프로그래밍 언어
Homework 6 : 논술 에세이 II

컴퓨터공학부 2005-11828
이소민

1. 서론

에세이 과제를 두 번째로 받으면서 이번에는 또 어떤 것에 관련된 내용을 읽게 될까 두근두근하는 마음이 있었다. 읽어야 하는 글 중 학술적이고 어려운 내용이 많았던 지난 번 과제와는 다르게 이번 과제에서 읽게 된 글들은 보다 비학술적이고 가벼운 마음으로 읽을 수 있는 글들이었던 것 같다. 또한 이해하기도 쉽고 저자들이 모두 글 쓰는 솜씨가 대단했기 때문에 그저 순수하게 글을 읽는 즐거움도 누릴 수 있었다.

에세이는 서론을 포함하여 다섯 부분으로 이루어져 있는데, 먼저 'Beating the Average'와 'Revenge of the Nerds', 'Toward Programming Interactivity'에 대한 내용 요약과 감상을 정리하고 그 다음으로는 'Strong Typing'에 대한 내용 요약과 감상과 'Growing a Language'에 대한 감상, 마지막으로 'Taste For Makers'에 대해 정리한 후 결론으로 마무리하려고 한다.

2. High-level Programming Language

'Beating the Averages'와 'Revenge of Nerds', 그리고 'Toward Programmer Interactivity : Writing Games In Modern Programming Languages'의 세 글은 모두 주로 high-level programming language의 문제, 그 중에서도 특히 LISP에 대해 다루고 있다.

먼저 'Beating the Averages'의 저자는 자신의 소프트웨어를 LISP를 이용하여 개발해서 경쟁자들을 따돌렸던 경험담을 이야기하고 있다. 실제 경험에서 나온 이야기이기 때문에 즐겁게 읽을 수 있었고 와 닿는 부분도 많았는데, 특히 공감이 갔던 부분은 "사람들은 자신이 우연히 사용하게 된 언어가 충분히 괜찮다고 생각하는 경향이 있다 (Whatever language people happen to be used to, they tend to consider just good enough)"라는 말이었다.

나는 프로그래밍 경험이 그렇게 많은 편은 아니지만 이번에 프로그래밍 언어 수업을 들으면서 새로 nML에 대해 공부해야 했을 때, '도대체 이것을 왜 해야 하지?'라는 생각을 했던 것이 사실이다. 여태까지는 C나 C++ 아니면 java로 주어진 문제들을 충분히 풀 수 있었는데 말이다. 그 정도의 언어면 충분하지 않은가?

그러나 nML로 과제를 해나가면서 내 생각이 얼마나 틀렸었는지, C로 못 하는 것들이 얼마나 많은지, nML이 얼마나 강력한 언어인지 뼈저리게 깨달을 수 있었다. 아니 저자의 말대로 모든 언어는 'Turing equivalent'하므로 물론 과제를 C로 하는 것도 가능은 했을 것이다. 그러나 그런 과제가 나왔다면 아마 당장 수강 취소원을 내지 않았을까. 같은 저자가 'Revenge of the Nerds'에서 인용하고 있는 것처럼 "Any sufficiently complicated C or Fortran program that contains an ad hoc informally-specified bug-ridden slow implement-ation of half of Common Lisp."이기 때문이다. 아마 여기서 Lisp를 nML로 바꿔 놓아도 무방할 것이다.

저자의 말 중에 가장 인상적이었던 것은 "일반적으로 기술은 빠르게 변화한다. 그러나 프로그래밍 언어는 다르다 : 프로그래밍 언어는 단순한 기술이 아니라 사람들이 생각하는 방식이다. (programming languages are not just technology, but what programmers think in) 그것은 반은 기술이지만 반은 종교다."라는 말이었다. 즉 그의 말에 따르면 단지 우리가 언어를 이용해서 프로그래밍을 한다는 것이 아니라 우리가 사용하는 언어가 우리의 생각을 결정짓

는 다는 것이다.

이 또한 nML로 과제를 하면서 몸으로 느낄 수 있었던 것인데 처음의 나는 C에 너무나 익숙해져 있었기 때문에 'nML 적으로' 사고하는 것이 힘들었다. 두 언어가 단지 문법의 차이를 익히는 것뿐만이 아니라 전혀 다른 사고방식을 요구했기 때문이다. 그러나 점차 과제들을 해나가면서 nML의 방식에 익숙해지게 되었고 그러자 이번에는 C가 갑자기 어색하게 느껴지게 되어버렸다. 만약 언어가 단지 '많은 것 중에서 어느 것을 사용하는가'의 문제였다면 이런 현상은 벌어지지 않았을 것이다.

'Revenge of the Nerds'에는 'Beating the Averages'에서 저자가 전개했던 생각들이 좀더 자세하게 펼쳐 지는데, 즉 "모든 프로그래밍 언어는 동등하지 않다. (All languages are not equivalent)"라는 것이 그것이다. 그리고 그는 가장 강력한 언어로 LISP를 꼽는데 LISP에 대한 지식이 기본적인 상식 밖에 없는 나로서는 그의 의견에 대해 맞다 혹은 틀리다고 판단할 수 있는 자격이 없는 것 같다. 하지만 그가 이야기하는 LISP만의 특징들이 대단히 흥미로워 보이는 것은 사실이다. 특히 매크로에 대한 이야기는 시간만 난다면 LISP에 대해 한 번 공부해보고 싶다는 생각을 불러 일으키기에 충분했다.

물론 저자가 그렇다고 모든 문제를 LISP나 그와 같은 high-level programming language로 해결해야 한다고 말하는 것은 아니다. 그 이외의 언어로 해결해야 하는 문제들도 분명 존재한다. 저자가 이야기하는 것은 결국 '어려운' 문제들에 대한 것이다. 마지막에 그가 던지는 말은 위트 있으면서도 뼈가 담긴 말인데 즉 어려운 문제(problem)을 해결해야 할 때 문제(question)는 충분히 강력한 언어를 쓰느냐 쓰지 않느냐가 아니라 (a) 충분히 강력한 언어를 쓸 것이냐, (b) 이를 위한 사실상의 인터프리터를 짤 것이냐, (c) 그냥 스스로 인간 컴파일러가 될 것이냐라는 것이다. 우리는 그냥 인간 컴파일러가 되기를 선택한 적이 너무 많지 않던가?

마지막 글인 'Toward Programmer Interactivity'는 high-level programming language로 게임을 짜는 등의 일을 할 수도 있다는 내용을 담고 있었는데, 한 번도 그런 일에 대해 들어본 적이 없었기에 신선하기는 했지만 위의 두 글에 비하면 읽는 재미는 좀 떨어졌던 것 같다.

3. Strong Typing

'Strong typing'은 '과연 타입시스템이 필요한가?'라는 의문을 시작으로 하여 여러 프로그래밍 언어들의 타입 시스템에 대한 내용을 다루고 있다. 먼저 Fortran과 C등의 타입 시스템에 대해 간단하게 살펴본 후에 ML의 타입 시스템에 대해 자세히 다루고, 마지막에 Perl의 타입 시스템에 대해 간단히 언급하고 있다.

nML을 쓰면서 nML의 타입 시스템이 얼마나 훌륭한지는 항상 느끼고 있었지만 이렇게 정리해서 놓고 보니 새삼스레 ML의 타입시스템이 얼마나 똑똑한 것인지 깨달을 수 있었다. 그리고 그에 비해서 C의 타입 시스템이 얼마나 엉터리였는지도. 사실 C에서 타입 에러가 나면 도대체 왜 어디서 무슨 에러가 어떻게 난 건지 이해할 수 없었던 적이 한 두 번이 아니었다. 그런데 그런 점을 나만 느꼈던 것이 아니라는 점은 일말의 위안을 주었다.

글에서 이야기하는 ML의 장점들은 정리하자면 다음과 같다. 첫 번째, 타입 에러들이 많이 일어나지만 이 타입 에러들은 진짜 문제를 가르쳐준다. 여태까지 nML 프로그램을 하면서 느낀 것이지만 nML에서 에러가 발생했다고 하는 경우, 그것은 <정말로> 프로그램을 잘못 짰다는 것이다. 즉 컴파일러가 어떤 바보 같은 생각을 했을까 하고 생각하는 것보다는 내가 무슨 바보 같은 짓을 했을까 하고 생각하는 편이 빠르다는 것이다.

두 번째로, 프로그램을 짤 때 단지 머리 속에 떠오르는 함수의 정의를 써 내려가기만 하면 된다는 것이다. 처음에 mML을 접했을 때에는 이게 굉장히 불안하고 당황스러웠다. C나 다른 많은 언어들을 사용하면서 변수들의 타입을 지정하는 것을 너무나 당연한 것으로 생각하게 되었기 때문이다. 변수의 타입에 대해 전혀 언급하지 않아도 알아서 타입을 체크해 준다니 도대체 어떤 원리로 하는 것인지, 그게 말이나 되는 것인지, 도무지 상상조차 할 수 없었다. 그러나 점점 nML에 익숙해지면서 타입을 일일이 써줄 필요가 없다는 것이 얼마나 편리한지를 깨달을 수 있었다. 사실 생각해보면 값으로 타입을 얼마든지 유추할 수 있는데 타입을 일일이 써 줘야 한다는 것이 얼마나 비효율적이고 부정확한 일인가? 심지어는 언어에 상관없는 숙제를 하면서 오랜만에 C를 잡았다가 컴파일이 안 되어 코드를 다시 보니 타입 선언을 하나도 해주지 않았다는 것을 깨달은 적도 있었다. 결국 그 때의 숙제는 nML로 한 기억이 있다.

마지막으로 저자는 Perl의 타입 시스템에 대해 간단히 언급하고 있는데 이

부분은 내가 Perl에 대해 거의 아는 것이 없다 보니 잘 와 닿지 않았다. 하지만 ML과는 전혀 다른 방식으로 쓸 만한 타입 시스템이 존재할 수 있다는 것, 그 마다 각기 다른 장점이 있다는 이야기였던 것 같다.

4. Growing a Language

'Growing a Language'는 프로그래밍 언어를 만들어가는 과정에 관한 글이다. 글의 주제는 다음과 같은 한 문장으로 정리할 수 있을 것이다 : 다른 사람이 프로그래밍을 짤 수 있도록 도와주려면 작은 프로그래밍 언어를 설계하는 것이 좋은가 아니면 큰 프로그래밍 언어를 설계하는 것이 좋은가? 저자의 결론은 성장할 수 있는 언어를 만드는 것이 가장 바람직하다는 것이다. 즉, 언어는 작게 시작해서, 이용자들이 증가함에 따라 성장해야 한다는 것이다.

저자가 바람직한 형태로 생각하는 프로그래밍 언어의 성장은 많은 사람들이 함께 언어를 성장시켜나가면서 적은 인원의 관리자들이 꼭 필요한 관리만을 하는 그런 양상이라고 생각된다. 그는 여기서 대성당과 바자의 비유를 드는데 즉 성당은 많은 사람들이 함께 짓지만 그 설계자는 단 한 명인 반면 바자에는 미리 정해진 계획이 없고 각 사람마다 자신이 하고 싶은 일을 한다는 것이다. 저자가 바람직하게 생각하는 방식은 아마도 대성당보다는 바자 쪽에 더 가까울 것이다.

하지만 이 글에서 글 자체보다 더 눈을 끄는 것은 바로 저자가 글을 써 나간 방식이었다. 저자는 한 음절로만 이루어진 단어를 primitive로 정의하고 primitive가 아닌 단어는 쓸 때마다 새로운 definition을 내놓으면서 써 나가는데, 이런 식으로 글을 써 나가는 방식은 어딘가 프로그래밍 언어로 프로그래밍을 하는 것과 묘하게 닮은 느낌을 준다. 저자의 말대로 프로그래밍 언어가 아무리 복잡해 보여도 실제 언어에 비하면 프로그래밍 언어는 엄청나게 작은 언어이기 때문이다. 그래서 스스로 쓸 수 있는 단어를 제한하고 써 나가는 저자의 글은 어딘가 부자연스럽고 일반적인 상황이라면 다른 단어가 쓰일 것 같은 상황에서도 특이한 단어를 쓰고 있다. (예를 들면 defect나 weakness 같은 단어를 써도 좋을 곳에서 굳이 wart 같은 단어를 쓰고 있다) 굳이 이런 방법까지 써야 했을까는 좀 의문이지만, 그럼에도 재미있는 시도라는 점에서는 변함이 없다. 저자는 마지막에 "높은 위치에 있는 사람들이 한 음절로 되지 않

은 단어는 말할 때마다 정의를 하고 말해야 한다면 괜찮은 일일 것이다"고 말하고 있는데 우리나라의 경우는 한 음절이 아닌 다른 방법을 써야 하겠지만 그래도 역시 해볼 만한 일이라고 생각된다.

5. Taste for Makers

'Taste for Makers'는 언뜻 보면 도대체 프로그래밍 언어 강좌에서 왜 이런 글을 읽어야 할까 하는 생각이 들 정도로 거의 인문학적인 분위기를 담고 있으면서도 굉장히 생각해볼 만한 여지를 많이 주는 글이었다. 이 글과 함께 'Beating the Averages'와 'Revenge of Nerds'의 저자인 Paul Graham이 단순히 글 솜씨뿐만 아니라 풍부한 인문학적 교양을 갖추었다는 것도 확인할 수 있었다. (홈페이지에 나와 있는 저자 약력을 보면 피렌체에서 미술까지 공부했다고 하니 부럽지 않을 수 없다.)

글의 내용을 간략히 정리해 보면 '좋은 설계(디자인)란 어떤 것인가'에 관한 것인데, Paul Graham이 뽑고 있는 좋은 설계의 특징들은 다음과 같다. 1. 좋은 설계는 간단하다. 2. 좋은 설계는 시간을 뛰어넘는다. 3. 좋은 설계는 알맞은 문제를 해결한다. 4. 좋은 설계는 암시적이다. 5. 좋은 설계는 약간 우습기도 하다. 6. 좋은 설계는 어렵다. 7. 좋은 설계는 쉬워 보인다. 8. 좋은 설계는 대칭을 이용한다. 9. 좋은 설계는 자연을 닮는다. 10. 좋은 설계는 재설계이다. 11. 좋은 설계는 모방할 수 있다. 12. 좋은 설계는 가끔 이상하다. 13. 좋은 설계는 무더기로 등장한다. 14. 좋은 설계는 가끔은 대담무쌍한 것이다.

그러면서 그는 자연과학의 이론들뿐만 아니라 예술과 문학의 여러 가지 좋은 작품들을 예로 든다. 예를 들어 좋은 설계는 무더기로 등장한다는 것에 대해 이야기하면서 그는 르네상스 시대 피렌체에 대해 이야기하는데 우피치 미술관 앞에 한번 가서 그곳에 서 있는 피렌체 출신 사람들의 동상을 구경한 사람이라면 아마 인정할 수 밖에 없는 일일 것이다. 또한 좋은 설계는 약간 우스꽝스럽다는 이야기를 하면서 로마의 판테온을 예로 드는데 개인적으로도 판테온은 세상에 존재하는 모든 건축물들 중에서 가장 아름다운 것일 거라고 생각한다. 그리고 아마 그 이유는 판테온에서는 다른 신전이나 성당에서 느껴지는 위압감이 느껴지지 않기 때문일 것이다. 경건하되 어딘가 유쾌한 분위기가

판테온에는 있다. 아마 이 건물을 고안한 하드리아누스가 딜레탕트였기 때문이기도 할 것이다.

저자가 이야기하는 좋은 설계라는 개념에는 많이 공감할 수 있었다. 분명 그것이 예술품이든 건축물이든 과학이론이든 아니면 프로그램이든, 보고서 '아, 아름답다'라고 느끼게 만드는 어떤 것이 존재한다. 그것은 아마 저자가 이야기하는 '좋은 설계는 간단하다'와 '좋은 설계는 쉬워 보인다'의 중간 어느 지점에 위치하는 감정일 것이다. 정말로 아름다운 것은 어떤 것이 그 형태 이외의 다른 형태로는 존재하는 것이 불가능할 것이라는 느낌을 주면서, 마치 그 형태로 항상 존재해왔고 존재할 수 밖에 없는 것처럼 보인다. 간단한 코드가 모두 잘 돌아가는 코드는 아닐지도 모르지만 잘 짜진 코드는 분명 간단한 코드일 것이다.

하지만 사실 이 글에서 무엇보다 공감할 수 있었던 부분은 처음에 저자가 객관적/주관적 취향에 대해 논하는 부분이었다. 첫 부분에서 저자는 취향이 개인적인 선호일 뿐이라는 것은 진실이 아니라고 이야기하는데, 분명 이것은 모든 것은 취향의 문제일 뿐이라고 외치는 시대의 조류에 반하는 것이기는 하지만, 나 역시 취향은 개인적인 선호의 문제가 아니라는데 전적으로 찬성하고 싶다. 좋은 취향이라는 것은 분명히 존재한다. 예를 들어 카라바조는 분명히 다비드에 비해 좋은 취향이고, 아무리 대중 예술을 옹호한다고 하더라도 모차르트는 모든 대중 음악가들을 합친 것보다 위대한 사람이다. 아무리 세상이 뒤집어지더라도 그 사실은 변하지 않는다. 문제는 개인적인 선호가 아니라 좋은 작품을 (저자의 표현을 빌리자면 '좋은 설계를') 알아볼 수 있게 만드는 훈련이라고 생각한다.

아마 저자가 이런 글을 쓴 이유도 여기에 있는 것이 아닌가 싶다. 엔지니어링이라는 것이 단순한 기술적인 측면만 있는 것이 아니라 창조적인 측면도 있는 것이라면, 그리고 좋은 설계를 알아보는 능력이 단지 타고나는 것이 아니라 훈련에 의해 길러지는 것이라면, 공학도들도 단순히 기술적으로 뛰어나면 된다고 생각할 것이 아니라 좋은 설계를 알아보는 안목 역시 길러야 하는 것이 아닐까? 그리고 어떻게 하면 그런 안목을 기를 수 있는 것일까, 그런 문제에 대해 생각해 봐야 할 것 같다.

6. 결론

이번 에세이 과제도 여러 가지 글을 읽으면서 많은 생각을 할 수 있었던 기회가 된 것 같다. 특히 Paul Graham의 글들은 생각할 점들을 던져주면서도 술술 읽혀 나간다는 점이 인상 깊었다. 영어로 읽어도 쉽게 알 수 있는 글 솜씨와 위트, 그리고 교양이 맞물려서 좋은 읽을거리들을 만들어낸 것 같다. 나도 저런 글을 쓸 수 있다면 좋을 텐데, 하는 점이 좀 부러웠다.

어느 새인가 프로그래밍 언어 과제도 벌써 6개 째라 감회가 새롭다. 특히 지난 번 에세이에 썼던 이야기를 돌이켜보면 얼마 지나지도 않았는데 프로그래밍 언어 수업을 들으면서 생각이 정말 많이 바뀌었구나 하고 새삼스럽게 놀라게 된다. 어느새 nML에도 그럭저럭 익숙하게 되었고 C보다야 nML 쪽이 훨씬 낫지 않나, 하고 생각하게 되어버렸으며 nML 정도의 타입 시스템은 당연히 있어야 되는 게 아닌가 하고 생각하게 된 것 같다. 어쩐지 한 학기 동안의 생각의 변화 치고는 무시무시할 정도로 스스로도 흠칫 놀라게 된다. 절대로 쉽게 들을 수 있는 과목이 아니었기 때문에 들으면서도 몇 번이나 왜 이걸 듣는다고 했을까 후회하곤 했는데 지금은 그저 정말 듣기를 잘했구나, 하고 생각하게 된다. 컴퓨터공학부에 왔다면 한번쯤은 꼭 들어볼 만한 강의인 것 같다.