프로그래밍 언어 에세이 #2

2008-11602

컴퓨터공학부

민현기

1. 들어가면서

프로그래밍 언어에 대한 이야기를 하기 전에 우선 실제 언어(natural language)에 대한 이야기를 해 보자. 언어란 사람 간의 의사소통을 하기 위한 도구로, 생각을 표현하는 매우 강력한 수단이다. 따라서 인간이 태어나면서 가장 먼저 배우는 것 중 하나가 바로 자기 생각을 표현하는 법, 즉 언어를 배우는 일이다.

그런데 이러한 언어, 즉 생각을 표현하는 수단이 유일하지 않다. 세상에는 무수히 많은 언어가 존재한다. 이 중 가장 많은 사람이 쓴다고는 할 수 없더라도 가장 '널리' 쓰이는 언어를 꼽으라면 당연히 영어일 것이다. 여러 나라가 영어를 공용어로 채택하고 있고 우리나라 역시 영어 때문에 수많은 사람들이 스트레스를 받고 있다. 게다가 대부분의 프로그래밍 언어 역시 문법구조가 영어로 이루어져있는 것을 감안할 때 새삼 영어의 대중성을 알 수 있다. 영어가 널리 쓰이는 이유는 무엇일까? 여러 가지 이유가 있겠지만, 영어가 '우수한 언어' 라서 사용되는 것은 아니다. 실제 핸드폰 문자나 컴퓨터 타자의 경우 한글이 영어보다 훨씬 빠르다는 실험 결과가 있으며 수많은 영어 인증시험 등을 볼 때 영어가 결코 배우기 쉬운 언어는 아니다. 즉, 영어가 널리 쓰이는 이유는 '언어 자체의 우수성'과는 전혀 무관한 다른 이유들이다. 중국어의 예를 생각해보자. 중국어의 경우 표의문자인 한자를 채택한 언어로서 배우기 매우 어렵고 다수의 한자를 외워야하는 불편한 언어이지만 사실상 세계에서 '제일 많이 쓰이는 언어'이다.

왜 이런 일이 발생할까? 더 좋은 언어가 있음에도 불구하고 여전히 중국인들은 한자를 사용하고 있고 수많은 사람들이 영어를 배우면서 골머리를 썩이고 있는. 바로 '익숙함'에 그 이유가 있다고 생각한다. 하나의 언어를 배우기란 엄청난 시간과 노력을 필요로 한다. 즉, 우리는 태어나서 처음 배운 언어, 즉 모국어를 가장 오래 사용했기 때문에 익숙해져버린 것이다. 즉 사용

하는 당사자는 다른 언어를 사용하는 것보다 자국어가 가장 편하고 쉽게 느껴지는 것이다. 즉, 수많은 영어 공인시험 역시 영어가 모국어가 아니기 때문에, 불편하고 힘들기 때문에 있는 것이다. 반대로 영어가 모국어인 사람이 한국어를 새로 배우기 위해서는 상당한 노력이 필요하고 아무리 노력하여도 모국어인 영어보다는 불편할 것이다. 이제, 이러한 '익숙함'이라는 관점에서 두 번째 에세이를 시작해보자.

2.프로그래밍 언어

프로그래밍 언어는 실제 언어와 매우 유사하다. 프로그래밍 언어 역시 프로그래머의 생각을 표현하는 수단으로 그 표현을 기계(튜링 머신)을 통해 확인하는 것이다. 프로그래밍 언어 역시 수없이 많이 있으며 많이 쓰이고 덜 쓰이는 언어가 있다. 대부분의 학교 및 산업 현장에서는 C, C++, Java 등의 언어를 사용하고 있다. 이 언어들 역시 널리 쓰이기 때문에 '좋은 언어'라고 말할 수 있는 것일까. 물론 자연어와는 다르게 프로그래밍 언어는 산업현장에서 사용되기 때문에 좀 더 효율적이고 좋은 언어를 찾으려는 움직임이 있다고 볼 수 있다. 그러나 그 '효율성'이라는 것이 과연 진짜 매우 좋은 언어이기 때문인가, 아니면 우리가 그 언어에 '익숙해졌기' 때문인가. 이것은 매우 중요한 문제이다. 왜냐하면 다른 언어를 사용하여 훨씬 더 좋은 결과를 낼 수 있는데, 단지 현재 많이 쓰이는 언어가 익숙하기 때문에 사용하는 것이라면 에너지와 자원의 낭비이기 때문이다.

확실히 위에서 언급된 C, C++, Java등의 언어들은 우리에게 '익숙한' 언어이다. 대부분의 학생들은 컴퓨터를 입문하면서 basic, C, java등의 언어를 접한다. 꼭 이러한 언어가 아니더라도 분명한 것은 대다수가 '기계적 언어'를 접하며 시작하는 것이다. C언어는 이미 30년이 넘은 언어이다 과연 30년도 넘은 언어가 현재의 모든 언어보다 효율적인가. 아니라고 생각한다. 우리는 C에, 절차적 언어에 너무 익숙해져 버린 것이다.

3.익숙함의 함정

프로그래밍 언어 과목에서, 모든 프로그래밍 언어는 튜링 complete 함을 배웠다. 이는 모든 언어는 튜링 머신에서 돌아가는 모든 프로그램을 작성할 수 있다는 점이다. 즉 자연어와 마찬가지로, 어떠한 언어를 쓰더라도 표현 방식이나 효율성 등은 다르지만 결국 원하는 생각을 표현할 수 있는, 즉 결과는 같다는 의미가 된다. 그렇기 때문에, 대부분의 사람들은 현재 많이 사용되는 언어, 즉 주류를 따라간다. 대중적이기 때문에 좋은 언어라는 믿음과 또한 사용하다 보면 어느새 그 언어에 익숙해져버리기 때문에 다른 언어를 쉽게 받아들일 수 없게 된다. 익숙함은 좋으면서도 한편으로는 두렵다. 어떠한 언어에 익숙해진다는 것은 그 언어를 매우 유연하고 강력하게 사용할 수 있다는 뜻이지만 한편으로는 새로운 것을 쉽게 받아들이기 힘든 관성이 생긴다는 것이다. 즉, 어떠한 언어에 익숙해진 순간, 다른 언어를 객관적으로 바라보기 힘들게 된다. 새로운 언어에 기존 언어에 없던 개념이나 다른 규칙들에 잘 적응을 하지 못한다면 그러한 것들이 모두 단점으로 보이게 될 것이다. 이것은 상당히 큰 부분으로, C언어 프로그래머들이 assembly에 비해 C언어의 강점을 주장하면서도 C보다 high level languge는 부정하는 점에서도 드러난다.

4.'좋은 언어'의 선택

따라서 어떠한 언어에 익숙해질 것인가, 어떤 언어를 사용해야 하는가의 문제는 매우 중요하다. 물론 자연어와 다르게 새로운 언어를 배우는 것이 그렇게까지 어렵지는 않지만 어떤 언어를 주로 사용해야 하는가 라는 문제는 매우 중요하다.

-Beating and Average- 에서는 Lisp을 이용하여 성공적인 software를 개발한 사례를 들고 있다. 이 글에서 주장하는 점 중 하나는 '언어에 관한 논쟁은 다분히 종교적이다' 라는 점이다. 위에서 언급했듯이 모든 언어는 turing complete하다. 또한 현재 사용되고 있는 언어의 대다수는 그에 맞는 장점이 있기 때문에 사용되는 것이다. 모든 부분에서 우수한 언어는 없다. 즉, 이러한 점에 기인하여 각 언어의 '추종자' 들은 자신의 언어의 장점을 부각시키고 다른 언어의 단점을 꼬집는다. 서로 이러한 주장을 펼치니 결국 의미없는 소모적 논쟁이 됨을 비판하는 것이다. 또한 이 글에는 Lisp과 같은 High level language에 대한 관점이 들어 있는데 바로 assembly와 C의 관계라는 점이다. 옛날 하드웨어의 기술이 떨어지던 때에는 C언어로 작업을 하다가도 메모리 부족현상으로, assembly를 이용하여 최적화를 시켰다고 한다. 하드웨어가 비약적으로 발전한 현재에는 이러한 작업을 많이 하지 않는다. 대신 이와 마찬가지로, Lisp과 같은 high level언어와 C와도 마찬가지로 서로의 단점을 보완할 수 있는 존재라는 것이다.

-Growing a Language- 와 -Tastes of Makers- 에서는 좋은 언어, 좋은 디자인에 대한 논의가 이루어진다. 과연 '좋은' 언어/디자인이라는 것이 있을까. 개인의 취향이 아닐까. 답은 No이다. 분명히 '좋은 언어', '좋은 디자인'은 존재하며, 이는 몇 가지 규칙에 의해 결정된다는 것이 이 글들의 핵심이다. 여기에서는 몇 가지 원칙을 들고 있는데 공통된 내용 중 하나가 바로 simplicity에 관한 것이었다. 즉, 단순한 것이 좋은 것이라는 것이고 작고 발전 가능성이 있는 언어가 좋은 언어라는 것이다. 이는 하드웨어 설계에 있어 단순함을 추구하는 RISC 아키텍쳐와 철학과도 일치하는 부분이다. simplicity에 관한 또 하나의 예를 들자면, 미국의 물리학자 파인만에 대한 일화가 있다. 그는 항상 '어떠한 물리 이론이라도 대학교 1학년 수준으로 설명할 수 없다면 이는 우리가 그 이론을 정확히 이해하지 못한 것'이라 주장을 하였다고 한다. 즉, 이는 단순화의 중요성을 강조한 것으로 어떤 이론이라도 대학

교 1학년 수준으로 단순화를 할 수 있어야 한다는 의미이다.

-Revenge of the Nerds- 에서도 이와 비슷한 내용이 나온다. 현재 주류 언어인 Java, C++ 등의 문제점을 알고 있으나 이를 수정하지 못하는 이유는 언어의 덩치가 너무 크고 복잡하기 때문이라는 것이다.

- 게임 프로그래밍 정글의 원시인들 - 에서도 유사한 내용이 나온다. Lisp의 경우 vector와 그에 따른 연산, 행렬과 그에 따른 연산을 표현하기 위해 벡터, 행렬같은 데이터 구조를 집어넣을 필요가 없다. 단순히 pair만 있으면 벡터, 행렬등의 데이터 구조를 표현할 수 있다는 점을 강조하고 있다. 또한 이 글에서는 흥미로운 실험이 등장하는데 바로 ocaml과 C언어의 비교이다. 동일한 프로그램을 C와 ocaml로 작성한 경우 C의 경우는 1000line에 125frame/sec, ocaml의 경우 700line에 100frame/sec 였다. 즉, code의 단순함과 perfomance라는 상호 보완적인 특성을 가지고 있다는 것을 의미한다.

이러한 모든 내용은 공통적으로 말하고 있다. '간단한 언어'가 좋다. 간단한 언어를 써라! 이러한 단순성에서 나온 것 중 하나가 바로 Type Checking System 이라고 생각한다. C언어가 대중화된 이유가 무엇이었을까? C를 처음 배울 때 본 책을 찾아보니 C의 장점으로 단순한 문법구조가 있었다. 이전의 언어보다 더 간단한 문법으로 같은 프로그램을 표현할 수 있기 때문에 강력하다는 것이다. 즉, C언어 역시 단순성에서 그 장점이 부각되는 것인데, 이러한 C의 단순성에 위배되는 것이 무엇인가? 바로 타입 시스템에 있다고 생각한다. C는 우리가 배운 K와 흡사한 언어이다. 모든 값들은 타입을 가지고 있기는 하나, 이를 컴파일러가 완벽하게 체크해줄 수가 없다. 따라서 프로그래머에게 떠맡겨버렸다. 프로그래머가 타입을 '책임'져야 하는 것이다. 프로그래머가 이용할 변수들의 타입을 명시하고 사용하여야 한다. 수많은 C 프로그래머들이 디버깅에 밤을 새는 큰 이유중 하나가 바로 이 타입 시스템이라고

생각한다. 프로그래머에게 떠맡기는, 즉 컴파일러가 프로그래머에게 알아서 올바른 타입을 넣기를 기대하는 타입 시스템의 'side effect'는 바로 프로그래머가 명시한 타입과 실제 값의 타입이 일치하지 않아도 된다는 것이다. 이는 장점인 동시에 단점이 될 수 있는 부분으로 일치하지 않는 타입을 이용하는 것을 C언어 프로그래머들은 흔히 '기교'라고 한다. 그러나 이는 매우 위험한 프로그래밍으로 에러가 났을 때 쉽게 파악하기 어렵게 하는 원인이 될 수도 있다.

mL은 이러한 점에서 탈피하여 '다형 타입 체킹 시스템'을 지원한다. 즉 프로그램이 실행되기 전에, 프로그램 텍스트만 보고 타입을 유추하는 시스템이다. 이러한 시스템은 안전(sound) 한가? Yes. 어째서? 바로, 형식 논리 체계를 이용하여 수학적으로 완벽히 증명했기 때문이다. 수학적 이론이라는 좋은 토대위에 쌓은 성은 안전하다. 즉, 타입 체킹 시스템이 실행되지 않는 프로그램을 실행된다고 하는 경우는 절대로 없다. 수학적 기반, 즉 이론이 튼튼한 언어는 단순해진다. 안전하기 때문이다. 즉, 타입 체크를 예로 들면, 어떠한 타입 명시를 하지 않아도 규칙에 따라 완벽히 체크가 되기 때문에 우리는 타입을 표현하기 위해 많은 노력을 할 필요가 없어 코드가 짧아질 뿐 아니라, 타입에 대한 생각을 배제한 채 오직 프로그램 작성의 문제에만 몰두할 수 있기 때문에 단순하면서도 좋은 언어가 되는 것이다. 퍼포먼스에 목숨을 걸지 않아도 되는 경우, 수학적 기반이 튼튼한 언어는 좋은 언어라고 할 수 있을 것이다.

5.마치며

세상에는 수없이 많은 언어들이 있다. C와 같은 절차적 언어부터 시작하여 Java, Small talk와 같은 객체지향 언어, Perl, Python과 같은 스크립트 언어, Lisp, Ocaml과 같은 함수형 언어 등등. 흔히들 하나의 언어를 잘하게 되면

다른 언어는 쉽게 배울 수 있다고 하는데, 이는 같은 계열의 언어에서 해당하는 이야기이고, 전혀 다른 유형의 언어를 접할 때는 오히려 기존 언어에 익숙해져서 적응하기 어려운 경우도 있다.

사실 위에서 언급했듯이 '좋은 언어' 라는 것은 시대에 따라 변하는 개념이라 생각한다. 여기서의 '시대'가 변하는 이유 중 하나가 바로 하드웨어의 발전이라고 생각하는데, 프로그램을 수행하는 기계가 매우 조악했던 시대에는 기계에 가까운 언어일수록 좋은 언어였다. high level language의 경우, 그 추상성 때문에 느린 기계에서 수행하기는 너무 버거웠기 때문이다. 다른 회사와 경쟁해야 하는데 수학적으로 완전한 언어를 고집하다가는 경쟁에서 도태되기 때문이었다. 그러나 하드웨어가 발전함에 따라 '좋은 언어'의 정의는 점점 기계와는 연관성이 멀어져 갔다. 아직은 기계의 성능에 얽매이지 않는, 즉 퍼포먼스를 완전히 배제한 채 순수한 언어적 관점에서의 논의가 이루어질 수 있을 때는 아니라고 생각한다. 그렇기에 분야에 따라 적합한 언어가 나누어지고 계속 수많은 언어가 생겨나는 것이 아닐까.

그러나 명심해야 할 것은 기계 환경은 항상 발전하고 있다는 것이다. 이는 현재의 언어가 끝이 아닌, '좋은 언어'의 정의 역시 계속 바뀌게 될 것을 의미한다. 어떤 언어에 '익숙해진다'라는 것은 그 언어의 고수가 된다는 것을 의미하겠지만 한편으로 변화를 받아들이지 못한다면 도태될 수도 있다는 것을 의미한다. 델파이, 코볼, 포트란 등 예전에도 수많은 언어가 있었지만 시대가 변하면서 주류 언어는 계속 변해왔다. 이는 앞으로도 마찬가지로 변해가는 시대에서 열린 마음을 가지고 새로운 언어들을 받아들여야 할 것이다.