 z = 7 이하부터는 PC 환경이 안드로이드 기기 환경보다 나은 성능을 보였다. 그리고 z = 5까지 계산한 결과를 안드로이드 기기(A)/PC(P) 백분율 기하 평균으로 나타내었을 때 157%로 안드로이드 기기와 PC 성능 차이가 크지 않았다. 안드로이드 기기의 처리 시간이 최댓값인 (18, 15, 5)에서 131.15ms로 연산에 많은 시간이 소요되지 않았다.

인터프리터 성능 평가실험을 통해 안드로이드 기기 환경이 프로그래밍 실습에 교육용으로 큰 제약이 되지 않는다는 것을 알 수 있다.

6. 고 찰

본 논문에서는 Scheme의 객체지향 확장형인 RScheme 시스템을 사용하여 서버 없이 독립적으로 작동하는 모바일 앱을 구현하였다. RScheme의 REPL 환경에서 작성된 코드는 인터프리터 방식으로 즉각적인 결과 확인이 가능하다.

Table 4. The comparison of the Tak function on ARM and x86

Input	Number of calls	ARM	x86
		time (ms)	time (ms)
18, 12, 9	1,085	0	0
18, 12, 8	4,321	0	0
18, 12, 7	16,701	4	0
18, 12, 6	63,609	7	0
18, 12, 5	240,457	67	0

환경에 따른 인터프리터 성능 평가에서 Tak 벤치마크의 입력값 (18, 15, 5)에서 안드로이드 기기 환경에서 급격히 속도가 늦어져 원인을 분석해 보았다.

Table 4에서 확인할 수 있듯이 x86의 경우 calls와 무관하게 실행 시간이 0ms가 나오고, ARM의 경우 calls의 숫자가 커질수록 실행 시간이 오래 걸리는 것을 확인할 수 있다. Tak 함수의 특성상 공간복잡도에 대한 의존도가 낮으므로, CPU 연산을 기준으로 놓고 해당 결과를 해석한다면 CPU 설계 관점으로 인해서 차이가 발생한다고 가정할 수 있다.

안드로이드, iOS 등과 같이 모바일 및 휴대용 기기를 위해서 설계된 ARM이 x86에 비해 사용 트랜지스터 개수 측면에서 불리하다. 이는 x86이 일련의 연산을 한 번의 주기로 실행하는 데 적합한 CISC(complex instruction set computer) 구조로 설계되어 있어 더 많은 트랜지스터를 사용한다. 따라서 x86은 연산 속도 측면에서 ARM보다 더 나은 성능을 보여준다. ARM의 경우 모바일 및 휴대용 기기를 위해서 설계되었기 때문에 연산 속도보다는 저전력을 위해서 더 적은 수의 트랜지스터를 사용한다. GAMESS(General Atomic and Molecular Electronic Structure System) 성능 평가를 통해 벤치마크를 수행했을 때 연산 처리 시간은 x86 〈 ARM64 〈 ARM32 순으로 x86이 효율적이며, 에너지 효율성은 ARM32 〈 x86 〈 ARM64 순으로 ARM64가 효율적이라는 연구가 존재한다[18].

그리고, 안드로이드 운영체제는 기기마다 성능에 관한 제한을 별도로 하고 있다. 따라서 본 논문에서와 같이 JNI를 사용하면 외부 애플리케이션을 호출하게 되면 부가적인 연산에 따른 부하가 발생한다. Java가 아닌 코드 호출과 관련된 비용이 발생하여 일부 애플리케이션의 경우 실제로 성능이 저하된다는 사실에 관한 연구가 이미 존재한다[19].

프로그래밍 교육에서 일반적으로 사용되지 않는 Scheme 언어의 보편성에 대한 방안이 필요하다. 프로그래밍 실습을 처음 접하는 학생들의 대부분은 블록 기반 언어의 스크래치나 엔트리, 또는 텍스트 기반 언어의 Python이나 C를 사용한다. 이러한 보편적인 학습 경험을 기반으로 Scheme 코드를 보편적으로 사용되는 다른 언어로 변환하는 방법에 관한 연구가 필요하다.

SRFI(Scheme Request for Implementation)는 Scheme 언어의 라이브러리 및 확장을 조정하기 위한 것으로, R^5RS 표준 이후 SRFI를 통해 특정 기능의 도입이 가능하다. 본 논문에서 사용된 RScheme은 현재 R^4RS 표준을 바탕으로 하고 있으므로 SRFI를 지원하지 못하고 있다. 그리고 R^6RS 표준은 SRFI 라이브러리 및 R^5RS 표준과의 호환성 문제로 교육용으로는 적합하지 않다는 의견이 있다[20]. 이러한 문제를 해결하기 위해 교육 및 연구 목적으로 SRFI의 지원 가능성과 라이브러리 호환성에 관한 연구가 필요하다.

7. 결 론

이 논문에서는 Scheme 프로그래밍을 지원하는 모바일 앱을 설계하고 구현하였다. Scheme 인터프리터로는 RScheme을 이용하였는데, C로 작성된 RScheme은 JNI를 활용하여 앱과 결합할 수 있다. C로 작성된 RScheme 인터프리터를 동적 라이브러리로 컴파일하면 JNI를 이용하여 동적 라이브러리를 Java에서 호출할 수 있으므로, NDK를 통해 안드로이드 앱으로 작성할 수 있다.

스마트폰 과의존으로 컴퓨터에 익숙하지 않은 학생들이 친숙한 스마트폰으로 소프트웨어 교육을 진행할 수 있고 네트워크를 사용하지 않아도 이용할 수 있는 프로그래밍 앱을 구현하였다. 안드로이드 기기와 PC 환경에서의 인터프리터 성능 평가실험을 통해, PC에 대한 안드로이드 기기의 성능 백분율을 측정한 결과 Derivative 벤치마크는 평균 131%, Tak 벤치마크는 평균 157%인 것으로 나타났다. 따라서 안드로이드 기기 환경의 성능도 일반 PC 환경에 못지않음을 알 수 있다.

절대적인 수행 시간을 보면 안드로이드 기기의 처리 시간 최댓값은 Derivative 벤치마크의 경우 19.8ms, Tak 벤치마크의 경우는 131.15ms로 나타났다. 이러한 짧은 연산 처리 시간은 실행 지연을 체감하기 어려움을 보여준다. 그러므로 안드로이드 기기 환경을 교육용으로 선택해도 큰 문제가 없다는 것을 알 수 있다. 또한, Scheme의 고차 함수와 다중 패러다임 언어 특징을 이용하면 학생들에게 다양한 문제 해결 방식을 보여주기에 적합할 것이라고 예상된다.

환경에 따른 인터프리터 성능 평가에서 Tak 벤치마크의 입력값 (18, 15, 5)에서 안드로이드 기기의 급격히 속도 저하 원인을 분석한 결과, CPU 설계로 인한 성능 차이와 JNI를 통한 외부 애플리케이션 호출 오버헤드인 것으로 분석되었다. 향후 이러한 성능 저하 문제를 해결하는 방안에 관한 연구를 진행할 예정이다. 또한, Scheme 코드를 보편적으로 사용되는 다른 언어로 변환하는 방법, SRFI를 위한 표준 지원과 라이브러리 호환성 해결에 관한 연구도 향후 연구로 생각해 볼 수 있다.

References

[1] B. Kim, “Revision and implementation of App Inventor open source,” *Journal of the Korea Institute of Information and Communication Engineering*, Vol.22, No.2, pp.221-226, 2018.

[2] H. Kim, “For improving quality of classes in liberal arts programming classes analysis of role of instructor and learning achievement and satisfaction,” *The Journal of the Convergence on Culture Technology*, Vol.9, No.3, pp. 745-752, 2023.

[3] K. Palin, A. M. Feit, S. Kim, P. O. Kristensson, and A. Oulasvirta, “How do people type on mobile devices? Observations from a study with 37,000 volunteers,” *Proceedings of the 21st International Conference on Human-Computer Interaction with Mobile Devices and Services*, 2019.

[4] M. M. T. Chakravarty and G. Keller, "The risks and benefits of teaching purely functional programming in first year," *Journal of Functional Programming*, Vol.14, No.1, pp. 113-124, 2004.
[5] C. T. Haynes, R. K. Dybvig, D. P. Friedman, L. Sadler, and G. Springer, "CISE educational infrastructure: Tools and techniques for use of the Scheme programming language in undergraduate education," Computer Science Department in Indiana University, 1993.
[6] G. J. Sussman and G. L. Steele Jr, "Scheme: A interpreter for extended lambda calculus," *Higher-Order and Symbolic Computation*, Vol.11, No.4, pp.405-439, 1998.
[7] D. Kolbly, RScheme: Design and Implementation [Internet], https://www.rscheme.org.
[8] K.-C. Son, "The predictor of android application speed up by using Android NDK," Master's thesis, The Graduate School of Electronics Engineering in Chonbuk National University, 2012.
[9] Google for Developers, Android NDK [Internet], https://developer.android.com/ndk.
[10] R. P. Gabriel, "Performance and Evaluation of Lisp Systems," Vol.263, Cambridge, MIT press, 1985.
[11] J. McCarthy, "An interesting LISP function," *ACM Lisp Bulletin*, Iss.3, pp.6-8, 1979.
[12] Evilbinary, RScheme for Android [Internet], https://github.com/evilbinary/RScheme-for-android.
[13] J.-K. Lee, J.-M. Choi, S-Y. Lee, H.-S. Choi, and C.-D. Lee, "A performance improvement study on Android application using NDK," *Journal of the Korean Information Processing Society*, Vol.19, No.2, pp.750-751, 2012.
[14] C. Lee and S. Oh, "Design and Implementation of JPP (JNI PreProcessor)," *Journal of the Korean Information Processing Society*, Vol.9, No.1, pp.129-136, 2002.
[15] C. Lee, "Java preprocessor for integration of Java and C," PhD thesis, Graduate School of Computer Science in Dongguk University, 2002.
[16] E. Koh, N. Kim, S. Hwang, and J. Lee, "Porting and implementation of a 3D cube game using Android NDK (Native Development Kit)," *Journal of Digital Contents Society*, Vol.14, No.3, pp.381-390, 2013.
[17] T. R. Beelders and J.-P. L. du Plessis, "Syntax highlighting as an influencing factor when reading and comprehending source code," *Journal of Eye Movement Research*, Vol.9, No.1, pp.1-11, 2016.
[18] K. Keipert, et al., "Energy-efficient computational chemistry: Comparison of x86 and ARM systems," *Journal of Chemical Theory and Computation*, Vol.11, No.11, pp.5505-5061, 2015.
[19] C.-M. Lin, J.-H. Lin, C.-R. Dow, and C.-M. Wen, "Benchmark Dalvik and native code for Android system," In *Proceedings of 2011 Second International Conference on Innovations in Bio-inspired Computing and Applications*, pp.320-323, IEEE, 2011.
[20] T. Kato, "Implementing R7RS on an R6RS Scheme system," In *Proceedings of 2014 Scheme and Functional Programming Workshop (Scheme '14)*, Washington DC, pp. 1-7, 2014.

김 동 섭

https://orcid.org/0009-0004-1250-8087
e-mail : seob614@naver.com
2021년 부산대학교 IT응용공학과(학사)
2022년 ~ 현 재 부산대학교 정보융합공학과 석사과정
관심분야 : Programming Language, Programming Education, Compiler and Interpreter, Mobile/Web Programming, Beacon Programming

한 상 곤

https://orcid.org/0000-0002-9037-7981
e-mail : sangkon@pusan.ac.kr
2013년 부경대학교 컴퓨터공학(석사)
2017년 ~ 현 재 부산대학교 정보융합공학과 박사과정
관심분야 : Programming Language, Functional Programming, Programming Education

우 균

https://orcid.org/0009-0006-0469-3723
e-mail : woogyun@pusan.ac.kr
1991년 KAIST 전산학(학사)
1993년 KAIST 전산학(석사)
2000년 KAIST 전산학(박사)
2000년 ~ 2004년 동아대학교 컴퓨터공학과 조교수
2004년 ~ 현 재 부산대학교 정보컴퓨터공학부 교수
관심분야 : Programming Language, Compiler and Interpreter, Korean Programming Languages, Programming Education, Program Analysis and Visualization, Software Testing