프로그래밍 교육에서 실습 언어의 선택*

이광근

2002년 6월

요 약

국내 대학들이 마련하고 있는 대부분의 프로그래밍 교과과정에서 놓치고 있는, 중요하지만 가르치기 어려운 내용들을 보완할 수 있는 방안에 대한 소견을 정리해 보았다. 강의에서 사용할 실습 프로그래밍 언어에 대한 제안으로 좁혀진 내용이다.

1 대개의 프로그래밍 교육

양질의 IT 인력을 양성하려는 학교에서 손쉽게 선택하는 프로그래밍 교육 방법은, 지금 현재 많은 사람들이 접하는 프로그래밍 언어들을 가지고 프로그램을 짜보도록 하는 것이다. 국내 대학 대부분의 경우를 보면 C나 Java를 프로그래밍 교육에서 강의하고 실습언어로 사용하는 것을 볼 수 있다.

사실 프로그래밍은 다양한 수준(기계어 수준에서부터 소프트웨어 기획/명세 수준 이상까지)에서 이루어지는 작업이고, 이 모든 수준에서 이용되는 언어들이 프로그래밍 실습언어로 사용되야 하겠지만, C나 Java를 주로 선택하는 이유는 아마도 현재 대다수의 소프트웨어 개발이 그러한 언어들로 이루어진다는 판단 때문일 것이다. 지금이 1950-60년대라면 대부분의 프로그래밍 교육은 아마 에셈블리어와 링커를 이용해서 진행되었을 것이다.

이렇게 "지금 현재 많은 사람들이 접하는" 프로그래밍 언어로 프로그래밍 교과과정을 마련하는 것은 무난한 듯 하다. 학교나 주변 교수들을 설득하기 쉽고, 학생들도 거부반응 없이 수업에 임할 것이다. 한가지 어려움

*[제2회 대학 소프트웨어 프로젝트 과목 활성화 웍샵] 2002년 7월 15-16일, 을 위해 준비됨.

이라면, 꾸준히 변하는 프로그래밍 언어의 인기차트에 따라 교과과정을 빨리 적응시켜야 한다는 점이다. 오늘은 C이므로, C로. 내일은 C++이므로 C++로. 지금은 Java이므로 Java로. 나중에는 ML이므로 ML로.

그러나 이런 교육과정의 아쉬운 점은, 프로그래밍 교육에서 무엇을 가르쳐야 하는 지에 대한 초점이 표면적인 것에 맞추어져 있다는 데 있다. 의아해하시겠지만, 지금 당장 널리 쓰이는 언어들은 현재 프로그래밍 기술의 중요 성과들을 충실히 갖추고 있지 못하다. 때문에, 그렇게만 프로그래밍 교육을 받은 학생들은 소프트웨어 기술의 뒷 북만을 치기 쉽상이 되거나, 수명이 짧은 프로그래밍 기술에 치우치게 될 것이다.

글로벌 선두를 차지하는 소프트웨어 인력풀은 이제는 기술흐름의 핵심을 맛본 사람들로 구성될 것인 바, 위와같은 교과과정은 그런 인력의 저변을 두텁게하는 데는 바람직하지 않다고 본다. 표면은 항상 변하고 그 유효기간은 짧다고 한다. 부지런히 지금 현재의 프로그래밍 언어로 교과과정을 만들고 재편해 보지만, 학생들이 익히는 내용의 유효기간은 수시로 변하는 표면의 그것만큼 짧을 수 밖에 없을 것이다. 열심히는 뛰지만 이기지는 못하는 축구팀을 만들 수 밖에 없는 훈련 프로그램이랄까.

2 프로그래밍 교육의 두가지 내용

프로그래밍 교육의 촛점을 표면에서 핵심으로 옮겨보면, 다음 두개의 질문에 맞춰질 것이다:

- 현재의 프로그래밍 기술 수준을 경험시키는가?
- 다양한 프로그래밍 생각의 틀을 갖추게하는가?

현재의 기술 수준을 익히도록 하는 것은 당연하다. 선풍기 기술만 알고 있는 엔지니어하고 물펌프와 배수시스템 기술까지 알고있는 엔지니어하고는 홍수난 논밭의 물을 제거하는 문제를 해결하는 능력에 차이가 날 것이다. 다양한 생각의 틀을 갖추게 하는 것도 필요하다. 문제를 정의하고 답을 궁리하는 생각의 틀을 적절히 선택할 줄 아는 엔지니어는, 그렇지 못한 엔지니어를 항상 능가할 것이다.

고민해야 할 점은 무슨 기술들이 현재의 프로그래밍 기술이고 어떤 프로그래밍 생각의 틀이 중요한 것이냐 인데, 이 고민의 답은 강의에서 사용하는 프로그래밍 언어를 통해 학생들에게 전달될 것이다. 따라서 문제는 무슨 언어들을 선택하면 되냐는 것인데, 아쉽게도 위의 두가지 질문에 대해서 C와 Java만을 선택하는 것은 부족한 면이 있다.

2.1 놓치기 쉬운 프로그래밍 기술

지금 널리 쓰이는 실습언어로 프로그램하면서 경험할 수 없는 소프트웨어 기술은, 프로그램의 오류를 자동으로 찾아내는 기술들이다. 프로그래밍 교육의 한 목표가 버그 없는 소프트웨어를 값싸게 만드는 기술을 익히게하는 것 인데, 이 기술의 핵심 동향을 C나 Java를 통해서는 익힐 수 없는 아쉬움이 있다.

소프트웨어 버그를 자동으로 찾아주는 기술은 지금까지 2세대 기술이 완성되었는데, C와 Java는 1세대 기술만을 갖추고 있을 뿐이다. 1세대 오류 검증 기술은 1970년대에 달성된 것으로, 생긴게 잘못된 프로그램을 자동으로 찾아내는 기술이다. 이 기술이 문법검증기술(parsing)이고, 완전히 완성되어 어느 프로그래밍 언어에서나 제공되는 기술이다. 덕택에, 어느 프로그래밍 교육에서도 프로그래머가 문법 오류를 손으로 찾아내도록 하는 실습언어를 사용하지 않는다.

C나 Java가 놓치고 있는 2세대 벅잡는 기술은 1990년대에 완성되기 시작한 기술인데, 타입검증(type checking)이라는 기술이다. 이 기술은 프로그램이 실행중에 잘못된 값이 잘못된 계산과정에 휩쓸릴 수 있는 경우가 없는지를 컴파일러가 미리 안전하게 확인해 준다. C나 Java만을 사용하는 프로그래머들은 이렇게 미리 엄밀하게 확인해 주는 실용적인 기술이 가능하다는 사실을 알지 못하게 된다. 이미 이 기술을 효과적으로 구현한 실용적인 언어들이 속속 출현하고 있고, 외국의 주요 대학들은 그러한 언어를 프로그래밍 실습의 언어로 선택하고 있다. 2세대 벅잡는 기술을 맛본 엔지니어는 그러한 기술의 가능성 마져도 알지 못하고 있는 엔지니어를 쉽게 능가할 수 있는 것은 뻔하다. 사실 이제는 어쩌면 3세대 벅잡는 기술[1]을 학생들에게 맛보이는 것도 필요한 시점이라고 보면, C와 Java에만 머문다는 것은 아쉬운 면이 많다.

2.2 놓치기 쉬운 프로그래밍 파라다임

C나 Java로 경험할 수 없는 생각의 틀은 값중심(value-oriented)으로 문제를 정의하고 해결하는, 수학에서 우리가 오랫동안 사용해온 간편한 언어 인프라이다. 값중심으로 생각하기는 물건(object)을 만들고 변화시키는 기계적인 과정에 신경쓰는 것에서 벗어나서, 값(value)을 정의하고 계산하는 과정에 집중하는 방법이다. 중고등학교 수학에서 대부분의 학생들이 이미 익숙하게 사용해 왔었던 쉽고 간단한 프로그래밍 개념이다. 예를들어 어느 수학책의 한 페이지에서 따온 다음의 단락을 보자:

[1] 생긴모습도 멀쩡하고, 실행중에 잘못된 값이 흘러들지도 않지만, 실행중에 가져야할 정교한 조건을 만족시킬 수 없는 프로그램을 컴파일러가 검증해 내는 기술.

"V를 $(interior\, U) \cup f^{-1}(W)$ 라고 하자.
그러면 $V \cap (interior\, W) = f(U)$ 가 사실임을 알 수 있다."

여기서 첫 문장에 나타나는 $interior\, U$는 U가 가지는 값을 변화시키지 않는다. 그래서 두번째 문장에 나타나는 U와 처음의 U는 같은 것이지, 문장들이 실행되면서 변하는 물건이 아닌 것이다. $f^{-1}(W)$라는 연산도 마찬가지다. 첫문장에서나 두번째 문장에서나 W는 같은 값을 가질 뿐이다.

왜, 수학(과학)의 발전을 소통시켰던 이러한 간편한 언어 인프라를 프로그래머가 익히도록 해야 할까? 컴퓨터 소프트웨어는 수학의 프로그램(논증)에서와 똑같이, 그 참/거짓을 판명해야 하는 필요성이 점점 중요해지고 있기 때문이다. 과학에서 프로그램의 참/거짓을 판명하는 것이 과학의 존재와 발전의 근간이었듯이, 컴퓨터 소프트웨어의 존재와 발전의 근간은 이제 프로그램의 참/거짓을 판명하는 기술이 되고있다. 프로그램이 옳고 그른지를 판명하는 것은 다름아니라 프로그램이 오류없이 생각대로 작동할지 안할 지 를 확인하는 것이기 때문이다. 이런 확인을 미리 자동으로 해주는 기술이 꽃피는 땅은 값중심의 개념을 바탕으로 한 프로그램들이 될 것으로 보인다. 이는 과학기술의 참/거짓을 효과적으로 소통시켰던 언어가 값중심의 언어였다는 사실로 쉽게 예상할 수 있는 일이다.

이러한 프로그래밍 개념은 이제 세계 주요 대학의 기초 프로그래밍 과목에서 강의되고 있는데, 이 개념이 지금까지 소프트웨어 세계에서 묻혀졌던 것은, 컴퓨터에서 그 개념을 실용적으로 지원하는 방법이 없었기 때문이다. 하지만, 이제는 그러한 자연스럽고 간단한 프로그래밍 개념을 효과적으로 구현해주는 프로그래밍 언어들이 이미 현장으로 나오기 시작했고 주요 부분에서 이용되고 있다. 전통적인 컴파일러 기술들이 에셈블리 프로그래밍을 만족스럽게 대체시켰듯이, 값중심의 프로그래밍 언어를 구현하는 새로운 컴파일러 기술들이 충분히 쌓여서 이미 산업체에서 쓰이는 언어들이 되었을 지경이다. 이런 사실을 우리가 보지 못했던 것은 아마도 우리로서는 C와 Java의 관성에 휩쓸릴 수 밖에 없는 자신감 부족이 한 원인이라고 본다.

3 실습 프로그래밍 언어들의 제안

C나 Java 만으로는 놓치는 점을 보충해 주는 실습 언어로는 논의한 대로 다음의 두 질문을 만족하는 것을 찾으면 될 것이다. 오류를 자동으로 찾아내 주는 기술의 현재 수준을 제대로 갖춘 언어인가? 값중심의 프로그래밍 파라다임을 익힐 수 있는 언어인가?

주요 프로그래밍 언어들을 이 두 기준과 관련해서 일별해 보면 (아래 표 참조) 선택할 언어는 두개의 그룹에서 하나씩이면 되는 것으로 정리된

다: C, C++, Java 등의 그룹과 ML[2], Haskell[3], Scheme[4]등의 그룹. 특히 C나 Java중에 하나, ML과 Haskell중에 하나를 조합해서 이용할 것을 제안한다.

	C	C++	Java	ML	Haskell	Scheme
1세대 벅잡는 기술	√	√	√	√	√	√
2세대 벅잡는 기술				√	√	
object-oriented		√	√			
value-oriented				√	√	√
산업체 실용성	√	√	√	√	√	√

이 제안이 우리 프로그래밍 교육에 사용할 실습언어를 지금 시점에서 올바르게 안내하는 것이라고 믿는다. 아무쪼록 잘못 처리하는 아는 바가 아니길 빈다.

□

[2] `www.ocaml.org, www.smlnj.org, ropas.kaist.ac.kr/n`

[3] `www.haskell.org`

[4] `www.drscheme.org, www.swiss.ai.mit.edu/projects/scheme`