**스프링 입문을 위한 자바 객체 지향의 원리와 이해**

1. **사람을 사랑한 기술**

**It from bit (비트에서 존재로) – 존 아치볼트 휠러**

**프로그래밍 언어의 진화**

이진 언어 – 어셈블리어 – C언어

컴퓨터가 인식할 수 알파벳은 0과 1뿐이다. 있음과 없음이 교차하는 가운데 무수히 많은 가능성이 생겨났다. 그러나 인간은 태생적으로 있음과 없음으로만 이루어진 세상을 살아갈 수 없었다. 우리는 있음과 없음의 교차로만 존재하는 단조로운 진화 과정을 따르지 않았다. 있음과 없음의 교차 조합은 집단화되고, 의미가 부여되어 우리가 인식하는 세상이 되었다.

모든 프로그래밍 언어의 목적 언어는 0과 1로 된 기계어이다. 사람이 어떤 방법으로 코드를 쓰든 코드는 결국 0과 1이 되기 위해 태어난다. 이진 언어로 코딩을 하던 시절에는 사람이 직접 0과 1을 입력해 코딩했다. 그러다가 프로그래머들은 우리의 진화가 그러했듯 0과 1의 교차 조합을 구분했고 거기에 1:1로 대응하는 일상의 언어를 붙여 의미를 부여했다. 어셈블리어의 탄생이었다. 사람들은 0과 1의 교차집합과 일상의 언어가 1:1로 매칭된 어셈블리(Assembly)라는 표를 들여다보며 코딩하기 시작했다.

모리스 월키스가 제작한 초기의 컴퓨터 애드삭(ADSAC)에서 기계어 01010101은 더하기를 의미했다. ‘더하기’라는 하나의 단어는 01010101에 대한 니모닉(Mnemonic)이 되었다. 그리스 신화 속 기억의 여신인 ‘므네모시네’의 이름에서 따온 말, ‘니모닉’은 기억하기 어려운 대상을 기억하기 위해 사람들이 부여한 낱말 혹은 정보의 집합이 되었다. 니모닉의 발전은 프로그래머의 생산성에 크게 기여했다. 어셈블리어의 탄생이었다. 그러나 초기의 컴퓨터 애드삭과 유니박은 어셈블리어 체계가 달랐다. 우리가 차이를 좋아하기 때문에 어셈블리어가 달랐던 것만은 아니다. 애드삭과 유니박은 기계어 또한 달랐고 이에 따라 유니박을 사용하는 프로그래머가 애드삭 프로그램을 개발하기 위해서는 새로운 어셈블리어를 처음부터 다시 배워야 했다. 새롭게 작성한 어셈블리어는 다시 기계마다 다른 어셈블러를 거쳐 0과 1로 된 이진 언어로 바뀌었다. 기계어와 어셈블리어는 하드웨어에 종속적이었던 것이다. 어셈블리어로의 발전 이후, 상승하던 프로그래머의 생산성 곡선은 기울기가 점점 감소하여 어느덧 고원에 도달해갔다.

문제가 무르익어갔다. 이 문제가 무르익어, 터져서 나온 과실이 C언어였다고 말한다면 과장일까? C언어는 하드웨어마다 새 언어를 배워야했던 개발자들에게 축복과 같았다. 어셈블리어보다 덜 복잡했으나 한 번의 작성만으로 모든 하드웨어에서 실행이 가능했다. 이는 새롭게 등장한 컴파일러 덕분이었다. 컴파일러는 C언어로 쓰인 명령을 어셈블리어와 기계어로 번역해주는 역할을 했다. 이제 프로그래머들은 하드웨어의 제한에서 어느 정도 벗어나 한 번의 코딩으로 모든 컴퓨터에서 작동하는 프로그램을 만들 수 있었다. 이렇게 된 배경에는 하드웨어 기술의 발전도 있었다. 초기 컴퓨터에는 메모리 공간이 부족했기에 컴파일에 대한 아이디어가 있어도 컴파일러를 저장하고 실행할 수 없었기 때문이다. 고원에서 벗어난 개발자들의 생산성 곡선이 다시금 상승세를 타고 우상향했다. 물론 C언어가 정말 단 한 번의 작성으로 모든 프로그래밍의 문제를 해결하는 마스터키였던 것은 아니다. C언어의 컴파일러는 하드웨어와 직접적으로 소통하는 것이 아니라 하드웨어의 특성을 추상화한 운영체제를 거쳐서 소통했다. 따라서 C언어 개발자가 운영체제에 맞는 각기 다른 컴파일러를 만들었다 하더라도 각각의 컴파일러는 다시 운영체제에 종속적일 수 밖에 없었다. 만일 어떤 운영 체제는 int라는 정수형 데이터의 메모리를 2바이트로 정의하는데 다른 운영 체제가 int를 4바이트로 정의한다고 가정해보자. 이렇게 되면 동일한 C 코드로 작성된 프로그램이어도 운영체제에 따라 다른 퍼포먼스를 보이게 된다. 이와 같은 이유로 C언어로 작성된 코드는 운영체제에 따른 수정이 불가피했다. 그러나 그럼에도 기계어나 어셈블리어에 비하면 C언어는 진화한 언어였다. 코드의 작성을 유용하게 했을 뿐 아니라 구조적 프로그래밍이라는 새로운 패러다임을 제시했기 때문이었다. 함수로 대표되는 구조적 프로그래밍은 2장에서 다루게 된다.

**진정한 객체 지향 언어 자바**