Programming Language
맛있는 반찬이 차려진 맛있는 밥상

김예성

November 24, 2010

Contents

1 언어 세상의 밥상

지난 번 과제에서는 '언어의 친절함'에 대해서 이야기 했다면, 이번 과제에서는 '언어 밥상'에 대하여 이야기 하려 한다. 우리네 프로그래밍 '언어 세상'–'언어 밥상'에는 서로 다른 많은 '언어'–'반찬'들이 있다. 언어의 표현력은 비슷하다고 할 지라도, 그 것이 가진 특징들–'독특한 맛'들은 저마다 다르다. 건강한 프로그래밍 세상을 살아가기 위해서는, 각각 어떤 언어들이 있고 각 언어의 맛들을 올바르게 이해할 필요가 있다.

그러나, 프로그래밍 언어를 대하는 우리네 자세는 일반적으로 편식이 심하다. 주류 언어라고 불리는 C, C++, Java에 대해서 소개하는 책은 넘쳐나고 실제 개발 환경도 거기서 크게 벗어나지 않는다. ASP.net이나 Objective C와 같은 다른 플랫폼을 위한 언어를 접할지라도, 모두 비슷한 갈래에서 나온 언어들을 다룰 뿐이다. 이런 환경에서는, 우리의 언어가 정말 '건강한지'를 판단할 기회를 갖는 것 조차 쉽지 않다.

이번 에세이에서는, 우리네 언어 반찬과 밥상에 대해서 이야기해보려 한다. 특별히 LISP이나 ML과 같은 갈래의 언어를 옹호/찬양하는 사람들이 말하는 '맛'에 대해서 읽기 자료를 바탕으로 살펴보고, 각 언어–반찬들이 맛있는지 맛없는지, 또 어떠한 맛을 가졌는지에 대하여 이야기해보겠다. 나아가, 이 과목을 수강하면서 품게된 '건강한 밥상'에 대한 생각들도 조금 이야기하겠다.

(여담 한 가지. 이 글의 제목인 '맛있는 반찬이 차려진 맛있는 밥상'은 읽을거리를 미리 읽어두다가 떠올라서 가지고 있던 주제인데, 6번 과제 말미에 쓰여진 '건강해진 밥상', 'Bon app´etit! ...' 이란 글귀를 보고 살짝 놀랐다.)

2 맛있는 반찬 이야기: 읽을 거리로부터

먼저, 받은 읽을 거리로부터 '맛있는 반찬'이 무엇인지에 대하여 살펴보자.

2.1 언어는 서로 같지 않다

반찬의 맛은 모두 다르다

"우리들의 보복: Revenge of Nerd" 로부터

많은 위압적Pointy-haired인 상사들은 언어는 다르지 않다는 생각에 어떤 언어를 사용할지를 강요한다. 관성에 '취해'있거나, 대중적인 것이 표준적인 '거시다'라는 생각에 빠져서 언어가 가진 차이점에 대해서는 이해하려 들지않고, 그저 강요할 뿐이다.

하지만, 우리가 익히 알고 있듯이 언어는 서로 다르다. 그저 다르다는 말로 표현할 수 있을 정도가 아닌, 다른 근원에서 출발한 서로 다른 갈래들의 언어들이 있다. 목적에 따라 좋은 언어와 나쁜 언어가 있는데, 상사들은 그 차이들을 생각해 보려 하지 않는다. 게다가 지난 글에서 이야기 했듯이 언어는 부족함을 보완하면서 발전하고 있는데, 새로운 언어로 갈아타는데에 대하여 생길 수 있는 비용에 대하여 지나치게 걱정하기도 한다. 사실 이 문제는 새로운 무기를 장착한 많은 좋은 언어들-이를테면 C#이라던가-이 만들어진 2010년을 살고 있는 현재에도, 많은 개발 현장에서 되풀이 되고 있다.

글쓴이는 이런 강요들에 대하여 보복할 수 있는 무기로 LISP을 소개하면서, 그 '자유로운 강력함'에 대하여 역설한다. 여기에 더하여 LISP이 만들어질 때 고려된 아이디어와, 실질적인 예도 함께 소개한다. LISP은 Function Type, Recursion, tree 구조 등과 같이 프로그래머가 자유롭게 생각할 수 있게 제시해 주는 틀들과, Garbage Collection과 프로그래머의 양손을 자유롭게 해주는 기반 기술들까지, 프로그래머를 '프로그래밍'에 집중하게 해주는 많은 요소들을 갖추고 있다.

2.2 적절한 언어의 선택이 성공적인 프로젝트를 만든다

맛있는 반찬을 고르기

"치고 나가기: Beating the average" 로부터

LISP을 이용한 사례로, Web Application인 Viaweb을 개발했을 때에 대하여 소개하고 있다. 글쓴이는 Viaweb이 성공할 수 있었던 가장 중요한 이유로 LISP을 사용하였다는 것을 꼽는데 주저하지 않는다. 그 당시의 많은 프로그래머들이 가장 익숙하게 사용하였던 평균average 언어 C를 사용하는 대신, LISP을 사용하므로써 개발 비용과 시간을 줄일 수 있었고, 결국 Yahoo Store에 사용될 수 있었다.

LISP은 개발의 집중도를 효과적으로 높여 성공적인 프로젝트를 이끌 수 있었다. 글쓴이는 강력한 언어를 사용하면 잠재적 우위를 가질 수 있다는 것과 그 예로 Viaweb을 개발할 때 사용한 LISP의 장점에 대하여 이야기 한다. 일반적으로 사용하는 C언어 같은 언어를 이용하다 보면, 프로젝트를 만들기 위해서 신경 써주어야 하는 기계적인-프로젝트 외적인 문제가 많을

수 밖에 없다. 그러나 글쓴이는 '비밀 병기' LISP을 사용함으로써 프로젝트에서 해결해야 하는 내적인 문제들에만 집중하여 빠르게 개발 할 수 있었다.

더불어, 글쓴이는 평균적인 해답을 사용하는 것이 과연 최적의 해답인지에 대해 고민이 필요하다고 말한다. 많은 사람들은 자신의 손에 익숙한 도구-언어를 사용하여 개발하는 것이 그리 나쁘지 않은 길이라고 여기는 것을 종종 볼 수 있다. 혹은 그 길이 최고의 길이라고 생각하기도 한다. 그러나, 이 글의 예에서도 볼 수 있듯이, LISP을 이용하여 꼭 필요한 부분들에 대하여만 고민하여 개발함으로써, 평균적인 언어를 사용하는 경쟁사들에 대하여 우위를 점할 수 있었다. 적절한 언어를 선택한 것이, 그냥 '평균 언어'를 사용하는 경쟁자들보다 앞서 나갈 수 있는 원동력이 된 것이다. 즉, 프로그래밍 언어는, 단순히 기술이 아니라 사람의 사고를 이끌어주는 '길잡이'로 작용하여, 성공적으로 프로젝트를 이끌 수 있었다.

2.3 좋은 디자인을 가진 언어란 무엇일까?

맛있는 반찬이란 무엇일까?

"만드는 사람의 안목: Tastes of Makers" 로부터

그렇다면, 언어에 '자유로운 강력함'을 가져다 주는 좋은 디자인은 무엇일지 생각해보자. 이 글에서는 좋은 디자인은 어떤 항목을 갖는지에 대하여 하나 하나 항목을 들어 소개하고 설명하고 있다. 간단함, 신속함을 가질 것, 정확한 문제를 풀 수 있을 것, 암시를 가지고, 재미있을 것, 심사 숙고해서 만들 것, 쉬울 것, 대칭적일 것, 자연스러울 것, 재사용 가능할 것, 독특하고 과감할 것 등이 이 항목들이다. 이에 대하여 단순히 프로그래밍 언어에 국한하지 않고 예술-문화에 걸친 다방면에 걸쳐 설명하고 있다.

언어도 이러한 디자인 특성을 가지고 있어야 한다. 많은 고민들을 가지고 만들어진 쉽고 자연스러운 '좋은 디자인의 언어'일수록, 프로그래머는 더 자유롭게 프로그램을 만들 수 있을 것이다.

또한, 프로그래밍도 이러한 안목에서 접근할 수 있어야 한다. 때때로 프로그래밍은 예술이라는 생각을 한다. 예전에 프로그래밍에 대하여 친구들과 이야기를 나누면서, 사실 컴퓨터공학과는 미대에 있어야 하는 것 아니냐라는 우스개소리를 한 적이 있다. 실제로 많은 창조적인 부분들이 예술의 그 것을 닮았으며, 이는 궁극적으로 그 것을 기술하는 도구인 프로그래밍 언어에 의해 완성된다. '프로그래밍 언어'는 좋은 디자인을 가져야 한다는 생각에 전적으로 동의한다.

2.4 프로그래밍 언어 키우기

맛있는 반찬 만들기

"프로그래밍 언어 키우기: Growing a Language" 로부터

그렇다면 언어를 어떻게 '잘' 키울수 있을까? 이 글의 글쓴이는, 언어를 키우는 형식에 대해서는 '작게 작게' 하지만 그 성장은 '크게 크게' 하자고 말한다.

궁극적으로 이야기하려는 것은 '가볍지만', '강력한' 언어를 만들자는 것이다. 사뭇 이 두 가지 개념은 다른 종착지를 가르키는 것 같지만 서로 같은 종착지로 도착할 수 있다. 언어의 성장 자체에는 많은 사람들이 이야기를 내놓고 함께 자유롭게 만들어가면서(라이브러리를 만들어간다거나), 그 형식이나 의미는 누구나 쉽게 익힐 수 있을만큼 충분히 작은 언어를 만들고 성장시키는 것이다. 이를 위해서 최소한의 관리자만이 언어를 관리하고 모두가 참여하는 형태의 성장을 제안한다.

하지만, 지금의 많은 프로그래밍 언어들은 너무 배워야 할 '부잡스러운' 것들이 많다. '그건 되지만 이건 안되고'란 이야기를 프로그래밍을 처음 접할 때 얼마나 많이 들었던가? 몇 년 전 친구들에게 '그건 배열이라서 포인터이고, 포인터는 사실 배열은 아니야.'라는 답변을 해줄 때, 어디부터 설명을 해야 하나라는 마음 한 구석이 답답했던 기억이 난다. 오늘도 어떤 친구에게 C++을 설명하면서, 'static 멤버 함수는 선언에서는 static을 붙여주었지만, 정의에서 붙이면 안 되고'란 이야기를 해주었다. 그 뿐이 아니다. C에서는 '컴파일러의 편의를 위해' 변수 선언이 항상 맨 앞에 와야 한다.

이 '부잡스러운 것에는' 그 안에 거대한 기계 원리가 들어 있는 경우도 많고, 또는 특별한 이유가 아예 없는 경우도 많다. 그냥 외워야 한다. 이러한 예는 수도 없이 많다. 이 모든건 '잘' 자라지 못한(원칙과 고민없이 성장한) 언어의 특성에 기인하는 것은 아닐까.

2.5 올바른 타입 시스템이 건강한 프로그래밍 환경을 만들다

맛있는 반찬 만들기2

"Perl 해커들을 위한 프로그래밍 언어:Strong Type" 로부터

다음으로 잘 자라지 못한 언어와 잘 자란 언어가 갈리게 된 중요한 잣대인 타입 시스템의 변천사에 대하여 이야기 해보자. 이 글에서는 다양한 언어들이 타입 시스템을 어떠한 관점에서 받아들이고 구현하려 했으며, 그 결과 각각 어떻게 성장하였는지에 대하여 설명하고 있다.

결론부터 말하면, Fortran과 ALGOL, C, Pascal과 같은 언어의 타입 시스템은 실패했다. 애초에 타입 시스템을 염두에 두고 만들어진 문법이 아니었기 때문에, 컴파일러가 인식하기 위한 타입들이 이미 존재하고 일관성도 없었다. 이에 기반하여 타입 시스템을 적용하려고 하다보니, 기본 논리에 맞지 않는 타입 시스템 요소들이 산재할 수 밖에 없었다. 이에 대한 대응책으로 Pascal은 Strong Type을 엄격히 적용하는 쪽으로, C는 느슨히 적용하는 쪽으로 그 성장하였으나, 결국 타입 시스템이 너무 불편하거나 목적이 달성되지 못할 수 밖에 없었다.

그에 비해, LISP에서 파생된 언어들–ML과 같은 언어들에서는 성공적으로 타입 시스템을 구현하였다. 수업 시간에 익히 들어온대로 비록 그것이 완전complete 하지 않을지라도, 타입을 개발자가 알려주지 않아도 적절한 타입을 유추하고 프로그래머가 흔히 작성하는 프로그램에 대해서는 대부분 안전sound한 동작을 보장해주도록, '쓸만한' 타입 시스템이 구현되었다.

2.6 이제 실제 현장에서 사용해보자

맛있는 반찬을 먹어봅시다

"게임 프로그래밍 정글의 원시인들" 로부터

마지막으로, 이러한 잘 디자인된 언어(ML이나 LISP등)들이, 실제로 게임 프로그래밍에 어떻게 사용될 수 있을지에 대해서 살펴보자. 이 글에서 잘 디자인된 언어에서는 Vector나 Tree와 같은 기본 자료 구조가 사용하기 쉬우며, 컴파일러가 타입을 잘 찾아주고, 다형성을 이용하여 한가지 코드를 다양한 목적으로 사용할 수 있으며, 함수를 그 자체로 값으로 사용할 수 있고, 배열에 대해 안전하고, 디버깅을 자유롭게 할수 있다. 실제로 동일한 내용을 담은 코드가 C++보다 LISP에서 더 쉽게 표현됨을 보여주고, 흔히들 우려하는 성능(속도)도 충분히 최적화 되어 있음도 이야기 해준다.

3 맛도 모르고 먹던 반찬 이야기

늘 먹던 것은 그 것이 정말 맛이 있나 깨닫지 못하고 먹을 때가 많다. 독립을 해봐야 어머니 밥이 맛있는 것을 알 수 있듯이 말이다.

대부분의 경우, BASIC이나 C, C++을 이용하여(혹 좋은 사람을 곁에 두었다면 Java로) 언어를 처음 배우게 된다. 맛도 모르고 먹던 반찬들에 대해서, 맛이 있는지 없는지, 어떤 맛이 있는지 생각해 보고자 한다.

3.1 주류 언어들의 필요성

이제, 언어 세상의 다른 한 축을 이루고 있는 또다른 '주류 언어'들–C나 C++과 같은 언어는 어떻게 사용되고 있고, 왜 필요한지도 살펴보자. 구체적인 수치를 들지 않아도 충분히 체감할 수 있듯이, 이 언어들을 사용하는 곳들은 충분히 넘쳐날 정도로 많다. 대부분의 OS가 C나 C++을 기반으로 개발되며, 특별히 성능이 매우 중요시되는 프로그램에서는 이런 언어의 특성이 요소 요소 큰 역할을 하고 있다. 특별히 C나 C++과 같은 언어들을 사용하면, 기계 부품들을 그대로 만질 수 있기 때문에 이러한 일들이 가능하다.

병역 특례를 위해 다니던 회사에서 개발하던 프로그램들의 주기능은 디스크의 데이터를 다루는 것이었다. 파일을 복구하기도 하고, 전체 파일에 대한 검색도 종종 필요했으며, 디스크를 비교하는 등의 기능들도 들어있었다.

워낙 다루는 데이터 양이 방대하기 때문에 효율적인 캐쉬를 구현하는 것이 굉장히 중요한 부분을 차지하고 있었다. 실제로 프로그램 엔진 파트에는 디스크를 적절하게 잘라서 읽어서 캐쉬하는 모듈이 내장되어 있었다. 이런 상황에서 C, C++은 주 언어가 될 수 밖에 없었다.

3.2 언어의 남용

하지만, 주류 언어들은 너무 남용되고 있어, 문제들을 키우는 원인이 된다. 읽기 자료에서도 살펴 본 바 있듯이, 새로운 언어를 검토하고 받아들이는데 너무 게으르다. 엔진을 C나 C++로 개발했기 때문에, 다른 영역을 다른 언어로 개발한다는 것은 생각할 엄두를 내지 못한다. 게다가 막연한 두려움까지 가지고 있어(성능이 저하 될지도 모른다던가), 일단 지금 C와 C++의 개발자를 구하기 쉬우므로 프로그램 전체를 C나 C++로 구현해버린다. 그리고는 버그를 잡는데 밤을 새고, 언어가 너무 불친절한 부분이 많다고 투덜댄다.

일례로, Microsoft 에서 C++을 이용하여 GUI를 구현하기 위해서 만든 라이브러리인 MFC가 가진 문제점들에 대해 살펴보자. MFC는 초기 설계시에 클래스의 동적 생성과 실행시 타입에 대한 정보가 필요했다. 하지만, 이런 사항들은 C++에서 지원하지 않았기 때문에, 매크로와 C++ 표준을 이용해서 직접 이런 것들을 구현했다. 그리고, C에 익숙한 사용자가 더 많다는 이유와, Windows는 Message 기반으로 작동한다는 것을 손댈 수 있게 하기 위해, 인터페이스의 많은 부분에서 C++의 핵심인 OOP를 포기한다. 이러한 문제점 때문에, 많은 프로그래머들에게 MFC는 깔끔하게 사용하기도 어려우며 어느 정도 코드가 길어지면 관리하기가 까다롭다는 비판을 듣게 된다.

이보다 더 큰 문제는 비판만 하면서 주류 언어에 대한 타성은 벗어나지 못하고 있다는 것이다. 현재도 MFC는 계속 사용되고 있다. Java 진영이라고 해서 문제가 없는 것은 아니다. 일단 그 언어를 종교처럼 믿기 시작하면 다른 언어를 검토하지 않는다. 적어도 Windows 에서는 C#이라는 훌륭한 도구를 GUI 개발에 이용할 수 있는데, 쉽사리 현장에 파급되지 못하고 있다. ML이나 LISP은 말할 것도 없다.

3.3 언어의 맛을 모르고 사용하다

문제는 자신의 주력 언어만 맹신하여 다른 언어를 살펴보지 못하는 것에 있다. 자신이 맹신하는 언어들에 대해 올바르게 이해하지 못하고 사용하거나, 다른 언어의 맛을 알아볼 생각을 하지 못한다. 주류 언어가 어느 상황에서 적절하며, 다른 언어는 어떤 상황에서 적절한지 그 차이에 대하여 생각하지 못한다.

다른 언어를 살펴보지 못한다는 것은, 주력 언어도 제대로 사용하지 못한다는 것이랑 다른 말이 아니다. 언어의 맛을 모르기 때문에 C++을 OOP의 개념을 사용하지 못하고 C처럼 사용하는 개발자도 많다. 더불어 나중에 분명 불편할 코드들을(타입이 불안한 코드들을) 종종 만들어 내는 상황에 빠지기도 한다.

주류 언어가 맛이 없는 것은 아니다. 기계적인 뿌리에서 시작한 언어인 만큼, 기계를 보다 손쉽게 다룰 수 있는 힘을 가지고 있다. 여기에 더하여 C++같은 경우는 추상화된 구조를 썩 괜찮게 OOP를 이용하여 구현 할 수 있다. 어느 면에 있어서는 맛있는 언어일 수 있다.

하지만 기계적인 언어들은 값 중심의 언어와 비교하면, 느슨하고 부족한 부분이 많이 있다는 것도 인지하고 준비해야 한다. 더불어, 가능할 때 값 중심의 언어가 가진 강력한 자유로움을 사용할 수 있도록 늘 준비하고–각각의 맛을 음미하고 있을 필요가 있다.

편식은 몸에 나쁘다.

4 맛있는 반찬과 맛있는 밥상

4.1 주류 언어를 경계한다

위에서 충분히 설명했듯이, 주류 언어는 자칫 언어의 맛을 모르고 사용하기 쉽다. 이미 대부분이 부류의 언어를 익숙하게 다루고 있기 때문에, 이 언어로 모든 것을 할 수 있을 것처럼 보인다. 실제로 모든 것을 할 수 있을지라도, 그것이 최적의 해는 아닐 수 있으므로, 늘 고민하고 검토할 수 있어야 한다. LISP이나 ML에는 훌륭한 타입과 좋은 자료 구조들이 이미 정의되어 있으며, C#은 GUI를 만들면서 타입 체크와 Gabage Collection도 훌륭하게 처리해 줄 수 있다.

또한, 기계 중심적인 주류 언어들은 안전하지 않은 요소들을 가지고 있으므로 경계해야 한다. 일례로 C나 C++과 같은 언어의 타입 불안전한 문제에 대하여, 안전한 LISP이나 ML이 대응 방안이 될 수 있지 않을지에 대하여 늘 염두하고 있어야 한다.

4.2 LISP과 ML을 경계한다

그러나 LISP과 ML과 같은 값 중심 언어에 찬양에 빠져 만능이라고 생각한다면, 역시 맛을 모르고 사용하는 일이 될 것이다. 기계를 엄밀하게 다루기 위해서는 C나 C++과 같은 언어가 가지는 장점이 있다. 부품을 만들고 이를 조립하는 식으로 프로그램을 구현 할 수 있는 LISP과 ML이 적합한 상황이 있을 수 있고, 그렇지 않은 상황이 있을 수 있다. ML도 불편한 점이 있다. ML의 강력함의 이면에는 (숙제를 하면서 만났던) 'a type이 a type과 다릅니다' 메시지나, 디버깅의 어려움 등등도 감수해야 함을 알아야 한다.

4.3 언어를 자유롭게 사용할 수 있도록.

골고루 먹읍시다

언어는 도구에 불과하다. 언어는 사람의 생각을 표현하기 위한 도구일 뿐, 그 중심은 '사람의 생각'이다. 생각을 가장 효과적으로 표현할 수 있는 언어를 고르면 된다. 기계 중심적으로 처리해야 하는 일을 ML이나 LISP으로 할 수 없다거나, 수학적인 모델을 엄밀하게 적용하는 일을 C로 처리하지 못하는 것이 아니다. 서로 표현할 수 있는 '의미'의 영역은 동일하다. 어느 것이 어떤 상황 적절한지 각각의 맛을 알아두고 사용하면 된다. 도구라는 것은 그 용도에 맞게 사용했을때 비로소 빛을 발한다.

남은 과제는 문제를 해결해 나가기 위하여 어떤 도구를 사용할 수 있는지 잘 알고 익혀두는, 그리고 그리고 꺼내서 사용하는 힘이다. 각 반찬들이 어떤 맛을 가지고 있는지 잘 알았다면, 맛있는 밥상을 꾸미고 건강하게 먹는 것이다.

골고루 먹으면 건강해진다. 생각할 수 있는 여유가 많아지고, 더 적은 노력으로 생각을 표현하고 더 좋은 생각들을 할 수 있을 것이다.

4.4 맛있게 먹겠습니다

프로그래밍 과제에서 조교님이 만들어주시는 Skeleton Code를 살펴보다가 lex과 yacc이 사용되었다는 것을 보고 놀란 적이 있다. C나 C++에서도 lex, yacc, flex나 bison을 이용하여 간단한 장난감들을 만들어 본 적은 있지만, 토큰을 관리한다거나 문법과 문맥을 관리하는 등 세세한 관리를 요하는 부분에서 자연스럽지 않거나 불편한 점들이 많았다. ML은 ML = Meta Language 를 표현하는데 아주 적합한 언어이다. lex과 yacc이 아주 자연스럽게 언어로 녹아들고, 아무렇지도 않게 사용되는 과정을 훑어 보면서, '이 때는 ML을 사용하면 정말 좋겠구나' 라는 생각을 한 적이 있다.

실제로, 회사를 다니던 시절에, 스크립트 엔진을 만들 필요성에 대해서 검토하다가 개발 인력 문제로 포기한 적이 있었다. 조금더 이 수업을 접했다면 다른 생각을 할 수 있지 않았을까?

자유롭게 맛있는 밥상을 마주할 날을 생각해본다. 수업 시간에 종종 생각하는 것처럼, 기계랑 맞닿은 엔진은 C++로 OOP로 구조를 잘 쌓아올려 개발하고, 스크립트 언어를 ML로 개발하고, 자료들은 C#으로 개발해서 GUI로 붙이고. 이것 저것 반찬을 차려놓고 맛있게. 이런 생각들이 조금씩 쌓이면, 어느 날 교수님께서 수업 시간에 말씀하신 대로 '제가 이런 걸 이 언어를 사용해서 이렇게 만들어 보려고 하는데요. 어떻게 하면 좋을까요?'라고 여쭈어 볼 수 있지 않을까 잠시 생각해본다.