C와 ML,
두 언어를 통해 보는 프로그래밍 언어의 변화
그리고 앞으로의 모습

2010년 2학기

Programming Language

Essay #1

서울대학교 컴퓨터공학부

2007-11797

이치민

목 차

1. 들어가며

프로그래밍 언어란 어떤 것이며, 왜 끊임없이 연구 되며 변하고 있을까.

오래 전부터, 그리고 어쩌면 현재까지도, 프로그래밍 언어는 컴퓨터에게 명령을 내리기 위해, 그리고 컴퓨터에서 쓰이는 프로그램을 만들기 위해 사용되는 한 작은 도구 같은 존재로 있어왔다. 그러나 시대가 변하고 있다. 점점 더 복잡해지는 세상 속에서 컴퓨터가 차지하는 비중은 점점 더 커져만 갔고, 더 많은 것들이, 더 복잡한 것들이, 더 중요한 것들이 컴퓨터 프로그램으로 작동되려고 한다. 그러한 상황 속에서 이제 프로그래밍 언어는 단순한 도구 이상의 의미를 갖게 되었다. 더 나은 세상을 위해 필수가 되어버린 프로그래밍 언어, 그렇기에 더 좋은, 더 나은 언어를 위해 프로그래밍 언어는 계속 연구되고 발전하고 있는 것이다.

프로그래밍 언어가 어떤 특징을 가지며 어떻게 변해왔는지, 그 역사와 현재의 모습을 두 가지 언어를 통해서 알아보고, 앞으로 프로그래밍 언어가 나아가야 할 미래의 모습까지 고민해보도록 하자.

2. C언어

2.1 C언어의 역사

C언어의 조상은 BCPL이란 언어이다. 1960년대 중반, BCPL이라는 언어가 만들어졌다. Bell 연구소에서 Multics라는 기계를 돌리기 위해 만든 언어이다. 이러한 BCPL을 발전시켜 Ken Thompson이라는 사람이 B언어를 만들었고 그것의 불편함을 개선해 NB라는 언어가 만들어졌다. 그리고 NB 언어에 타입 시스템을 비롯한 다양한 변화를 주어 C언어가 처음 탄생하게 된 것이다. 이후 C언어의 발달된 portability 때문에 C언어는 UNIX의 기반 언어가 되었으며 더욱더 널리 사용되게 되었다. 그러나 급속도로 퍼져나가다 보니 다른 많은 문제들이 생겼다. 컴파일러의 난개발이 그 대표적인 문제인데, 그런 문제들을 해결하기 위해 C언어의 표준화 작업이 이루어졌다. 그리고 그런 과정을 거쳐 만들어진 것이 현재의 C언어이다.

2.2 C언어로 느끼는 기계 중심의 언어.

C언어는 만든 언어가 아니라 만들어진 언어이다. 좋은 언어를 위해 만든 언어가 아니라, 단지 필요에 의해 만들어진 언어라는 뜻이다. C언어는 BCPL에서 B로, B에서 NB로, 그리고 C로 오면서 필요를 위해, 편의를 위해 많은 것을 떼고 붙인 결과물이다. 물론 그러한 과정 끝에 만들어진 C언어는 편리하긴 하다. 사용자를 조금 더 편하게 하기 위해 많은 편리한 기능들이 만들어졌으며 그 대표적인 기능인 포인터는 사용자의 필요에 의해 메모리를 직접 건드릴 수도 있게 만들었다. 그러나 그러한 필요, 편의를 위해 마구잡이로 추가된 것들은 오히려 많은 문제를 발생시키기도 했다. 실수로 잘못 참조한 메모리 때문에 수도 없이 segmentation fault가 나며 사소하게 잘못 생각한 배열의 index는 아예 엉뚱한 결과를 만들어내기도 했다. 게다가 그러한 실수는 찾아 고치기도 매우 어려운 것들이었다. 물론 능숙한 프로그래머라면 그런 실수는 거의 하지 않을 지도 모르며, C는 아직까지도 발전 중인 언어이다. 그러나 C는 태생적으로 그런 일들이 생길 수밖에 없는 언어이다. 그리고 그 이유는 다음에 설명할 기계 중심의 언어라는 특징에서 비롯된다.

C언어는 기계와 가까운 기계 중심의 언어이다. C언어가 그렇게 된 이유는 다름 아닌 C언어의 조상인 BCPL에 있는데, BCPL 자체가 특정 기계를 돌리기 위해 개발된 언어이며, 당연히 목적 역시 해당 기계가 잘 작동하도록 기계에 명령을 내리는데 있었기 때문이다. 그리고 그러한 BCPL를 변화, 발전시켜 만든 C언어가 본 모습인 BCPL의 특징에서 크게 벗어날 수 없다는 것은 당연한 결과일 수밖에 없었다. 사실 이는 비단 C언어만의 특징이 아니다. 그 시기에 만들어진 많은 언어들, 그리고 최근까지도 만들어지는 많은 언어들이 이러한 특징을 가지고 있으며, 이러한 언어들은 기계에 명령을 내리는 방식 이라는 틀 속에서 조금씩 다르게, 조금 더 편리하게 만들어진 언어들이다.

이들, 기계 중심의 언어들을 이용한 프로그래밍은 논리의 적용이라기보다는 명령들이 나열이다. 컴퓨터에게 명령을 하나하나 내리고 그 명령들의 결과로 답을 만들어 내는 것이다. 어찌 보면 너무나 당연하고 자연스러운 프로그래밍인 것 같다. 실제로 우리는 오랫동안 그렇게 해왔으니까. 그러나 이러한 방식의 문제점은 사람이 생각하는 논리적인 방식과는 큰 차이가 있다는 데 있다. 그리고 그러한 차이는 위에서 말했던 수많은 오류들을 필연적으로 발생시킬 수밖에 없는 구조를 만들고 있다. 기계가 아닌 사람의 논리에 가깝게 만들어진 언어로 대표되는 ML에 대해 생각해보고, 그 차이를 느껴보도록 하자.

3. ML

3.1 ML이란

시간이 지나고 프로그래밍 언어에 대한 연구들이 활발해지면서, C로 대표되는 기존의 언어들의 문제점을 해결하기 위해 패러다임을 아예 바꾼 2세대 언어들이 등장했다. 그 중 하나가 ML이다. ML는 1960년대 Robin Milner등이 만들어낸 언어이다. ML의 가장 큰 특징은 수학적, 논리적 모델을 사용해 언어를 만들었다는 데 있다.

3.2 ML로 느끼는 논리 중심의 언어.

위에서 말했듯 ML을 만들 때는 수학적, 논리적 모델을 사용하였다. proof method나 lambda calculous 등이 그것이다. 그로 인해 생기는 매우 큰 장점을 생각해 보자.

필자처럼 C 프로그래밍에 대한 경험이 그리 많지 않은 사람이든 C에 매우 능숙한 사람이든 상관없이 C 프로그래밍 경험이 조금이라도 있는 사람이라면 ML을 처음 배웠을 때 모두 같은 느낌을 받았을 것이다. syntactic 오류만을 잡아주었을 때 semantic 오류가 발생하지 않고 한 번에 돌아갈 때의 그 감동 말이다. 물론 C에 익숙해 있던 내가 처음에 ML로 프로그래밍 한다는 것은 매우 어색한 일이었다. 어떤 명령을 내리고 싶어도, 혹은 대충 변수를 만들어 이런 저런 잡다한 것들을 기억하게 만들고 싶어도 뜻대로 할 수 없었기 때문이다. 그러나 그것은 나를 생각을 하게 만들었고, 그 결과로 나온 프로그램은 참 뭐랄까, 확실히 아름다웠다. 그렇게 semantic 오류가 잘 나지 않는 이유, ML로 만든 프로그램이 아름다워 보이는 이유는 그것이 기계를 위해 만들어진 언어가 아니라 사람을 위해, 사람의 생각과 논리를 위해 만든 언어라는 것 때문이다. 이것이 바로 논리적인 언어 ML의 가장 큰 특징이다. C언어를 가지고 프로그램을 만들 때는 '이 일을 하고 저 일을 하고 이렇게 저렇게 고치면 그런 결과가 나오겠지' 라는 생각을 가지고 프로그래밍을 한다면, ML로 프로그램을 만들 때는 그러한 일을 하는 프로그램의 논리와 구조를 생각하고 프로그래밍 하게 된다. 그러기에 이렇게 만들어진 프로그램은 인간이 보기에 아름다울 수밖에 없는 것 같다. 또한 그것 외에도 ML은 type checking 등을 통해 최대한 프로그래머의 논리적인 실수를 방지해 준다. 이는 모두 단순한 명령의 나열을 위한 언어가 아니라, 논리를 두고 만들어진 언어라는 것, 사람의 생각과 논리를 그대로 옮기기 용이하게 하고 기계를 위한 것이 아닌 사람을 위해 만들어진 언어라는 데에서 나올 수 있는 것들이다. C와 ML의 이러한 본질적인 차이, 그 차이를 통해 프로그래밍 언어의

변화의 모습 대해 생각해보자.

4. C와 ML을 통해 비교해보는 프로그래밍 언어의 변화

C를 통해 본 기계 중심의 언어의 특징과 ML을 통해 본 2세대 언어, 즉 논리 중심의 언어의 특징은 우리에게 프로그래밍 언어는 변하고 있다는 것을 보여준다. 그리고 그 변화의 방향은 바로 올바르고, 아름다운 프로그램을 추구하는 데 있다. 기계에게 명령을 내리기 위해, 단순한 명령의 나열을 이용해 프로그래밍을 하던 것에서 이제는 논리를 적용시키고 논리로 프로그래밍 하는 언어로 변화했다. 그 변화는 분명 기계 친화적인 언어에서 발생할 수 있는 문제점을 극복하고 사람이 더 편리하도록, 올바르고 더 아름다운 프로그램을 만들 수 있도록 하기 위함이었고 실제로 그 변화는 우리에게 더 아름다운 프로그램을 제공해 주고 있다. 이는 프로그램이 세상을 만들어가는 상황 속에서 어찌 보면 당연하며, 필수적인 변화 과정이 아니었을까.

그러나 중요한 것은 아직도 변화'중'이라는 것이다. 위에서 논리적이며, 인간에 가까운 언어라고 극찬해 마지않았던 ML역시 완벽한 언어는 아니다. ML로 세상 모든 프로그램을 만들어 낼 수 있는 것도 아니며, ML이 가장 편리한 언어라고 말할 수도 없다. 또한 ML로 만들어진 프로그램이 아무리 논리적이라고 한들 그 프로그램의 완벽성을 증명해 낼 수도 없다. 그렇다면 프로그래밍 언어는 앞으로 어떤 방향으로 나아가게 될까.

5. 프로그래밍 언어의 미래

프로그래밍 언어를 연구하는 학문에서 독특하게 대두되는 이슈 중 하나가 있다. 이광근 교수님께서 번역하신 "계산이란 무엇인가에 대한 아이디어들"에 나와 있는 '같은 프로그램이란 무엇인가 (semantic equivalence)'이다. 이것이 무슨 의미가 있고 왜 중요할까. 누차 얘기했듯이 이제 컴퓨터 프로그램은 세계를 장악하고 있다. 그러한 상황에서 한 프로그램의 정확성이란, 굳이 입으로 설명할 필요조차 없이 매우 중요할 것이다. 같은 프

로그램이란 무엇인가에 대한 이슈는 결국 어떤 프로그램의 정확성을 따지는 것과 그 맥락을 같이 한다고 생각한다. 어떤 두 프로그램의 비교는, 한 쪽이 정확한지, 어느 쪽 이 옳고 그른지를 우리에게 얘기해 줄 수 있는 힘을 가지고 있다. 그것의 중요성은 말할 필요도 없고, 그러한 과정은 물론 수학적, 논리적인 증명에 의해 해결되어나가야 할 문제들이다. 그러나 역사적으로 사용된 언어들에게서 그런 것 들을 증명해 내기란 불가능에 가까운 일이라는 것을 알 것이다. 여기에 바로 우리 프로그래밍 언어가 앞으로 나아가야 할 방향이 담겨 있다.

수많은 프로그램이 세상을 만들어 나가는 상황에서, 프로그래밍은 더욱더 확실한 이론적, 수학적, 논리적인 모델을 가지고 만들어져 나가야 할 것이다. 기계와의 가까움을 넘어, 사람에게 편리한 언어, 그리고 그것 또한 넘어 논리적이며 완벽하다고 증명될 수 있는 언어, 그러한 아름다운 언어를 만들어나가는 것이 앞으로 프로그래밍 언어가 발전해 나가야 할 방향이며 이 시대에 프로그래밍 언어를 공부하고 있는 우리에게 주어진 숙제일 것이다.

6. 마치며

ML을 처음 사용할 때의 그 느낌은 아직도 잊혀지지 않으며, 난 아직도 후배들을 만나면 ML을 추천한다. 복잡하게 설명할 수는 없지만 재미있고 신기한 언어라고.

단순히 명령문을 끄적거리며 언어를 이용하던 나에게, 프로그래밍 언어 수업과 이번 에세이 숙제는 큰 생각의 변화를 가져다주었다. 단순히 기존의 방식에 만족하지 말고 다음 세대의 언어를 위해, 더 나은 언어를 위해 조금 더 생각하고, 조금 더 고민하며 힘찬 발걸음을 내디뎌야 할 때가 온 것 같다.