“Beating the averages”와 “Revenge of the nerds”를 읽고

2009-20769
김성준

나는 전기공학부 학생이지만 회로나 반도체, 통신 등 전기공학부 주력(?) 보다는 프로그래밍에 관심이 많았고 대학원도 소프트웨어 관련 연구실로 진학했다. 열심히 미분방정식을 푸는 친구들 사이에서 열심히 키보드 두들겨가며 코딩을 해왔다. 대학교 1학년 때부터 코딩을 했으니 어언 횟수로 6년째 프로그래밍 언어와 함께 해오고 있다.

그 동안 여러 프로그래밍 언어를 접해왔다. 1학년 때 ‘프로그래밍의 기초’라는 수업에서 처음으로 C를 통해 프로그래밍이 무엇인지 경험할 수 있었다. 비교적 간단한 문법으로 이리저리 키보드를 눌러 프로그램을 짜고 내가 생각하는 데로 동작하는 것을 보니 흐뭇하기만 했다. 그리고 2학년 때는 ‘프로그래밍 방법론’에서 C++을, ‘자료구조’에서 Java를 처음 사용해봤다. 3학년 때는 이 세 언어를 쭉 사용했었고, 4학년 졸업 프로젝트 때 Python을 조금 공부했다. 그리고 2008년 1년간 인턴을 하며 Ruby나 bash shell script를 공부해야 했었고, 지금 프로그래밍 언어 수업을 들으며 Ocaml과 Haskell을 공부하고 있다.

그러고 보면 나름 다양한 언어를 공부했다고 할 수 있겠다. 명령형 언어의 대표주자라고 할 수 있는 C, OOP를 위한 C++과 Java, 스크립트 랭귀지인 Python, Ruby과 Shell script, Functional 랭귀지인 Ocaml과 Haskell. 이런 언어들은 각기 고유한 문법과 특성들을 가지고 있었다.

하지만 돌이켜 생각해보면 여러 언어들을 잘못 공부해 왔다는 생각이 든다. 각기 다른 프로그래밍 언어는 이전의 언어가 가진 문제점을 개선하고 새로운 특성을 추가하는 방향으로 발전해 왔다. 따라서 그 언어를 배우고 사용하는 우리도 그러한 점을 명확히 깨닫고 각 언어의 특성을 이해하는데 중점을 둬야한다. 그러나 이제까지는 자의반 타의반으로 과제를 하기 위해 언어를 배우면서 그러한 언어의 특성을 이해하는 과정이 빠진 것 같다. 단지 문법을 외우고, 예제를 통해 코딩하는 법만을 배운 채 사용해 왔다. 그리고는 새로운 언어를 이전의 언어의 틀에 비춰 이해하곤 했다. C++을 이용해 코딩을 하고 있지만 정작 나온 프로그램은 C와 거의 동일한 프로그램이 만들어지고 마는 것이다.

“Beating the averages”와 “Revenge of the nerds”라는 글을 읽으면서 막연히 잘 못 되었다는 느낌을 좀 더 명확히 할 수 있었다. 이 두 글은 high-level 언어인 Lisp의 좋은 점을 역설하고 있다. Lisp은 1950년대에 John McMarthy에 의해 설계되었고 그의 제자 Steve Russell에 의해서 실제로 구현되었다. Lisp은 처음부터 프로그래밍을 위해서 만들어진 언어는 아니다. 다만 John McMarthy가 Turing 머신을 좀 더 편하게 정의하고자 하는 과정에서 만들어진 이론적인 논리 모델이었다. 이를 Steve Russell이 Lisp interpreter를 만들면서 실제 프로그래밍 언어가 된 것이다.

이 두 글은 모두 Lisp의 강력함에 대해 이야기하고 있다. 언어의 강력함이란 인간의 생각을 얼마나 편리하고 정확하게 표현할 수 있느냐 하는 것이다. 물론, ‘프로그래밍 언어’ 수업에서 배운 것과 같이 현재의 high-level 언어들이 표현할 수 있는 프로그램들의 집합은 거의 동일하다고 할 수 있다. 하지만 프로그래머가 얼마만큼의 노력을 통해서 그러한 프로그램을 작성할 수 있느냐의 문

제인 것이다. 나무를 이용하더라도 집을 지을 수 있다. 하지만 좀 더 강력한 콘크리트와 철근을 이용한다면 좀 더 쉽고, 좀 더 빠르게, 좀 더 크고 튼튼한 집을 지을 수 있는 것이다.

Lisp의 강력함을 이 두 글에서는 다른 언어들이 가지지 못 한 특성들로 설명하고 있다. "Revenge of the nerds"에서는 Lisp이 만들어지던 당시 다른 언어들은 가지지 못한 Lisp만의 특성 9가지를 나열하고 있다.

1. Conditionals
2. A function type
3. Recursion
4. Dynamic typing
5. Garbage-collection
6. Programs composed of expressions
7. A symbol type
8. A notation for code using trees of symbols and constants
9. The whole language there all the time.

이 중 1 – 5는 이제는 널리 퍼지고 당연하다고 생각되는 특성들이다. 그리고 6은 현재 주류 언어에서 나타나고 있으며 7은 Python에서 구현되더 있다. 하지만 아직도 8, 9는 Lisp이 유일하다. 8, 9는 프로그래머가 프로그램을 만드는 프로그램을 만들 수 있다는 것을 의미한다. 이러한 방법을 macro라고 한다.

이처럼 그 언어는 이전의 언어들이 가지지 못한 고유한 특성에서 그 강력함을 찾을 수 있다. 그 당시의 다른 언어로 Fortran 1을 들고 있다. 이 언어는 지금의 Fortran과는 달라서 수학 연산을 추가한 어셈블리 언어와 비슷했다고 한다. 이를 Lisp과 비교했을 때 어떤 언어가 더 강력한지는 말할 것도 없다. 특히나 위의 9가지 특성 중 많은 부분을 다른 언어를 통해서 경험했고 이런 특성이 없는 언어는 상상조차 할 수 없다.

하지만 나는 그 동안 프로그래밍 언어를 공부하면서 이런 특성을 잘 이해하지 못 한 것 같다. C++ 언어로 프로그래밍을 하면서도 한동안 클래스, 상속, 다형성, 템플릿, STL 등을 사용하지 않았다. 파일의 확장자만 c에서 cc로, 컴파일러만 gcc에서 g++로 바뀌었을 뿐 프로그램 그 자체는 C 프로그램과 거의 다를 바가 없었다. 이는 Java나 Python, Ruby를 사용할 때도 별반 차이가 없었다. 상위 언어에서 하위 언어의 특성만을 골라 사용했다고 볼 수 있다.

왜 이런 일이 벌어졌을까? 아마도 과제를 하기 위해서 언어를 배웠기 때문일 것이다. 학년이 높아짐에 따라, 언어가 발전한 순서대로 수업에서 요구하는 언어가 달라졌다. 그리고 과제는 지정된 언어를 이용해서만 구현되어야 했다. 당시 중요한 것은 시간 내에 과제를 해서 내는 것이었고 이를 위해서 언어의 문법을 조금 배우고, 예제를 통해 쓰는 법을 배운 뒤, 바로 코딩을 시작하였다.

이런 급박한 진행 속에서 이 언어만의 고유한 특성을 이해하는 일은 뒷전이었다. C++에는 C와 달리 클래스가 있고 상속될 수 있다는 것은 배웠다. 하지만 이러한 특성들이 어떤 도움이 되는지, 이를 통해 만든 프로그램이 C 프로그램보다 어떤 점에서 강력한지 배울 기회는 없었던 것 같다.

그 과제가 새로운 특성을 요구할만큼 어렵지 않았다는 것이 다른 이유이다. 과제가 너무 쉬웠다는 말은 아니다. 다만 C++을 배운 수업에서 요구한 문제가 C++이 아니면 전혀 풀 수 없었던 것

은 아니다. 귀찮기는 했지만 충분히 C로도 풀 수 있는 문제였다. 이렇다 보니 어지간 해선 C++을 열심히 배워 사용하기 보다는 C++을 C의 틀에 적당히 맞춰 사용했을 뿐이다.

그래도 이번 학기에 프로그래밍 언어 수업을 들으며 이러한 생각을 바꾸게 되었다. 수업에서 말하는 '더 좋은 언어'라는 표현이 잘 와닫지 않았다. 흔히 말하는 C로도 충분히 많고 다양한 프로그램을 짤 수 있다고 생각했기 때문이다. 하지만 Ocaml을 하면서 그 표현의 의미를 조금은 깨달을 수 있었다.

가장 큰 비교가 되었던 예제가 '컴파일러'였다. 학부 때 컴파일러 수업을 들으면서 lex와 yacc를 이용해 parser를 만드는 과제가 있었다. 이 주에서 AST를 만드는 과정이 제일 힘들었다. Lex와 yacc가 c 기반이었기 때문에 AST를 표현하기 위해서 수많은 structure와 사용자 정의 type, union을 사용해야만 했다. 그리고 그 AST를 다루기 위해서 매번 structure의 union type의 값을 통해서 어떤 expression을 나타낸 structure인지 알아보고, 각 structure마다 필요한 변수를 얻어내는 함수를 일일이 만들어야 했다. 참으로 힘들었던 시간이었다. 하지만 Ocaml에서 이 같은 반복적이고 즐겁지 않은 일은 확 줄어버렸다. Ocaml이 가지는 강력함 때문이었다. 이 때부터 Ocaml과 같이 강력한 언어의 매력을 안 것 같다.

이런 생각을 하면 글을 읽던 도중 '어떻게 그럴 수 있을까?'라는 생각이 들었다. 사실 Lisp은 1950년대에 만들어진 언어이다. 그런 언어가 어떻게 현대의 언어보다 강력할 수 있을까? 이런 이유를 두 글에서는 Lisp의 뿌리에서 찾고 있다. 앞서 설명한 것과 같이 Lisp은 프로그래밍 언어로써 만들어지지 않았다. 다만 이론적인 모델로서 만들어지고 후에 프로그래밍 언어로 구현된 것이다. 이에 반해 현대의 언어들과 같은 계보에 있는 Fortran 1의 경우에는 처음부터 프로그래밍 언어로 구현되었고 추후에 개량되어 현재의 Fortran 5의 모습을 갖추고 있다. 이렇게 각자 다른 뿌리를 가지고 있다. Lisp은 태생부터 강력한 언어였으며 Fortran은 태생부터 빠른 언어였다. 그리고 구후 Lisp은 점차 빨라지는 방향으로 진화했고 Fortran과 같은 주류 언어들은 좀 더 강력해 지는 방향으로 진화한 것이다. 현대에 이르러서는 두 줄기 모두 비슷한 지점에 모였다고 할 수 있지만 아직은 Lisp이 다른 언어들에 비해서 강력한 것이다.

이 글들을 읽기 전에는 Lisp에 대한 선입견을 가지고 있었다. 온통 괄호로 둘러싸인 코드. 어떤 일을 하는 코드인지 전혀 감이 오지 않았다. 정말 이게 프로그램 코드인가라는 생각도 들었다. 더구나 1950년라는 오래된 시절의 언어라는 말에 어셈블리 언어처럼 실제로는 쓸 수 없는 언어라고 생각했다. 그러나 이런 글을 통해서 만들어진 시기와 언어의 강력함에는 별반 상관관계가 없다는 것을 깨달았다. 지금이라도 Lisp이라는 언어가 어떤 언어인지, 특히 macro로 어떤 것인지 알아보고 싶은 마음이 든다.

그렇다면 Lisp이라는 언어는 이런 강력함에 비해서 널리 사용되지 못 한 것일까? 두 글에서는 공통적으로 평균적인 것에서 벗어나는 것에 대한 두려움을 그 원인으로 말하고 있다. 글에서 밝히고 있듯이 프로그래머에게 있어 언어는 단순한 기술이 아니라 습관이다. 한번 손에 익은 프로그래밍 언어를 다른 것으로 바꾸기가 쉽지 않은 것이다. 그리고 나처럼 바꾼다 하더라도 새로운 언어를 이전 언어의 틀에 맞춰서 이해하기 일수여서 제대로 언어를 받아들이지 못 한다.

이는 프로그래밍 언어가 프로그래머의 시야를 제한하기 때문이다. 나 또한 그랬듯이 프로그래밍 언어를 배우면 이 언어를 통해서 모든 프로그램을 작성할 수 있다는 생각이 든다. 그와 동시에

내가 짤 수 있는 프로그램이 그 언어를 통해서 표현할 수 있는 것으로 한정된다. 만약 쉽게 작성되지 않는 프로그램이 있다면, 이는 나의 문제이지 언어 자체의 표현력의 문제라고 생각하지 않는 것이다. 이는 새로운 언어를 받아들일 때도 마찬가지이다. 글에서 나온 'Blub paradox'와 같이 자신의 언어보다 약한 언어를 보면 어떤 부분이 모자란지 바로 알 수 있다. 그러나 방향을 바꿔서 자신의 언어보다 강력한 부분은 볼 수도 없고 볼려고 하는 생각도 하지 못 하는 것이다.

그러나 그 볼 수 없는 부분이 새로운 언어를 배우는 진짜 이유라고 할 수 있다. 클래스와 상속, 템플릿과 STL을 쓰지 않는다면 C++을 쓰는 의미가 없다고 할 수 있다. Functional 언어의 특성을 쓰지않고 let .. in 만으로 프로그램을 만든다면 Ocaml이나 Haskell을 쓰는 의미가 없다. 이렇게 다양한 언어의 특성과 사용할 때를 알고 적시적소에 사용하는 것이 필요하다.

이 글에서는 Lisp이라는 강력한 언어를 통해 경쟁에서 이긴 두 기업을 소개하고 있다. 기존의 사람들은 감히 시도하지 못 했던 시도를 통해서 성공을 거두었다고 할 수 있다. 물론 그 시도는 무모한 것이 아니라 오랜 경험을 통한 자기 확신을 기반으로 한다. 그러한 확신을 막고 있던 '남들과는 다른' 이라는 걱정을 떨쳐버린 것이다.

나 역시 이러한 생각들에서 벗어나지 못 했던 것 같다. 막상 호기심에 다양한 언어를 공부하면서도 이를 직접 연구나 후에 직장에서 쓸 생각을 못 했던 것 같다. 새로운 것에 대한 막연한 동경만을 가지고 있을 뿐, 이렇나 언어가 가지는 강력한 힘을 제대로 깨닫지 못 했고, C++이나 Java와 같은 주류 언어만이 실제 현장에서 쓸모가 있을 것이라고 생각했다.

이런 글을 통해서 그런 생각들이 한참 잘 못 되었고 나의 가능성을 제한하고 있다는 생각을 했다. 새로운 마음으로 내가 배운 언어들을 돌아보고, 내가 놓친, 그 언어만의 특성을 이해하도록 해야겠다. 그리고 글에서 말하는 것처럼 남들이 최선이라고 말하는 것에 집착하기 보단 내가 하고자 하는 일에 최선인 것을 선택하는 연습을 해야겠다.