Programming language Homework 6

Exercise 1 essay

아름다움을 찾아 떠나는 여행자

지구환경시스템공학부

2003-12316 노건일

1. GOTO 100

두 번째 에세이를 쓰기 위해 주어진 글을 읽다가, 문득 잊고 있었던 옛 기억이 떠올랐다. 그것은 15년 전 BASIC이라는 언어를 배울 때 받았던 가르침이었다.

'프로그래밍을 하면서 각 줄마다 항상 10단위의 번호를 매겨라.'

이미 쓰여진 줄 사이에 새로운 줄을 삽입할 경우를 대비해 10단위로 번호를 매기라는 것이고, 이 번호들은 프로그램 실행 중 특정한 줄로 바로 이동할 수 있게 해주는 GOTO문을 위한 것이었다. 그런데 그로부터 불과 10여 년이 흐른 지금, 그 존재조차 까맣게 잊고 있을 만큼 10 단위의 줄 번호와 GOTO문은 과거의 유물이 되어버렸다. 이것들은 왜, 어디로 사라진 것일까?

2. 좋은 언어

Paul Graham(2002b)의 에세이는, 무능력하고 꽉 막힌 상사(Pointy-haired boss)들이 프로그래밍 언어에 대해 갖고 있는 잘못된 생각을 비판하며 시작한다. 그들은 모든 언어가 똑같은 능력을 갖

고 있으므로 기왕이면 많은 사람들이 사용하는 '표준의' 언어를 사용하는 것이 옳다고 생각한다. 그래야 내 밑에서 일하던 프로그래머가 '아무 이유 없이' 그만두더라도, 쉽게 다른 사람으로 교체할 수 있으니 일석 이조이다.

그런데 그들의 생각은 일견 타당해 보인다. 현대의 프로그래밍 언어들은 Turing completeness를 만족한다. 즉, 어떤 언어를 사용하더라도 컴퓨터로 실행 가능한 어떤 프로그램이든 만들어낼 수 있다는 것이다. 그렇다면 굳이 사람들이 많이 사용하는 '좋은' 언어를 두고 '이상한' 언어를 사용할 필요가 있을까?

잠시 복잡한 얘기에서 벗어나, 100m 달리기 시합을 해보기로 하자. 그런데 주변에 마땅한 트랙을 찾을 수 없다. 그래도 우리는 쉽게 포기하지 않는다. 우리가 직접 100m 달리기 경기를 위한 트랙을 만들기로 하자. 가장 먼저 해야 할 일은 직선으로 정확히 100m의 거리를 재는 것이다. 이 때 우리는 여러 가지 방법을 사용할 수 있을 것이다. 1m짜리 줄자를 사용할 수도 있다. 혹은 둘레의 길이를 정확히 알 고 있는 바퀴를 똑바로 굴려, 바퀴 수를 셀 수도 있을 것이다. 지루한 작업이 되겠지만 어쨌든 우리에게 충분한 시간이 주어진다면, 우리는 목적을 달성할 수 있을 것이다. 하지만 여기에 또 다른 매력적인 대안도 존재한다. 전자기파의 반사특성을 활용하여 자동으로 대상 점까지의 거리를 측정해주는 광파거리측정기가 그것이다. 만약 전문가가 이 도구를 사용한다면 그는 몇 분 안에 정확히 이곳으로부터 100m 떨어진 지점을 찾아줄 것이다. 만약 측정해야 하는 거리가 더 길어지거나 장애물 등의 존재로 광파거리측정기의 적용이 곤란하다면, 더욱 발전된 기술인 GPS 측량기술을 활용할 수도 있다.

1m 줄자와 광파거리측정기, GPS 측량기술 중 어떤 것을 사용하더라도 우리는 모두 같은 일을 할 수 있다 – 거리를 재는 것. 하지만 줄자만 써왔던 사람의 눈에는 굉장히 '이상한' 도구인 광파거리측정기와 GPS는, 사실은 훨씬 더 효율적으로 같은 목적을 달성할 수 있게 해주는 매우 강력한 도구인 것이다. (1m짜리 줄자로 서울에서 부산까지의 거리를 재는데 얼마나 걸릴까? GPS 측량기술을 사용하면 이 작업은 불과 몇 초 만에 끝난다.)

조금은 뜬금없었던 이 이야기는 프로그래밍 언어에도 그대로 적용될 수 있다. 프로그래밍 언어는 많은 사람들의 노력으로 계속 발전해왔고, 결코 꽉 막힌 상사가 단순히 생각하는 것처럼 모든 프로그래밍 언어가 똑같은 능력을 갖는 것은 아니다. 다른 언어와 비교했을 때 상대적으로 더 좋

은 언어가 존재하는 것이다.

Paul Graham(2002b)은 자신의 에세이를 통해 이러한 '좋은' 언어가 바로 Lisp이라고 말한다. 그는 Lisp이 갖고 있는 장점을 다음과 같이 말한다.

- 함수타입의 존재
- 재귀함수
- 동적 타입 검증
- Garbage-collection
- 식으로 표현된 프로그래밍 (포트란 등에서 식과 명령문을 구분하는 것과 달리)
- 심벌(Symbol) 타입의 존재
- 나무구조와 심벌, 상수를 사용하는 코드에 사용되는 표현기법의 존재

이러한 점들은 Lisp을 더 발전된 언어로 만들었고, 결국 다른 언어들이 이러한 점들을 따라잡기 위해 노력하도록 만들었다. 실제로 먼저 언급된 5가지 장점은 현재 대부분의 언어들에 채택되어 그것들을 과거에 비해 더 발전된 모습으로 바꿔놓았다. 하지만 여전히 Lisp은 다른 언어들이 갖지 못한 장점을 많이 갖고 있다고 한다. 그렇다면 저자의 주장대로 정말 Lisp이 가장 좋은 언어일까?

Mark-Jason Dominus(1999)에서는 또 다른 좋은 언어가 소개된다. 바로 우리가 수업시간에 배우고 있는 ML이다. 이곳에서는 ML의 장점으로 타입 검증 기술을 들고 있다. 위에서 살펴본 대로 Lisp이 동적 타입 검증 기술을 갖고 있는데 비해, ML은 정적 타입 검증 기술을 갖추고 있다. 이는, 주어진 코드를 컴파일 하기 전에 코드 안에 포함된 모든 값들의 타입을 자동으로 유추하여 그 일관성을 검사하는 기술을 말한다. 사실 정적 타입 검증 기술은 C나 Pascal 등의 언어에서 먼저 시도되었다. 하지만 이 언어들은 이 기술을 제대로 구현하지 못했고, ML이 그것을 더욱 강력하고 발전된 형태로 만들었을 때 드디어 진정한 의미의 정적 타입 검증 기술이 세상의 빛을 볼 수 있었다. 그리고 이는 오히려 Lisp의 동적 타입 검증과 비교해보더라도 많은 장점을 갖고 있다. 그 중 가장 두드러지는 장점이 바로 프로그램의 오류를 실행 전에 미리 찾아낼 수 있다는 것이다. 이는 프로그래머가 더욱 안전한 프로그램을 만들 수 있도록 도와준다는 점에서 Lisp보다 뛰어나다.

그렇다면 Lisp이 아니라 ML이 가장 뛰어난 언어인 것일까? ML보다 더 좋은 언어는 없을까?

만약 누군가 '아니, ML이야말로 가장 뛰어난 언어다.' 라고 대답한다면 그것은 사실이 아니다. 혹은 가까운 미래에 그것은 사실이 아닐 것이다. 왜냐하면 앞에서도 언급했듯이 프로그래밍 언어는 지금까지 발전해왔고, 앞으로도 계속해서 발전해갈 것이기 때문이다.

하지만 모든 언어가 발전할 수 있는 것은 아니다. Guy Steele, Jr. (1998)는 언어의 발전이 어떻게 이루어질 수 있는지를 설명한다. 그에 따르면, 성장할 수 있도록 만들어진 언어가 발전할 수 있다. 오히려 처음부터 모든 것이 완벽하게 짜여져 있는 것처럼 보이는 거대한 언어는 오히려 도태될 가능성이 높다는 것이다. 보통 어떠한 언어의 최초 디자인에는 소수의 개발자만이 관련된다. 그들이 처음부터 끝까지 모든 것을 완벽하게 만드는 것은 불가능하다. 작지만 탄탄한 기초를 갖고 있는, 성장할 수 있는 언어를 만들어 그것을 많은 사용자들에게 던져주는 것이 그들이 선택할 수 있는 최고의 방법이다. 사용자들의 불만과 요구는 언어가 성장하는 원동력이 되고, 더욱 열성적인 사용자들은 스스로 그 언어를 발전시키기 위한 작업에 참여할 것이다. 결국 제 2, 제 3의 ML이 탄생하고 우리는 점점 더 좋은 언어를 사용해서 프로그래밍 할 수 있게 될 것이다.

3. 좋은 언어의 사용

다시 한번 Paul Graham(2002b)이 싫어하는 꽉 막힌 상사들을 생각해보자. 이들은 좋은 언어에 대해 알지 못한다. 혹은 알고 있더라도 그 필요성을 무시해버린다. 그저 지금까지 어떻게든 사용해 왔던 낡은 도구만을 고집한다. 왜냐하면 아직도 많은 사람들이 그 도구를 사용하고 있기 때문이다. 결국 이들은 주류언어(C나 JAVA같은)에 대한 관성을 상징한다.

하지만 Paul Graham(2001)은 이 관성을 보기 좋게 역이용한 사례를 소개하고 있다. 지금은 Yahoo! Store로 이름이 바뀐 Viaweb의 이야기이다. Viaweb은 사용자들이 웹 기반의 온라인 쇼핑몰을 운영할 수 있도록 지원하는 프로그램이다. 당시 대부분의 경쟁자들이 C++나 Perl과 같은 언어를 사용하는 것에 반해, Viaweb의 개발자들은 과감히 Lisp을 선택했다. 그들은 좋은 언어를 사용해서 경쟁자들보다 더 빠르게 더 좋은 기능을 구현할 수 있었고, 경쟁자들의 입장에서는 그들이 어떤 엄청난 비밀병기를 숨기고 있는 것처럼 보였다. 그리고 그것은 사실이었다. Lisp이 바로

그들의 비밀병기였던 것이다. 결국 그들은 관성에 갇혀있던 모든 경쟁자를 제치고, 엄청난 속도로 성장하여 압도적인 시장 점유율을 달성한다.

James Hague(1999)에 또 다른 사례가 있다. Crash Bandicoot이라는 게임은 Lisp과 유사한 형태의 좋은 언어로 만들어졌다. 그 덕분에 등장인물은 기존 게임에서는 불가능하다고 생각되던, 달리고, 날고, 하늘에서 땅 위의 적을 공격하고, 보트를 운전하고, 물 속을 수영하는 등 엄청나게 복잡한 동작을 수행할 수 있었다. 결국 이 게임은 천만 부 이상이 팔려나가는 큰 성공을 거두었다.

과연 Viaweb이나 Crash Bandicoot의 개발자들이 좋은 언어를 사용하지 않았더라도 그러한 성공을 거둘 수 있었을까? 이들이 다른 많은 경쟁자들처럼 주류인 언어를 고수했다면 결과는 어떻게 달라졌을까?

Paul Graham(2001)은 관성에 갇힌 사람들을 신경 쓸 필요는 없다고 말한다. 오히려 신경 써야 할 상대는 관성에서 벗어나 좋은 언어에 대해 잘 알고 있는 사람이고, 그는 정말로 위험한 경쟁자라고 말이다.

4. 아름답게

창조, 무언가를 만드는 일은 참 즐겁다. 그래서인지 많은 사람들은 끊임없이 창조활동을 한다. 화가는 그림을, 작곡가는 음악을, 요리사는 요리를, 수학자는 정리를, 공학자는 기술과 발명품을 만든다. 하지만 무엇인가를 잘 만드는 일은 어렵다. 수십, 수백 개의 그림을 그려야 좋은 화가가 될 수 있고, 수 많은 실험과 실패를 거쳐야 훌륭한 공학자가 만들어 진다. 그리고 그보다 어려운 것이 아름다운 것을 만드는 일이다.

하지만 우리 주변을 살펴보면, 아름답게 만들어진 것들을 의외로 많이 찾아볼 수 있다. 레오나르도 다빈치의 모나리자, 신라의 석굴암, 아인슈타인의 , 유선형의 스포츠카 등등.

이런 것들은 어떻게 만들어질 수 있었을까? 여기에 너무나 유명한 명언이 있다.

'Simple is the Best'

아름답고 위대한 것은 단순해 보인다. 하지만 그것이 단순해지기까지의 과정은 결코 단순하지

않다. 아름다움의 창조자들은 지금 그들에게 주어진 것이 아름답지 않다는 문제를 발견한다. 그들은 시간의 흐름에 휩쓸리지 않는 아름다움을 갈망하며, 자연을 관찰하고 그 안의 법칙을 발견하려고 애쓴다. 남들이 보기엔 이상한 행동을 하기도 하고, 때로는 상식을 거스르는 과감한 결단을 내려야 할 때도 있다. 그리고 이렇게 만들어진 것 역시 아직 충분히 아름답지 못하다는 사실을 깨닫는다. 그는 다시 갈망하고, 고뇌한다. 이러한 과정을 겪어서 탄생한 단순함이야말로 아름답고 위대한 것이다.

프로그래밍 언어의 발전과정도 마찬가지이다. 누군가 기존의 언어가 아름답지 않다고 생각했고 그것은 그 언어의 발전, 혹은 새로운 언어 창조의 원동력이 되었다. 많은 고민과 노력 끝에 만들어진 아름다운 언어는 그 사용자에게 단순함을 허락한다. 더 이상 복잡한 기계적 사고에 머물지 않고 고차원의 논리 영역에서 프로그래밍할 수 있도록 해주는 것이다. 그리고 이러한 아름다운 언어를 사용할 때, 또 다시 아름다운 창조물이 나올 수 있다. 고객의 요구에 신속하게 대응하는 쇼핑몰과, 실제와 같이 복잡한 동작을 수행하는 영웅 캐릭터는 아름다운 언어를 사용했기에 만들 수 있었다.

그리고 누군가는 지금의 언어가 아직 충분히 아름답지 못하다고 생각할 수 있다. 이들은 다시 새로운 언어를 만드는 원동력이 될 것이다. 진정한 아름다움을 찾기 위한 여정은 아직 많이 남아 있다. 그 여정은 쉽진 않겠지만, 그 여행자들에게 진정한 즐거움과 기쁨을 선사할 것이다.

5. 참고문헌

Paul Graham(2001), 치고나가기/Beating the Averages

Paul Graham(2002a), 만드는 사람의 안목/Tastes of Makers

Paul Graham(2002b), 우리들의 보복/Revenge of the Nerds

Guy Steele, Jr. (1998), 프로그래밍언어 키우기/Growing a Language

James Hague(1999), Toward Programmer Interactivity : Writing Games In Modern Programming Languages

Mark-Jason Dominus(1999), Strong Typing