언어가 사고를 지배한다

Programming Language 2015 Fall Essay

2009-11841 최영진

요약

2015년 현재, 하나의 IT 기업인 애플Apple Inc.의 시가총액은 6,580억 달러로 대한민국 1년 예산의 두 배가 넘는다. 현대인에게 컴퓨터와 스마트폰 없는 삶이란 상상하기 불가능하고, 미래로 갈수록 정보통신 기술에 대한 의존도는 기하급수적으로 커질 것이다. 인간의 지성과 문화를 지탱하는 것이 언어이듯이 컴퓨터를 이야기할 때 프로그래밍 언어는 빼놓을 수 없는 주제이다. 어떤 언어들로 개발을 해보고, 수업 및 읽을 거리들을 통해 그 연구들을 접하면서 언어의 표현력이 곧 내 사고력이라는 생각이 들었다.

많은 사람들이 그렇듯이 C 언어를 처음 배웠는데, 당시에는 C가 깊은 학문적 토대 없이 당장의 개선에만 초점을 두고 만들어졌다는 이야기는 생각도 하지 못했다. C 언어는 여전히 널리 쓰일 뿐더러 특정 분야에서 무척이나 가치있는 언어이지만 이 태생적 한계로 필연적인 불편함이 수반되었다. “계산이란 무엇인가에 대한 아이디어들”을 읽어보면 언어의 발전이 얼마나 어렵고 아직 미흡한지 느낄 수 있다. “19세기 논리학, 21세기 계산학”은 프로그래밍 언어의 큰 진일보였던 부분을 잘 나타내준다. 처치Church의 Lambda Calculus는 많은 언어의 기초가 되었다. 실제 사용 예시로, 페이스북Facebook은 스팸Spam 차단에 Haskell을 사용하였다. 월가의 한 회사는 금융 시스템을 ML로 구축하기도 했다. 벤처업계의 구루Guru로 유명한 Paul Graham은 “Beating the Averages”라는 글의 도입부에서 창업 후 Lisp를 사용했던 경험을 들려주었다.

특정 언어가 특정 상황에 잘 맞다거나, High-level의 언어를 사용할 때 생산성이 높아졌다는 이야기도 공감하지만 무엇보다 내게 크게 다가왔던 것은 생각하는 지평의 확장이었다. 프로그래밍 원리 수업 때 Scheme이라는 언어를 만나고, 대학에서 가장 컸던 지적 즐거움을 느꼈다.

그러나 Paul Graham이 “Beating the Averages”에서 역설하듯이, 새로운 환경을 접하는 것은 여전히 어려운 일이다. 내게는 그것이 크나큰 매력으로 다가온다. 흐름을 선도할 수 있으면 그 무엇보다 좋겠으나, 나 같은 소시민에게는 새로운 개념에서 오는 사고의 확장만도 충분한 즐거움이다.

본문

2015년 현재, 하나의 IT 기업인 애플Apple Inc.의 시가총액은 6,580억 달러로 대한민국 1년 예산의 두 배가 넘는다. 구글Google Inc.이나 마이크로소프트Microsoft의 가치가 러시아 증시보다 크다는 것도 유명한 이야기이다. 이 거인들이 아니더라도 세상의 주요 흐름이 제조업에서 금융으로 이동했다가 이제 IT로 급물살을 타고 있는 것은 명백한 사실이다.

현대인에게 컴퓨터와 스마트폰 없는 삶이란 상상하기 불가능하고, 미래로 갈수록 정보통신 기술에 대한 의존도는 기하급수적으로 커질 것이다. Bill Gates조차도 컴퓨터 발전의 미래를 예측하지 못했을 정도로 컴퓨터와 인터넷은 급속도로 확산되었다. 지금은 사물 인터넷IoT이라 하여 평범한 물건에 전자칩을 부착하고, 기계 학습Machine Learning이라 하여 인간에 근접한 행동을 만들어내고 있는 시대이다.

인간의 지성과 문화를 지탱하는 것이 언어이듯이 컴퓨터를 이야기할 때 프로그래밍 언어는 빼놓을 수 없는 주제이다. 프로그래밍 언어란 컴퓨터에게 명령을 내리는 도구로, 모든 소프트웨어 기술의 핵심이다. 1+1을 계산하라는 등의 명령을 컴퓨터가 실행할 수 있도록 한다. 개념은 간단하지만 시기와 분야에 따라 수많은 언어가 존재한다. 현재에도 가지각색의 언어가 가지각색의 용도로 쓰이고 있다. 그런데 이렇게 많은 프로그래밍 언어가 연구되고 만들어진 것이 백 년이 될까말까하다는 것은 놀라운 사실이다. 삶에 어마어마하게 가까이 있는데도 말이다.

어떤 언어들로 개발을 해보고, 수업 및 읽을 거리들을 통해 그 연구들을 접하면서 언어의 표현력이 곧 내 사고력이라는 생각이 들었다. 나는 사정상 전공수업을 몇 개 듣기도 전에 소프트웨어 회사에서 병역특례를 하게 되었는데, 상용에서 많이 쓰는 언어만 사용하여 언어들이 굉장히 비슷하다고 여겼다. 논리적인 생각이 중요하지 언어야 상황에 따라 금방 배워서 쓸 수 있는 것이라고 믿었다. 그러나 이것은 깊이 없는 이해에서 비롯된 오만한 생각이었고, 다른 패러다임을 접했을 때 큰 충격을 받을 수밖에 없었다. 어쩌면 회사에서의 반복적인 코딩 경험이 새로운 것을 습득하는데 도움이 되었을 수는 있겠다. 아무튼 그 생각의 흐름을 언어의 발전 과정과 관련 글들과 함께 살펴보고자 한다.

많은 사람들이 그렇듯이 C 언어를 처음 배웠는데, 당시에는 C가 깊은 학문적 토대 없이 당장의 개선에만 초점을 두고 만들어졌다는 이야기는 생각도 하지 못했다. 그저 컴퓨터가 내가 원하는 문자를 출력하고, 반복을 통해 구구단을 쉽게 계산할 수 있다는 것이 고무적이었다. 그래서 C 언어의 개발자인 Dennis M. Ritchie가 쓴 “The Development of the C Language”를 흥미롭게 읽었다. 아마도 세계에서 가장 유명한 언어일텐데, 개발 과정은 전혀 어렵지 않았다. 고급 수학이나 이해할 수 없는 용어가 없었다. 이 글이 연구 논문이라기보다 기술적 개선 과정이었기 때문이었다. UNIX System Programming을 위하여 B 언어에 Floating Point 연산, Type System 등이 추가되어 C 언어의 모체가 탄생하였고, 이후로도 그때 그때의 필요성에 따라 언어가 정립되어 갔다.

C 언어는 여전히 널리 쓰일 뿐더러 특정 분야에서 무척이나 가치있는 언어이지만 이 태생적 한계로 필연적인 불편함이 수반되었다. 속도라는 무시 못할 측면 때문에 임베디드 시스템에서는 여전히 메인 언어이며, 개인적으로는 알고리즘 대회에서 사용했었다. 그러나 간단한 문자열 처리에서조차 끝값 등의 처리에 애먹은 경험은 많이들 공감할 것이다. 회사에서는 C++로 게임 서버를 구현하면서, 수많은 크래시Crash들이 잡고 보니 아주 간단한 문제들이었음을 알았다. 이 정도는 컴퓨터가 해줄 수 있지 않을까라는 생각이 들 수밖에 없었다. 실제로 그 후 프로그래밍 언어 분야는 많은 연구가 있었고 새로운 언어들이 등장하였다.

"계산이란 무엇인가에 대한 아이디어들"을 읽어보면 언어의 발전이 얼마나 어렵고 아직 미흡한지 느낄 수 있다. 같은 프로그램이 무엇인지 정의하는 것조차 생각해보면 절대로 만만한 문제가 아니다. 또한 우리는 알고 있는 논리적 계산법을 자연스럽게 받아들이고, 무의식적으로 그것이 전부라고 느낀다. 허나 지금도 새로운 계산법과 그에 기초한 프로그래밍 언어가 만들어지고 있다. 나는 상상도 하지 못한 부분이었기에 자신의 범위 밖을 생각한다는 것이 얼마나 어려운 일인지 여기서도 알 수 있었다.

"19세기 논리학, 21세기 계산학"은 프로그래밍 언어의 큰 진일보였던 부분을 잘 나타내준다. 프로그래밍 언어 수업에서 배운 내용을 정리하는 계기가 되었다. Gentzen은 Deduction의 개념을 쌓았고, Alonzo Church는 Lambda Calculus를 창시했다. 그가 도입한 함수를 함수의 인자로 사용하기, 함수의 반복 호출, 재귀 호출, 타입은 지금 생각해도 가히 혁명적이다.

처치의 Lambda Calculus는 많은 언어의 기초가 되었다. 멀티 플랫폼을 기반으로 현재 가장 널리 쓰이는 언어 중 하나인 자바Java 역시 Lambda Calculus의 개념을 도입하였다. 순수 함수형 언어들인 Lisp, ML, Haskell 등이 Lambda Calculus에서 출발하였음은 물론이다. 연구실과 일부 매니아들의 전유물이라고 생각되던 이 언어들은 현업에서도 폭넓게 쓰이고 있다.

실제 사용 예시로, 페이스북Facebook은 스팸Spam 차단에 Haskell을 사용하였다. 강력함은 널리 알려져 있었지만 소수의 매니아들만 쓰던 언어를 거대 기업에서 과감하게 도입한 것이다. 가장 큰 이유는 많은 Task를 같은 시간에 실행하는 것 - 즉 Concurrency에 특화되어 있어 스팸 처리에 효율적이었기 때문이었다고 한다. 결과는 성공적이었다. 다만 지금은 좀 더 많이 알려진 언어인 Go나 Rust를 쓴다고 한다. 사족이지만 이 유연함이 페이스북뿐만 아닌 실리콘밸리 문화일 것이고, 그것이 지금의 실리콘밸리를 만들지 않았나 싶다.

월가의 한 투자회사는 금융 시스템을 ML로 구축하기도 했다. 은행이나 증권사는 특유의 보수적 문화 때문에 메인스트림 언어만 사용하지 않을까했는데, 오히려 기술이 회사의 주가 아니다보니 독특한 시도를 해볼 수 있었다고 한다. Yeron Minsky는 CMU 강연에서 그의 회사 Jane Street이 정보분석 및 거래 소프트웨어를 Caml로 구현하였다고 말했다. 전 세계에서 짧은 시간의 미묘한 가격 차이로 이득을 보기 위해서는 정확성, 신속성이 무엇보다도 중요한데 Caml이 이 조건들을 만족시켰다는 것이다.

벤처업계의 구루Guru로 유명한 Paul Graham은 "Beating the Averages"라는 글의 전반부에서 창업 후 Lisp를 사용했던 경험을 들려주었다. 대기업은 평균 연 10% 정도의 성장을 한다. 그

러나 스타트업이 기존 기술(언어)을 그대로 사용하여 "평균적으로" 성장하면, 아주 높은 확률로 망하게 될 것이다. 50%에 한참 못 미치는 스타트업만이 살아남기 때문이다. 경쟁력을 확보하는 데는 여러 방법론이 있겠으나, Paul Graham이 택한, 그리고 IT 스타트업이 지향해야 할 방법은 차별화된 기술력이다. 그는 Lisp라는 High-level 언어를 사용하여 처음으로 거대 일반 사용자용 어플리케이션을 구축하였는데, 경쟁자들은 도저히 개발 속도를 따라올 수 없었다. 심지어 그는 해당 시점에서 가장 강력한 언어 외의 다른 어떤 선택도 실수라고 단언한다.

특정 언어가 특정 상황에 잘 맞다거나, High-level의 언어를 사용할 때 생산성이 높아졌다는 이야기도 공감하지만 무엇보다 내게 크게 다가왔던 것은 생각하는 지평의 확장이었다. 워프 Benjamin Whorf라는 언어학자는 언어가 사고를 지배한다는 충격적인 주장을 했다. 조지 오웰 George Orwell은 그의 소설 "1984"에서 언어를 축소시킴으로서 대중들의 생각을 줄이고, 절대 권력자의 권력을 강화하는 모습을 묘사했다. 이들은 여전히 논란이 있는 가설이지만, 나는 프로그래밍 언어 분야에서 비슷한 이야기를 할 수 있을 듯하다. A 언어로 a 문제를 풀고, 좀 더 어려운 a'문제를 푸는 것도 발전이다. 그러나 새로운 B 언어로 b 문제를 풀며 이전과는 전혀 다른 방식으로 생각하는 것을 볼 때가 있다.

프로그래밍 원리 수업 때 Scheme이라는 언어를 만나고, 대학에서 가장 컸던 지적 즐거움을 느꼈다. 처음에는 수많은 괄호쌍들이 기괴하게 보였지만, 코드를 데이터처럼 쓸 수 있다는 사실이 충격적이었다. 이는 분명 Scheme을 알지 못한 상태에서 사고 블럭을 쌓는 것만으로는 도달할 수 없었던 생각이라고 본다. (물론 그게 가능한 사람들도 있을 것이고, 그런 사람들이 새로운 패러다임과 새로운 언어를 만드는 것이리라.)

그러나 Paul Graham이 "Beating the Averages"에서 역설하듯이, 새로운 환경을 접하는 것은 여전히 어려운 일이다. 보통 그 원인을 귀찮음이나 변화를 두려워하는 인간 본성에서 찾는다. 그는 사고 범위 밖의 강력함은 느낄 수 없다는 것이 더 중요한 원인이라고 말한다. Blub이라는 가상의 언어를 사용하는 사람은, Blub 안에서 Blub의 방식으로만 생각하기 때문에 Blub이 충분히 좋게 느껴진다는 것이다. 이 부분에서 나는 무릎을 탁 칠 수밖에 없었다. 나 역시 이제서야 어렴풋이 공감하기 때문이었다. Paul Graham은 사람들이 잘 쓰지 않는 언어가 바로 그 점에서 경쟁력이 있으며, 사용하는 걸 두려워하지 말라고 했다.

내게는 그것이 크나큰 매력으로 다가온다. 나는 내가 만든 프로그램을 사람들이 사용한다는 것보다 프로그램을 만들 때 하게 되는 논리적 고민에서 더 큰 흥미를 느낀다. 그런데 어느 순간, 그 고민하는 방식 자체가 추가될 때의 짜릿함은 말로 표현할 수 없다. 갑작스레 키가 커서 보이지 않던, 그리고 존재조차 몰랐던 것을 볼 때의 느낌일 것이다.

흐름을 선도할 수 있으면 그 무엇보다 좋겠으나, 나 같은 소시민에게는 새로운 개념에서 오는 사고의 확장만도 충분한 즐거움이다. 다시 한 번 배움을 게을리하지 말아야겠다는 생각을 한다.