2009 Fall 4190.310
Programming Language

컴퓨터공학부
2006-11782 송영찬

좋은 프로그래머와
좋은 프로그래밍 언어

1. 서론

좋은 프로그래머란 어떤 프로그래머를 말하는 것일까? 일반적으로 좋은 프로그래머란 '잘 돌아가는 프로그램을 빠른 시간 내에 만들 수 있는 능력을 가진 사람' 이라고 생각할 수 있다. 즉, 올바른 프로그램의 생산성이 높은 사람이라고 할 수 있다.

그렇다면 좋은 프로그래머가 되기 위해서는 어떻게 해야할까?

현재로서는, 좋은 프로그래머가 되기 위해서는 좋은 프로그래밍 센스를 가지는 수밖에 없다. 경영학, 물리학, 법학, 통계학 등 같은 분야에는 그 분야에서 필요한 계산이라든지 연산, 전략을 어떻게 세우면 좋은지에 대한 일반적인 지침들이 관련된 법칙이나 공식에 기초해서 많이 밝혀져 있다. 이에 비해 프로그래밍에 대한 일반적인 지침은 상대적으로 부족한 실정이다. 이러한 현실에서 좋은 프로그래머가 되기 위해서는 곧 좋은 프로그래밍 센스를 갖추어야만 한다는 결론이 나온다.

좋게 말해서 '좋은 프로그래밍 센스를 갖추면 좋은 프로그래머가 될 수 있다.' 이지, 나쁘게 말하면 '아무리 프로그래밍 연습을 해도 좋은 센스가 없다면, 좋은 프로그래머가 되기 힘들다.' 라는 뜻이다. 이렇게 되면 어떤 문제가 발생할까?

가장 큰 문제는, 소위 '천재'라고 불리는 사람들에 대한 의존도가 지나치게 높아진다는 것이다. 다르게 말하면 어떤 프로그램의 생산성을 프로그램을 담당하는 사람들의 능력을 보고 예상하기 힘들다는 것이다. 이런 문제는 프로그래밍에 관련한 업계, 특히 게임개발기업을 비롯한 IT기업의 경영에 영향을 미칠 수 있다.

그렇다면, 프로그램의 생산성이 개인의 능력에 지나치게 영향을 받지 않기 위해서는 어떻게 해야할까? 앞에서 말한 다른 분야의 '어떤 지침들'에 해당하는 것이 있으면 좋을 것이다. 이에 대해 필자는 '프로그래밍 언어의 선택'이 가장 중

요하다고 생각한다. 물론, 주위 선배들이나 교수님들에게 듣는 중요한 지침들도 있다. 예를 들면, “프로그래밍보다 설계에 비중을 둬라.” 와 같은 것이 있다. 하지만, 이러한 지침은 ‘일반적인’ 지침에 해당한다. 필자는 이런 일반적인 지침보다 상대적으로 소홀히 하는 것들 중에서 중요하다고 생각하는 것을 이 글을 통해 강조하고 싶고, 그 것이 바로 ‘프로그래밍 언어 선택의 중요성’ 이다.

그러면 ‘프로그래밍 언어 선택의 중요성’ 과 프로그래밍 언어를 선택할 때, 어떤 점을 주의해야하는지 살펴보도록 하자.

2. 본론 1: 프로그래밍 언어의 관성

관성이란 무엇인가? 본래 물리학에서 말하는 관성이란, 움직이던 물체는 움직이려 하고, 멈춰있는 물체는 계속 멈춰있으려 하는 성질이다. 즉, 원래의 상태를 유지하려는 성질이다.

이러한 관성은 사회 전반적인 현상뿐 아니라, 개개인의 일상생활에서, 그리고 프로그래밍 언어의 선택에서도 쉽게 발견할 수 있다. 일반적으로 사람들은 처음 배운 프로그래밍 언어를 계속 사용하려 한다. 가장 큰 이유는 역시 ‘익숙하기’ 때문일 것이다. 이러한 사실은 ‘Beating The Average'에서도 “Whatever language people happen to be used to, they tend to consider just good enough”라고 언급하고 있다. 또, 사람들은 일반적으로 처음 배우는 언어로 C나 C++, Java를 선택하는데 이는 최근에 대중적으로 사용되는 언어들이기 때문이다.

상황이 이렇다보니, 사람들은 어떤 프로그래밍이든지 C, C++, Java를 선택하여 작업을 하는 경우가 많다. 과연 이게 좋은 선택일까? 필자는 그 것은 좋은 선택이 아니라고 확고하게 말할 수 있다.

모든 프로그래밍 언어는 그 언어에 적합한 Problem Set을 가지고 있다. 빠른 연산을 필요로 하는 곳에 Script언어를 선택한다면, 하드웨어를 기술하는 언어로 Lisp을 선택한다면 누구라도 납득하기 힘들 것이다. 이런 점에서 필자는 'Beating The Average'에서 “When you choose technology, you have to ignore what other people are doing, and consider only what will work the best." 라는 대목에 크게 공감할 수 있었다. ‘Beating The Average' 의 저자는 다른 프로그래머들과 다르게 Lisp을 사용함으로서 성공한 사례를 제시하며 Lisp의 강점을 소개하고, 위와 같이 하는 일에 적합한 기술(여기에서는 프로그래밍 언어)을 선택할 것을 강조하고 있다.

‘Revenge Of The Nerds'에서도 프로그래밍 언어 선택의 중요성을 역설하고 있는 다음과 같은 대목을 발견할 수 있다.

In fact, choosing a more powerful language probably decreases the size of the team you need, because if you use a more powerful language you probably won't need as many hackers, and hackers who work in more advanced languages are likely to be smarter.

이렇게 여러 사람들이 언어 선택의 중요성을 역설하는데도 불구하고 사람들은 쉽게 사용하던 언어를 바꾸지 않는다. 앞에서도 이야기했지만 그 것은 ‘익숙함’을 버리지 못하기 때문이다. 하지만 이 ‘익숙함’이 단순히 사용의 익숙함이라면 큰 문제가 없다. 문제는 사용하던 언어에 맞춰진 사고방식이다.

여러 프로그래밍 언어를 다루어본 사람이라면 알겠지만, 각 프로그래밍 언어마다 그 언어로 구현하기 편한 방식이 있다. 그 방식에 따라 문제를 푸는 방법도 달라지며, 자연히 사고방식도 달라진다. 필자가 다루어본 언어 중에서 가장 사고방식이 다른 두 가지 언어를 꼽으라면 C와 ML이다. C의 경우 순차적인 명령형 언어로서 할 일을 단계적으로 명시하는 것이 주된 문제 해결의 방법인데 반하여, ML은 주로

Recursion을 이용하는, 즉, 큰 문제를 작은 문제들의 합으로 쪼개어 작은 문제를 품으로서 큰 문제를 푸는 방식을 취한다. 이렇게 사람들의 사고방식에까지 영향을 주는 '언어'를 "프로그래밍 언어는 단순한 기술이 아니라 사람들이 생각하는 방식이다. (programming languages are not just tech-nology, but what programmers think in) 그것은 반은 기술이지만 반은 종교다." 라고 'Beating the Average'의 저자는 표현하고 있다.

위에서 필자는 프로그래밍 언어 선택의 중요성을 언급하였고 또한, 사람들의 프로그래밍 언어에 대한 관성을 가지고 있는 이유를 살펴보았다. 그렇다면 이제 이런 질문을 해볼 수 있겠다.

그렇다면 프로그래밍 언어에 대한 관성을 가지고 있어도 괜찮은 언어를 만드는 것은 어떠할까? 즉, 필요에 따라 확장이 가능한 언어를 만드는 것은 어떨까?

이에 대해 'Growing a Language'의 저자는 그렇게 확장성이 있는 언어가 좋은 프로그래밍 언어이며, 우리에게 필요한 언어임을 아래와 같은 대목을 통해 말하고 있다.

> I need to design a language that can grow. I need to plan ways in which it might grow|but I need, too, to leave some choices so that other persons can make those choices at a later time.

실제로 최근에 많이 쓰이는 언어 중 하나인 C는 B언어로부터 시작하여, 사람들이 필요하다고 생각한 기능들이 추가되어 오늘날의 C에 이르렀다. 이것은 프로그래밍 언어에서 확장성이 얼마나 중요한 것인지 보여주는 단편적인 예라고 할 수 있다.

3. 본론 2: 좋은 Design과 프로그래밍 언어

이제 좋은 프로그래밍 언어가 갖추어야할 또 다른 조건에 대해 생각해 보자.

'Taste for Makers'에는 프로그래밍 언어 뿐만 아니라 어떤 분야에서도 적용할 수 있는 '좋은 디자인'에 대해 언급하고 있다. 그 중에서 필자가 특히 공감한 것은 아래 네 가지였다.

Good design is simple.
Good design is hard.
Good design looks easy.
Good design resembles nature.

위 네 가지를 요약하면, 좋은 디자인은 간단하지만 어렵고, 그러면서도 쉽게 보이며, 자연을 닮아있다는 것이다. 이제 이 네 가지를 프로그래밍 언어에 대해 생각해보자.

먼저, 좋은 프로그래밍 언어는 단순하다. 당연한 말이지만 프로그래밍 언어는 단순할수록 사용하기 편하다. 또 이것은 세 번째의 '좋은 프로그래밍 언어는 쉬워 보인다'와 연결할 수 있다. 단순하면 사용하기 편하기도 하지만, 그 단순한 것을 이용해서 만들어놓은 프로그램을 보면, 그 언어를 사용하기 쉬워보일 것이다.

또, 단순한 프로그래밍 언어라는 것은 짧은 Code로 프로그래밍이 가능한 언어라는 의미이기도 하다. 'Revenge Of The Nerds' 의 저자는 "Code size is important, because the time it takes to write a program depends mostly on its length." 라는 대목을 통해 Code size의 중요성을 언급하고 있다. 그리고 Powerful한 Language일수록 프로그램의 길이가 짧다("So the more powerful the language, the shorter the program")는 언급도 하면서 다시 한 번 프로그램의 길이가 그 프로그래밍 언어의 Level에 관계가 있음을 시사하고 있다.

다음으로, 좋은 프로그래밍 언어는 어렵다. 여기서 어렵다는 것은 그런 프로그래밍 언어를 만들기가 어렵다는 뜻이다. 프로그래밍 언어는 기본적으로 인간을 대신해서 기계가 어떤 일을 하도록 할 일을 지시하는 목적을 가진다. 따라서 인간이 기계에 명령할 수 있는 모든 것을 표현할 수 있어야하는데, 이것을 최대한 단순하게 만들어야 하는 것이다. 우리가 사용하는 한글이나 영어와 같은 언어도 이와 같은 특성을 갖고 있다. 적은 수의 기본글자로 우리가 표현하고자 하는 모든 단어, 문장을 만들 수 있다.

세 번째로, 좋은 디자인은 쉬워 보인다. 위에서 잠시 언급했지만, 쉬워보이는 것은 단순한 것과 연결할 수 있다. 간단한 예로, C의 입문서를 보면 1000페이지를 넘는 것이 대부분이다. 그런데 필자가 사용해본 언어 중에서 입문서가 가장 얇았던 ML은 300페이지를 넘지 않았다. 두 언어를 처음보는 사람이 입문서를 보고 언어에 대해 어떻게 생각할지 상상하면 답은 자명하다. C보다는 ML이 쉬워보일 것이다.

마지막으로, 좋은 디자인은 자연을 닮는다. 이 말을 필자는 '좋은 프로그래밍 언어는 인간의 사고를 자연스럽게 표현할 수 있다'로 해석하였다. 사람이 생각하는 것을 그대로 표현할 수 있는 프로그래밍 언어가 있다면 정말 프로그래밍이 쉬워질 것이다. 처음 컴퓨터가 만들어지고 프로그래밍을 위해 사용되었던 기계어나, 어셈블리어는 사람이 이해하기 힘든 Low Level Language에 속한다. 이에 비해 최근에 사용하는 C나 Java, ML, Lisp과 같이 사람이 이해하기 편한 언어는 High Level Language에 속한다. 'Beating the Averages' 의 저자는 저자가 사용한 언어인 Lisp을 사용함으로서 얻은 이득을 말할 때, "because Lisp was so high-level, we wouldn't need a big development team, so our costs would be lower." 라고 언급하며 High-Level 언어의 장점을 부각시키고 있다.

High-Level 언어의 장점을 단적으로 말하라면 무엇보다 디버깅에 소요되는 시간이 아닐까 한다. High-Level언어로 만

든 프로그램 코드는 Low-Level에 비해 사람의 생각에 가까운 형태를 가지기 때문에 코드를 이해하기 쉽기 때문에 High-Level 언어일수록 디버깅 시간이 짧다. 필자의 경험 속에서도 그 것을 알 수 있는데, 비교적 오래된 언어인 C보다 High-Level 언어인 ML이 디버깅하는데 시간이 적게 걸린다.

4. 본론 3: 프로그래밍 언어와 Type Check

마지막으로 최근 프로그래밍 언어 분야의 이슈가 되고 있는 Type Check에 대해서 생각해 보자. Type Check는 과거 Fortran에서부터 Lisp, C, ML, 그리고 Perl에 이르기까지 계속 발전해왔다. 'Perl해커들을 위한 프로그래밍 언어' 에서는 이러한 Type Check의 발전에 관한 내용을 다루고 있다.

Type Checking은 어떤 프로그램을 실행할 때, 오류를 정확히 자동으로 찾아주는 즉, Sound하고 Complete하게 오류를 찾아주는 프로그래밍 언어를 구현하기 위해 반드시 필요한 기술이다.

최초의 오류 검증은 단순 문법검증(Parsing)을 통해서만 이루어 졌다. 그러나 이 방법은 형식은 정상이나 논리적인 오류가 있는 프로그램에 대해서는 오류를 찾지 못하였다.

1990년대에 들어서서야 Type Checking을 통해 논리적 오류까지 검증할 수 있게 되었다. 이러한 Type Checking 기술에도 여러 단계가 있는데, Fortran의 경우 compile time에 Type Checking을 수행하는 Static Type Checking을 사용하였다. 그리고, Lisp의 경우에는 Performance는 좀 낮아질 수 있으나 Flexibility를 제공하는 Run-time Type Checking을 사용하였으며, C나 Pascal의 경우 pointer를 제공하는 Algol-based Type System을 사용하였다.

그러나 위에서 소개한 Type Checking 방법들은 모두 Type Checking이 틀릴 수 있다는 문제점을 지니고 있다. 그래서

대안으로 제시되는 것이 Strong Type Checking Language이다. Strong Type Checking은 입력으로 들어온 프로그램을 Scan하여 Type 방정식을 만들고, 그 방정식을 풂으로서 Type을 Check하는데, 대표적으로 ML이 String Type Checking Language에 속한다.

Strong Type Checking Language의 가장 큰 장점은 바로 거의 모든 프로그램의 실행에 있어서 Sound하고 Complete한 실행을 보장하며, 논리적 오류까지 잡아준다는 것이다. ML을 사용해본 사람은 알겠지만, 필자는 ML을 사용하면서 가장 좋았던 것이 바로 C의 Segmentation Fault와 같은 어디에서 발생했는지 알 수 없는 오류가 없다는 점이었다. 이러한 장점은 프로그래머로 하여금 보다 효율적인 프로그래밍을 가능케 한다.

5. 결론

위에서 필자는 프로그래밍을 할 때, 프로그래밍 언어 선택이 중요하다는 것을 말하고, 좋은 Design이 어떤 Design인가에 입각하여 어떤 프로그래밍 언어가 좋은 언어인가에 대한 견해를 밝힌 후, 마지막으로 최근 프로그래밍 언어의 주된 발전방향으로 이슈가 되고 있는 Type Checking에 대해서 언급하였다.

좋은 프로그래머가 되기 위해서는 상황에 맞는 프로그래밍 언어를 선택할 줄 알아야 한다. 그리고 좋은 프로그래밍 언어는 좋은 디자인을 가지며, 특히 Strong Type Checking Language라면 프로그래머의 입장에서 더 효율적인 프로그래밍이 가능하다. 필자의 경우, 이러한 String Type Checking Language로 ML을 사용해 보았는데 처음에는 익숙하지 않아서 C, Java가 훨씬 편하다고 생각했었다. 그러나 ML을 1년 정도 사용한 지금으로서는 ML이 C, Java보다 훨씬 강력하고

높은 프로그래밍 생산성을 가지고 있다고 생각한다.

ML을 비롯한 Strong Type Checking Language가 아직 완전하게 자리를 잡은 것은 아니다. 또한, Strong Type Checking Language가 완벽하게 완성된 것도 아니다. 따라서 사람들이 Strong Type Checking Language의 장점을 알게 되고, 관심을 갖고 연구를 하면 더 좋은 언어가 나올 수 있다.

필자도 아직은 프로그래밍 언어의 기초 정도를 맛본 어린 아이에 불과하지만, 최근 이슈가 되고 있는 Strong Type Checking이 기존의 Type Checking보다 프로그래머의 입장에서 얼마나 편한 프로그래밍 환경을 제공하는지 경험으로 알 수 있었다.

앞으로 프로그래밍 언어가 더 발전해서 프로그래밍 실력이 개인의 프로그래밍 센스에 비례하는 것이 아니라 주어진 문제에 적합한 언어를 선택하고 적합한 알고리즘을 선택 및 생각해 내는 능력에 비례하는, 그래서 누구나 노력하면 좋은 프로그래머가 될 수 있는 날이 오기를 기대해 본다.