Programming language Homework 3

Exercise 1 essay

진보를 향한 움직임

- 공간정보공학 분야에서 바라본 프로그래밍 언어

지구환경시스템공학부

2003-12316 노건일

1. 서론

'프로그래밍 언어'와 처음 만난 건 15년 전이었다. PC가 가정에 보급 되기 시작하고, 컴퓨터 학원이 크게 인기를 누리던 때였다. 나도 또래 친구들과 함께 컴퓨터 학원을 다니게 되었고, 그곳에서 프로그래밍의 세계로 첫발을 내디뎠다. 언어의 종류는 DOS 환경의 BASIC이었던 것으로 기억이 난다.

우리 학부는 크게 도시, 자원, 토목공학의 세 분야로 나눌 수 있다. 나는 그 중에서도 토목공학의 한 분야인 공간정보공학을 전공하려고 한다. 공간정보공학은 측량공학에 그 뿌리를 둔 학문으로 대상물의 위치와 관련된 속성 정보들을 수집, 활용하는 방법을 연구하는 학문이다. 컴퓨터와 친해지는 게 도움이 될 것이라는 지도 교수님의 말씀에 따라, 15년 전의 설렘을 가슴에 안고 이 강의를 수강하게 되었다.

사실 수강신청을 하면서 고민이 많았다. 프로그래밍과 관련된 강의를 듣고 싶은데, 어떤 강의를 들어야 좋을지 선택하기 힘들었다. 그런데 이 강의의 계획서를 보면서 처음엔 ML이라는 접해보지 못했던 언어에 대한 부담감이 느껴졌고, 왜 많은 사람들이 사용하는 C나 JAVA같은 언어를 사용하지 않는지 의아했지만, 그 안의 내용들이 점점 나를 매혹시켰다. 결국 나는 이 강의와 프로그래밍의 원리를 동시에 수강신청 하는 과감한 결정을 내렸고, 하루하루 수업을 들으면서, 그리고 이번 에세이 과제를 위해 주어진 글들을 읽으면서 점점 그 결정이 옳았다는 생각을 갖게 되었다.

이제 그 동안의 느낀 점들을 두서없이 풀어놓으려 한다.

2. 가장 사랑 받는 언어, C

현재 가장 널리 쓰이는 언어 중 하나인 C의 역사의 시작은, 1960년대 BCPL의 개발로 거슬러 올라간다. BCPL은 '기계를 잘 작동시키는 방법'에 대한 고민에서 시작되어 이후 많은 시행착오를 겪으며, 크고 작은 단점들을 점차 개선하는 형태로 발전했다.

1970년 Ken Thompson은 BCPL의 문제점들을 개선하여 C의 전신이라고 할 수 있는 B를 만들었다. B는 모든 자료를 'word' 혹은 'cell' 이라고 불리는 고정된 길이의 비트 패턴으로 취급했다. 따라서 이 자료들은 타입에 대한 구분이 전혀 없었고, 연산자의 종류에 관계없이 어디에든 사용될 수 있었다. 심지어 A라는 문자와 B라는 문자를 서로 '더하는' 연산도 가능했다. 물론 그 결과는 아무런 의미도 없는 값(혹은 의도하지 않은 값)이 되고 만다. 이런 점은 언어에 유연성을 부여한다는 점에서 장점으로 여겨지기도 했지만, 많은 경우 작성자가 의도치 않은 형태로 자료가 흘러 다니게 되는 상황을 막을 수 없다는 문제점의 원인이 되었다.

B의 이런 문제점을 해결하기 위해 Dennis Ritchie는 C를 디자인 하며 타입 시스템을 도입한다. 이후로 C는 int, char등의 기본타입에서 시작하여 그것들을 조합한 array, pointer, function등의 다양한 타입을 지원할 수 있도록 발전해왔다.

하지만 그럼에도 불구하고 C의 타입 시스템은 여전히 불안정한데, 이는 두 가지 측면에서 기인한다.

첫째로, C의 태생적 한계성이다. 즉, C는 BCPL이라는, 타입의 구분이 없는 기계 중심의 언어를 기반으로 발전된 언어이기 때문에 본질적으로 그 한계성을 내포하고 있다.

둘째는, C가 기존 언어와의 호환성을 포기하지 않았다는 점이다. 시작부터 문제점을 안고 시작했지만, 그 문제점을 발견한 시점에서 그것을 통째로 뜯어 고치기에는 이미 너무 많은 것들이 이루어져 있었다. 타입 시스템을 제대로 구현하기 위해서는 이런 것들을 모두 포기해야 했지만 그런 결정을 내리지 못했다. 교수님께서는 이것을 '관성'의 개념을 빌어 표현하셨다. 한번 힘을 받아 움직이기 시작한 C라는 물체는 그것을 멈출 수 있는 힘을 받을 때까지 계속해서 덩치를 키워가며 움직이고 있는 것이다.

사실 C의 이런 호환성은 커다란 장점으로도 작용한다. 지금까지 수많은 사람들이 C와 호환되는 언어를 이용해 프로그래밍을 해왔고, 지금 이 순간에도 수많은 사람들이 다시

C를 사용해서 프로그래밍을 하고 있다. 그리고 이들은 만들어진 코드들을 open source library를 통해 서로 공유한다. 이것은 마치 도시가 성장하는 과정에서 수도, 도로망 등의 인프라가 구축되는 것과 흡사하다. 인프라의 구축에는 많은 비용과 시간이 필요하지만, 한번 구축되면 누구나 손쉽게 접근과 사용이 가능하다. 따라서 도시가 성장하기 위해서는 인프라가 잘 구축되어야 하고, 인프라가 잘 구축된 도시는 그만큼 성장 가능성이 높다고 할 수 있다. C라는 도시는 open source library라는 자신만의 인프라를 구축하며 계속해서 덩치를 키우고 있다. 하지만 우리는 여기에서 이런 질문을 해볼 수 있다. 크게 성장한 도시가 정말로 살기 좋은 도시인가? 사람들이 많이 사용하는 언어는 정말로 사용하기 좋은 언어인가?

3. 프로그래밍 언어가 딛고 서야 할 곳

디딜 곳 없이 설 수 있는 것은 없다. 좋은 도시를 만들기 위해서는 수많은 구조물을 버틸 수 있는 단단한 지반이 필요하다. 이를 위한 기초 공사를 무시하고 아무리 좋은 인프라를 구축한다고 한들, 살기 좋은 도시를 만들기는 힘들다. 이런 도시에서는 어느 날 자고 일어났더니 아파트 한쪽이 땅속으로 파묻혀 버릴 수도 있는 것이다. 그렇다면 프로그래밍 언어는 어떤 기초가 필요할까?

Douglas Adams가 지은 '은하수를 여행하는 히치하이커를 위한 안내서'라는 공상과학소설이 있다. 소설 속 우주 어딘가에, 고도로 진화한 지성체들이 살고 있는 행성이 있었다. 이들은 '삶, 우주, 그리고 모든 것에 대한 명쾌한 해답'을 구하기 위해 그들의 기술로 만들 수 있는 최고의 컴퓨터를 디자인한다. 하지만 그들의 기대를 저버리고 장장 750만년의 계산을 통해 컴퓨터가 내놓은 해답은 바로 42. 허탈해하는 사람들에게 컴퓨터는, 그들이 진짜 질문의 의미를 알지 못하기 때문에 이 해답을 이해할 수 없는 것이라고 말한다. 그래서 그들은 다시 진짜 질문의 의미를 찾기 위해 더욱 발전된 컴퓨터를 새롭게 디자인하는데, 이 컴퓨터는 무한하고도 미묘하게 복잡해서 유기체 그 자체가 그 작동 원리의 일부로 포함된다. 이 컴퓨터의 이름은 놀랍게도 '지구'이다.

지구의 생명체, 즉 우리 인간들이 궁극적인 컴퓨터였다는 이러한 문학적인 표현을, 단지 소설 속에서만 가능한 허무맹랑한 상상력으로 치부해버릴게 아니다.

John McCarthy(1963)는 프로그래밍의 기초를 수리 논리학에서 찾아야 한다고 말한다. 수리 논리학은 논리학을 수학에 응용하는 학문이고, 논리학은 인간의 사고방식을 연구하는 학문이므로, 결국 프로그래밍 언어는 인간의 사고에 기초하여야 한다는 것으로 해석할 수 있다. 나아가 Philip Wadler(2000)에서 이런 생각에 역사적인 타당성을 부여하고 있다는 것을 발견할 수 있다. 논리학 영역에서 발전된 Gentzen의 Natral deduction과, 기계적으로 계산 가능한 모든 함수를 표현할 수 있는 Church의 Lambda calculus가 마치 동전의 양면처럼 본질적으로 같다는 것이 1969년 Howard에 의해 증명된 것이다.

즉, Church는 인간적 사고 방식을 기계에 적용하는 방법을 찾아냈다고 할 수 있고, 그 후로 이것을 활용한 프로그래밍 언어들이 개발되는 과정이 이어졌다. 실제로 Scheme, ML, Haskell 등의 언어들이 Church의 이론을 기초로 해서 만들어졌고, 지금 Programming Language 수업의 실습언어로 활용되는 OCaml도 ML의 한 갈래이다.

그렇다면 이렇게 인간을 토대로 한 논리적 기초를 기반으로 개발된 ML등의 언어는 C를 필두로 한 기계의 관점에서 개발된 언어에 비해 어떠한 장점을 갖는 것일까?

먼저, 스스로 타입을 추론하여 Semantics를 검증하는 기능을 들 수 있다. C가 기존의 타입을 구분하지 않는 언어(BCPL, B)에서 타입 시스템을 갖춘 언어로 발전된 이유는 프로그램의 안전성을 획득하기 위함이다. 따옴표(‘’)를 실수로 한 개 적지 않는 등의 문법적(syntax) 오류를 바로잡아 주는 기술은 현재 거의 모든 프로그래밍 언어에서 완성된 형태로 적용되고 있다. 하지만 문법적으로 완벽한 프로그램일지라도 작성자의 의도대로 실행된다는 보장은 결코 할 수 없다. 이를 위해 고안된 기술이 바로 타입 검증이다. 이 기술은 어떤 프로그램 안의 특정한 위치에서 그곳에 적합한 타입의 데이터가 들어오는지를 검사함으로써 프로그램이 작성자의 의도대로 실행될 지의 여부를 실행하기 전에 미리 알 수 있도록 도와준다. 따라서, 이를 위해서는 정확한 타입 시스템이 필수적이다. 하지만 위에서도 언급했듯이, C의 타입 시스템은 불안정하다. 게다가 작성자가 의도적으로 타입을 비틀어버리는 경우도 많이 있는 만큼, C에서는 타입 검증 기술이 구현되기 힘들다.

둘째로, 이해하기 쉬운 코드를 작성할 수 있다는 점이다. 이것은 함수에 대한 개념의 변화에서 발생한다. C와 기존의 언어에서 함수는 굉장히 특별한 존재였다. 하지만 사실 인간의 사고에서 함수는 그리 특별한 존재가 아니다. 이것은 수학이라는 탄탄한 기초 위에서 증명될 수 있다. 그리고 이러한 생각이 Lambda calculus에 의해 프로그래밍 언어로 이식됨으로써, 함수를 기계들의 방식이 아닌 인간의 방식으로 다룰 수 있게 되었고, 이제서야 인간이 사고하는 방법을 그대로 활용해서 프로그래밍을 할 수 있게 된 것이다.

결국 이러한 차이를 만든 결정적 요인은, 수학과 논리학이라는 기초이다. 더 진보된 언어의 개발자들은 그렇지 않은 언어의 개발자보다 더 충실하게 기초를 다졌다. 누구나 높은 곳으로 가기를 희망한다. 하지만 더 높은 곳으로 힘껏 도약하기 위해서는, 먼저 더 낮은 곳으로 눈을 돌려 자신의 발 밑이 충분히 단단한지 확인해 보아야 한다. 수천 년에 걸쳐 다져온 수학과 논리학의 단단한 기초 위에서 도약한 언어가 더 뛰어난 것은 너무나 당연해 보이는 사실이다.

4. 진보를 향한 움직임

그런데 왜 사람들은 진보된 언어를 더 선호하지 않는 걸까? 진보된 언어가 정말 더 안전하고 사용하기 쉽다면 왜 아직까지 많은 사람들이 기존의 언어를 고수하는 걸까?

그 이유는 앞에서 살펴봤던 대로, C와 같은 기존의 언어에 대한 관성이 아직 작용하고 있기 때문이라고 할 수 있다. 지금까지 C가 구축한 방대한 인프라는 아직도 많은 사람을 끌어들이는 매력요인으로 작용한다.

Yaron Minsky(2009)도 Jane Street이라는 무역회사에서 사용할 소프트웨어를 개발하면서 이러한 점에 대해 고민을 했다. 그는 소프트웨어 개발에 사용할 언어를 선택해야 했다. 만약 C와 같은 언어를 사용하면 기존의 많은 open souece를 이용할 수 있지만, ML등의 언어를 사용한다면 거의 처음부터 모든 것을 손수 해야 했다. 하지만 그는 결국 안전성과, 변화하는 환경에 잘 대응할 수 있다는 점 등을 이유로 ML의 한 갈래인 OCaml을 소프트웨어 개발의 핵심 언어로 선택했다.

사실 이 밖에도 HP사의 HOL, Austrailia DSTO의 Isabelle의 개발사례를 포함하여 이러한 사례는 많이 있고, 또한 점점 늘어나고 있다. 다시 말하면 C의 관성에서 벗어난 사람들이 점점 늘어나고 있고, 그들은 이제 오히려 그 관성의 방향을 돌리는 쪽으로 작용할 것이다. 그리고 불과 수십 년 동안 만들어진 기존 언어들의 인프라는, 수천 년간 만들어진 수학과 논리학이라는 거대한 인프라 앞에서 오히려 사소한 것에 불과하다는 사실이 이를 뒷받침한다.

이처럼 지금까지 C를 향해있던 관성은, ML과 같이 진보된 형태의 언어의 더욱 커다란 관성에 흡수될 것이며, 진보를 향한 이러한 움직임은 계속해서 ML을 지나 더 나은 형태의 언어로 나아갈 것이다. 그리고 이것은 다시 미래에 나올 더욱 진보된 새로운 언어들의 궁극적인, 그리고 공통적인 운명이라고 할 수 있다.

5. 결론

공간정보공학은 컴퓨터 공학의 영향을 크게 받았다. 컴퓨터의 활용으로 인해 전통적인 측량과 지도 제작에서 벗어나 과거에는 상상할 수도 없을 만큼의 방대한 정보를 실시간으로 수집하고, 처리할 수 있게 되었다. 과거에는 각을 정교하게 잴 수 있는 좋은 측량장비와 지도를 그리기 위한 좋은 종이가 필요했다면, 지금은 방대한 양의 자료를 적절히 가공하여 유의미한 정보를 얻어낼 수 있는 알고리즘을 담고 있는 좋은 소프트웨어가 필요하다. 그리고 이런 소프트웨어는 반드시 좋은 프로그래밍 언어가 뒷받침 되어야 만들 수 있다.

그런 이유로 굉장히 흥미롭고, 진지하게 살펴본 프로그래밍 언어의 발전과정은 나를 상당히 고무시켰다. 프로그래밍 언어는 성공과 실패, 다양한 시행착오와 오류로부터 시작하여 수학과 논리학, 인간의 사고력을 바탕으로 자신만의 독특한 개념을 가진 독자적인 학문 분야로 발전해 나가고 있었다. (이광근, 1999) 어제에 대한 고민을 잊지 않은 채, 더 나은 내일을 바라며 끊임없이 진보적인 움직임을 보여주고 있었다.

나는 이번 에세이를 통해 이러한 과정을 기쁘게 지켜볼 수 있었고, 앞으로도 계속해서 지켜보며 응원하고 싶다. 나아가 공간정보공학도 더 나은, 진보를 향한 더 커다란 움직임에 동참하기를 바라고 나 자신이 그 선두에 설 수 있는 사람이 되길 염원한다.

6. 참고문헌

Dennis M. Ritchie(1993), The Development of the C Language, (HTML) HOPL-II

Robin Milner(1993), An Interview with Robin Milner, Communications of the ACM Vol.36(1)

John McCarthy(1963), A Basis for a Mathematical Theory of Computation

Philip Wadler(2000), Proofs are Programs: 19th Century Logic and 21st Century Computing

이광근(1999), 계산이란 무엇인가에 대한 아이디어들

Yaron Minsky(2009), Yaron Minsky's talk at CMU