프로그래밍 언어론

Report #3

2017.04.03

20154073

강다현

Q. 2장의 언어를 개발 배경, 설계 과정, 언어 개요(특징, 구현방법 포함),   
평가의 순으로 정리하라.

**🏵Fortran**

**개발 배경**

Fortran이 개발되기 이전에는 사용 가능한 컴퓨터들 중에는 부동-소수점 하드웨어가 없어 소프트웨어로 부동-소수점을 처리에 많이 시간이 소요되었기 때문에 그 당시의 프로그래머들은 기계어의 효율성을 선호하면서 해석 시스템을 사용하지 않았습니다. 하지만 하드웨어에 의한 첨자와 부동-소수점 명령어들을 갖는 IBM 704 시스템과 함께 Fortran이 사용되었다.

**설계 과정**

Fortran이 개발된 환경은 첫 번째로 컴퓨터 메모리 용량이 작고 느렸고, 상대적으로 신뢰성이 없었다. 두 번째로 컴퓨터는 주로 과학 계산 분야를 위한 것이었다. 세 번째는 프로그래밍을 하기 위해 존재하는 효율적이고 효과적인 방법이 없었다. 네 번째는 프로그래머의 비용과 비교했을 때 컴퓨터가 고비용이었다. 이러한 이유로 Fortran는 이러한 환경을 반영해서 설계되었다.

**언어 개요**

Fortran 0는 1995년 1월에 구현이 시작될 때부터 1957년 4월 컴파일러가 발표될 때 까지 수정되었다.

Fortran Ⅰ의 모든 제어문은 704 명령어들에 기반하였고, 데이터 타입 문장이 없었다. 이름이 I, J, K, L, M, N으로 시작되는 변수는 묵시적으로 정수 타입이고, 다른 모든 변수는 묵시적으로 부동-소수점 타입이었다. 또한 Fortran은 컴파일러에 의해 생성되는 기계 코드가 직접 코딩으로 생성할 수 있는 코드의 절반 정도 효율적일 것이라는 것이다. Fortran 개발 그룹은 컴파일러를 구성하는데 있어 최적화에 가장 많은 노력을 쏟았고 그 결과 상당히 효과적이었다.

Fortran Ⅱ 컴파일러는 Fortran Ⅰ 컴파일 시스템에 포함된 많은 버그를 수정하고, 다른 특징들을 추가했는데, 서브루틴들의 독립적인 컴파일 기능이 그것이다. 독립적인 컴파일 기능으로 프로그램의 일부 변경이 전체 프로그램이 다시 컴파일 되지 않아도 되었다. 미리 컴파일 된 기계어 버전을 포함시키는 능력은 컴파일 과정을 상당히 단축시켜, 훨씬 큰 프로그램을 개발하는 것을 실질적이게 했다.

FortranⅢ은 개발은 되었으나, 널리 배포되지 않았고, 다음의 Fortran Ⅳ은 그 당시에 가장 널리 사용된 프로그래밍 언어들 중 하나가 되었다. 이 Fortran Ⅳ는 1966년 Fortran 66으로 표준화되었는데, 변수에 대한 명시적 타입 선언, 논리 If문, 그리고 부프로그램을 매개변수로서 다른 부프로그램에 전달할 수 있는 능력 등이 있었다.

Fortran Ⅳ는 1978년 새로운 표준이 된 Fortran 77로 대체되었는데, Fortran 77은 Fortran Ⅳ의 대부분의 특징을 포함하고, 문자 스트링 처리, 논리 루프 제어문, 그리고 선택적인 Else 절을 갖는 If 등을 추가했다.

Fortran 90은 Fortran 77과 상당히 달랐는데 동적 배열, 레코드, 포인터, 다중 선택문 , 모듈 등이 추가되었고, Fortran 90의 부프로그램들을 재귀적으로 호출될 수 있었다. 다만 Fortran 77에서 리스트 구조는 삭제되었다. 또한 Fortran 90은 두 개의 간단한 구문을 변경하였는데, 첫 번째로 문장들의 특정 부분에 대해 특정 문자 위치 사용을 요구한 고정된 코딩 형식이 삭제되었다. 두 번째로는 FORTRAN 공식 철자가 Fortran이 되었다는 것이다. 새로운 규약에서는 키워드와 식별자에서 첫 번째 문자만이 대문자라는 것이다.

Fortran 95는 이전에서 몇 개의 변경만 이루어졌다. 새로운 반복구조 Forall이 Fortran 프로그램을 병렬화하는 작업을 수월하게 하기 위해 포함되었다.

Fortran 2003은 객체-지향 프로그래밍, 매개변수화 된 파생 타입, 프로시저 포인터, 그리고 C 프로그래밍 언어와의 상호 연동성 등을 위한 지원을 추가하였다.

가장 최근 버전의 Fortran 2008은 지역 영역을 정의하는 블록과 병렬 실행 모델을 제공하는 co-array, 그리고 상호종속성이 없는 루프를 명세하기 위한 DO CONCURRENT 구조 등을 위한 지원을 추가했다.

**평가**

Fortran은 모든 후속 프로그래밍 언어들이 Fortran의 영향을 받았다는 사실과 컴퓨터 사용에 미친 효과는 실로 엄청난 것이다. 또한 Fortran Ⅰ과 90 사이의 버전들이 갖는 특징인 모든 변수에 대한 타입과 기억공간이 실행 시간 전에 고정되고, 실행 중간에 새로운 변수나 기억공간이 할당될 수 없었다는 것은 단순성과 효율성을 위해 유연성을 희생한 것이다. 이로 인해 재귀적 부프로그램의 가능성을 제거하고, 동적으로 형태를 키우거나 변경하는 데이터 구조를 구현하는 것을 어렵게 했다. 이러한 것이 가능했던 이유는 초기 버전의 개발 당시에 생성되었던 프로그램은 주로 수치적이었고, 최근 소프트웨어 프로젝트에 비해 단순했기에 희생이 크지 않았다. 그리고 Fortran의 초기 버전들은 다방면에서 어려움을 겪는다. 그러함에도 불구하고 Fortran 소프트웨어에 대한 상당한 투자에 힘입어 반세기 이상 동안 계속 사용되어 왔다. Alan Perlis는 Fortran은 중요한 사업에서 매우 유용한 부분이라고 판명되었기 때문에 살아남았고, 앞으로도 살아남을 것이라고 했다.

**🏵Ada83**

**개발 배경**

Ada는 미 국방성(DoD)을 위해 개발되었다. 1974년 DoD에서는 컴퓨터 응용의 절반 이상이 내장 시스템이었는데, 450개 이상의 다른 프로그래밍 언어들이 DoD 프로젝트를 위해 사용되었지만, 어떠한 언어도 DoD에 의해 표준화되지 않았다. 이러한 점 때문에 응용 소프트웨어는 거의 재사용되지 않았고, 어떠한 소프트웨어 개발 도구도 생성되지 않았다. 또한 상당히 많은 언어가 사용되고 있었지만, 어떤 언어도 내장 시스템 응용에 적절하지 않았다. 이러한 이유로 1974년 내장 시스템을 위한 한 개의 고급 언어의 개발을 제안했는데 이 언어가 Ada이다.

**설계 과정**

1975년 Malcolm Currie가 HOLWG를 결성하였다. HOLWG의 초기 헌장은 새로운 DoD 고급 언어를 위한 요구사항들을 식별, 가능한 후보 언어가 있는지를 결정하기 위해 기존 언어들을 평가하고, 프로그래밍 언어들의 최소한의 집합의 채택이나 구현을 권장하는 것이었다. 이를 수행하기 위해 HOLWG는 새로운 언어를 위한 요구 사항 문서를 생성했고, 이 문서는 군의 부서들, 연방정부 기관들, 선택된 산업체와 대학 대표자들, 그리고 유럽의 관심 있는 업체들에게 배포되었는데 이 언어를 위해 제출된 많은 제안서들이 평가 과정을 거쳐 총 4개로 좁혀진다. 이 언어들은 Pascal에 기반되었다. 1979년 해군 군수품 사령부 Jack Cooper가 그 이름을 Ada로 제안하고, 이것이 채택되었다. 그리고 Ada는 10,000명 이상의 독자에게 배포되어 공개적인 테스트와 평가를 위한 학술회의가 개최되었고, 보고서를 바탕으로 수정되었다.

**언어 개요**

첫째로 Ada 언어의 패키지는 데이터 객체, 데이터 타입에 대한 명세, 그리고 프로시저들을 캡슐화하기 위한 수단을 제공한다. 프로그램 설계에서 데이터 추상화의 사용을 위한 지원을 제공한다.둘째로 Ada 언어는 예외 처리를 위한 많은 기능들을 포함한다. 이것은 실행-시간 오류들 중 임의의 한 개가 탐지된 후에 프로그래머가 제어하는 것을 허용한다. 셋째로 Ada에서 프로그램 단위들이 포괄적일 수 있다. 예시로 여러 타입의 데이터에 사용하는 정렬 프로시저를 사용하는 것이 가능하다. 포괄적 프로그램 단위들의 유용 가능성은 프로그래머에 의해 중복되기보다는 재사용될 수 있는 프로그램 단위들의 범위를 증가시킨다. 넷째로 Ada 언어는 랑데부 메커니즘을 사용해 태스크라 명명된 특정 프로그램 단위들의 동시 실행을 제공한다. 랑데부는 태스크 간의 통신과 동기화 방법의 이름이다.

**평가**

Ada 언어는 1970년대 후반의 소프트웨어 공학과 언어 설계의 대부분 개념들을 포함한다. 이러한 것에 대해 대부분의 사람들은 그 특징들이 유용하다는 데 동의한다. 설계가 완성된 지 거의 4년이나 지난 1985년 실제로 유용한 Ada 컴파일러가 나타났다. Ada는 한 동안 언어 규모가 너무 크고 복잡하다는 비평을 받았다. 하지만 다른 사람들은 Ada를 그 당시의 언어 설계의 전형으로서 긍정적으로 평가했다.

**🏵PL/I**

**개발 배경**

PL/I도 Fortran과 마찬가지로 IBM 생산품으로 개발되었다. 1960년대 초반, 산업계의 컴퓨터 사용자들은 분리되어 과학 프로그래머들과 사무 응용 프로그래머들로 나뉘어져 있었다. 과학 프로그래머들은 부동-소수점 데이터 타입과 배열들을 많이 사용하여 주 언어는 Fortran이었다. 반대로 사무 응용 종사자들은 세련되고 효율적인 입출력 기능과 십진수, 문자 스트링의 데이터 타입을 필요로 했고 주로 COBOL을 사용했다. 이 두 그룹은 초반에는 접촉이 없었으나, 이 후 과학자들은 큰 데이터 파일을 모으기 시작했고 이러한 데이터는 세련되고 효율적인 입출력 기능을 요구했고, 사무 응용에서 일하는 사람들은 부동-소수점 데이터와 배열들이 필요해 졌다. 이러한 점에서 부동-소수점과 십진수 산술을 처리할 수 있고, 과학과 사무 응용 모두를 처리할 수 있는 한 개의 보편적인 컴퓨터 개념을 야기했다. 따라서 IBM System/360 컴퓨터 계열의 개념이 탄생했고, 이와 더불어 사무와 과학 응용 모두에 대해 사용될 수 있는 프로그래밍 언어의 아이디어가 탄생했다.

**설계 과정**

초기 설계 개념은 호환성을 유지하는 Fortran Ⅳ의 확장일 것이었으나, 그 목적은 곧 포기되었다. 1965년까지 이 언어는 NPL로서 알려졌다. NPL에 대한 첫 번째 보고서는 1964년 3월에 열린 SHARE 회의에서 배포되었고, 4월에 더 보충된 보고서가 간행되었다. 실제로 구현될 버전이 1964년 12월에 영국에 있는 IBM Hursley 연구소의 그룹에 의해 간행되었다. 1965년, NPL이라는 이름이 영국의 국립 물리학 연구소와 혼동되는 것을 피하기 위해 PL/I로 변경되었다.

**언어 개요**

PL/I는 프로그램들이 동시에 실행하는 부프로그램들을 생성하는 것이 허용되었다. 이는 PL/I에서 효율적으로 개발되진 못했다. 다음으로 23개의 다른 종류의 실행-시간 오류(예외)들을 탐지하고 처리하는 것이 가능했다. 또한 부프로그램들이 재귀적으로 사용되는 것이 허용되었다. 그러한 기능은 허가되지 않을 때가 있었는데, 이는 비재귀적 부프로그램들에 대한 더 효율적인 링킹을 허용했다. 다음으로 포인터가 데이터 타입으로서 포함되었고, 마지막으로 배열의 일부분이 참조될 수 있었다.

**평가**

PL/I는 전반적으로 유용하고 구현될 수 있는 구조를 포함해야 한다는 전제에 기반하고 프로그래머가 어떻게 구조와 특징들을 이해하고 효과적으로 사용할 수 있는가에 대한 고려가 충분치 않았다. 큰 규모로 인한 복잡성을 갖는 문제 외에도, PL/I는 잘 설계되지 못했다고 생각되는 구조들을 많이 포함했다. 포인터, 예외 처리, 그리고 동시성이 그것들이다. 1970년대 이 언어는 사무와 과학의 모든 응용에서 널리 사용되었고, 또한 그 당시에는 다양한 부분의 집합의 형식으로 대학에서 교육 도구로서 널리 사용되었기 때문에 사용자의 관점에서는 PL/I는 적어도 부분적 성공이라고 고려되어야 한다.

**🏵Smalltalk80**

**개발 배경**

Smalltalk의 개발을 야기한 개념들은 Alan Kay 박사 학위 논문 연구에서 비롯되었다. Kay는 데스크톱 컴퓨터가 비 프로그래머에 의해서 사용될 것이고 따라서 강력한 휴먼 인터페이스 기능들이 필요할 것이라고 생각했다. 1960년대 후반의 컴퓨터들은 주로 일괄 처리 방식이었고, 전문 프로그래머나 과학자들에 의해 독점적으로 사용으로 사용되었다. 비 프로그래머에 의한 사용을 위해, 컴퓨터는 상당히 대화적이어야 하고, 그 사용자와 인터페이스에서 정교한 그래픽을 사용해야 할 것이라고 생각했다.

**설계 과정**

Kay는 연구센터에서 Dynabook(범용 정보 처리기)에 대한 아이디어를 제출해, 센터에 고용되어 학습 연구 그룹이 만들어졌는데, 이 그룹은 Kay의 프로그래밍 패러다임을 지원하는 언어를 설계하고, 그 언어를 당시에 유용 가능한 가장 좋은 개인용 컴퓨터에 구현하는 것이었다. 이 결과로 Xerox Alto 워크스테이션과 Smalltalk-72 소프트웨어로 구성된 Dynabook이 탄생되었다. 이들은 아동에게 프로그래밍을 가르치는 여러 가지의 실험을 포함하여, 많은 연구 프로젝트가 이 시스템과 함께 수행되는데, 실험과 더불어 여러 버전의 언어를 추가 개발했다. 이 마지막 버전이 Smalltalk-80이다.

**언어 개요**

Smalltalk은 오직 객체들로 이루어 져 있다. Smalltalk에서 모든 계산을 동일하고 균일한 기법에 의해 이루어 진다. 즉 객체의 한 메소드를 호출하기 위해 그 메소드에 메시지를 전달하고 메시지에 대한 응답도 객체가 된다.메시지는 데이터 객체, 특정적으로 그 객체에 대해 정의된 한 메소드에 전달되고, 다음 호출된 메소드가 실행되고, 메시지가 전달된 객체에 속한 데이터를 수정한다. Smalltalk에서 객체 추상화는 클래스이며, 구문은 대부분의 다른 프로그래밍 언어와 유사하지 않다. 대부분 산술과 논리식, 통상적인 제어문들보다는 메시지의 사용 때문이다.

**평가**

Smalltalk은 컴퓨팅의 그래픽 사용자 인터페이스와 객체-지향 프로그래밍을 발전시켰다. 지금의 소프트웨어 시스템에 대한 사용자 인터페이스의 대부분인 방법인 윈도우 시스템은 Smalltalk으로부터 시작되었다. 또한 오늘날 가장 중요한 소프트웨어 설계 방법론과 프로그래밍 언어는 객체 지향적이다. 객체-지향 언어의 아이디어는 SIMULA 67로부터 나왔지만, Smalltalk은 최초의 순수 객체 지향 언어이기 때문이다.

**🏵Swift**

**개발 배경**

애플은 Object-C만으로도 아이폰과 아이패드, OS X 앱을 모두 개발할 수 있는데 굳이 애플이 스위프트 언어를 개발한 이유는 첫째로 미국을 비롯한 IT를 배우는 전 세계 학생들이 스위프트를 통해 재밌고 즐겁게 앱을 개발할 수 있는 애플의 미래 개발자 생태계 구축, 둘째로 스위프트 출시가 최근 실리콘 밸리의 린 스타트업 방식에 영향을 받아 스타트업이 자신만의 아이디어를 빠르게 앱 프로토타입에 적용시켜 성공 여부를 검증하기 위함이다.

**설계 과정**

Object-C는 애플의 대표적인 프로그래밍 언어로 사용되었다. 이 Object-C 언어는 C, C++에 기반을 두고 있지만, 사실 Smalltalk의 영향을 더 많이 받았다. 그리고 Object-C가 C언어에 기반한 만큼 포인터, 레퍼런스, 수많은 객체 지향 패러다임의 클래스와 다중 상속 등 개념을 정확히 알아야만 프로그래밍이 가능했다. 특히 객체 지향 프로그래밍 교육에만 수개월이 걸리는데, 그래서 Swift는 Object-C 언어의 한계를 넘고자 익히기 쉽고 쓰기 쉬운 애플의 새로운 컴퓨터 프로그래밍 언어로 탄생했다.

**언어 개요**

Swift는 기존 Object-C와 완벽한 호환이 가능해 Object-C를 사용하면서 동시에 Swift 언어 개발이 가능하고, Object-C보다 속도가 최대 2.6배 빠르다. 또한 객체 지향 언어이면서 동시에 함수형 언어이며, 컴파일 과정을 거치지 않는 스크립트 언어이다. 데이터 타입이 존재하며 타입간 구분이 엄격하다.

**평가**

아직 나온 지 얼마 되지 않았고, 주로 아이폰/iOS 개발에만 사용되고 있지만, 프로그래밍 언어순위에서 Object-C와 같이 랭크되는 등 빠른 성장세를 보인다. 또한 경쟁사인 구글도 Java의 저작권을 둘러싼 오라클과의 분쟁과 영향으로 Swift로의 전환을 고려한다는 말도 있었다. java와 C#의 장점을 거의 가지고 있으면서도 둘 다 LLVM을 기반으로 C/C++/Object-C처럼 기계어로 최종번역된다는 점에서 성능상의 우위가 있다.

**🏵Ruby**

**개발 배경**

Ruby는 1995년 마츠모토 유키히로가 개발한 동적 객체 지향 스크립트 프로그래밍 언어이다. 루비는 “스트레스가 없는 쉬운 프로그래밍을 지향하고 우리의 삶은 반영해야 하며, 프로그래밍 언어는 컴퓨터가 아닌, 인간 중심으로 설계 되어야 한다.”는 마츠모토 유키히로의 철학으로 개발되었다.

**설계 과정**

Ruby는 파이썬 등의 언어로부터 일부 문법 사항을 계승하고 발전시켜 일반적 목적에 적합하게 만든 언어이다. 언어의 문법이 쉽고 확장이 편리해서, 잘 디자인된 라이브러리를 이용하면 처음 시작한 사람도 복잡한 작업을 상대적으로 쉽게 할 수 있다. 또한 객체 지향 프로그래밍 언어지만, 객체 지향에 대한 흔한 선입견과 달리 초보자들도 쉽게 다룰 수 있다.

**언어 개요**

Ruby는 우선 순수 객체지향언어로 모든 것이 객체인 언어로 원시 타입이 존재하지 않는다. 그 외에도 모듈이 존재하는데 모듈도 역시 클래스이며 이 클래스는 모두 열려있는 OpenClass로 사용자가 마음대로 수정하는 것이 가능하다. 특이한 점은 객체지향 언어이면서 함수형 언어의 특징도 가진다. 또 요즘 고급언어들이 지원하는 람다도 포함되어 있다. 그리고 언어 자체를 쉽게 확장시킬 수 있는 메타 프로그래밍을 지원하여 매우 큰 프로그램도 작은 소스로 가능하게 한다.

**평가**

초반에는 일본에서 조금 쓰이는 정도였으나 Ruby on Rails라는 웹 프로그래밍 프레임워크가 나오면서부터 인기를 얻게 되었다. 초보자들도 쉽게 다룰 수 있도록 만들어 져 배우기가 쉽고 웹과 front, back end 심지어 임베디드까지 사용 용도가 다양하고, 확장성과 이식성이 높지만 반면에 속도가 느리고, 기본 API들이 일본어로 작성되어 있고 국내에서는 잘 이용하지 않아 관련 서적이나 자료를 찾기 어렵다.

**🏵Rust**

**개발 배경**

2006년 개발자 그레이든 호아레의 개인 프로젝트로 시작되었으며, 2009년 호아레의 고용주인 모질라가 개발에 참여했다. 2010년에 처음으로 일반에 공개되었으며, 같은 해 Objective Caml로 작성된 초기 컴파일러에서 Rust로 작성된 컴파일러로 전환되었다. rustc는 2011년 처음으로 자기 자신을 컴파일 하는데 성공했다. 2012년 1월 첫 번째 알파 버전인 0.1이 발표되었다. 모질라의 정책에 따라, 러스트는 전적으로 오픈소스로 개발되고 있으며, 커뮤니티로부터 피드백을 받고 있다.

**설계 과정**

모질라에서는 새로운 엔진을 제작하기 위해 여러 가지 언어를 비교했다. 우선 C++의 경우 엔진의 실행속도를 따라 잡을 수 있지만 제작속도 및 용이성이 떨어지는 문제점이 있었고, 새로운 엔진을 제작할 때 멀티 프로세싱을 하기 위해 쓰레드를 사용해야 했는데, C++의 쓰레드는 숙련된 C++ 프로그래머가 아니면 사용하기가 매우 어렵고, 숙련된 C++ 프로그래머는 값이 비쌌다. 다음으로 Java는 쓰레드 사용이 C++보다 용이하고 메모리 관리도 보다 편했지만, 가상머신 위에서 돌기 때문에 속도가 느리다는 단점이 있어 곤란했다. 이러한 장단점이 있어 곤란할 때 모질라는 엔진의 실행속도, 제작속도 및 제작의 용이성에 맞게 새로운 언어를 만들자는 역발상을 하게 되었고, 소속 개발자였던 그레이던 호아레의 개인 프로젝트인 Rust를 발견하고 지원하게 되었다.

**언어 개요**

Rust의 문법은 중괄호로 코드 블록을 구분하고 키워드를 사용하는 등 C와 C++과 유사한 모양이나 의미상으로는 크게 다른 문법을 갖고 있다. 또한 메모리 오류를 발생시키지 않도록 하기 위해NULL 포인터나 초기화되지 않은 포인터가 존재하지 않도록 강제하고 모든 변수는 초기값을 가지고 할당된다. 타입 시스템은 하스켈의 영향을 받아 타입 클래스를 지원한다. 변수 선언지 타입을 지정하지 않아도 컴파일러가 타입 추론을 통해 해당 변수의 타입을 지정할 수 있다. 병행 프로그래밍은 얼랭과 유사한 actor 기반의 모델을 하고 있다. 각각의 태스크는 데이터를 직접 공유하지 않고 메시지 전달을 통해서만 데이터를 교환할 수 있다. 메시지 복사 시 성능 저하를 막기 위해, unique box의 경우 데이터 복사 없이 메시지를 전달하기 위해 소유권만 변경된다. 객체 시스템은 implementation, trait, struct 또는 enum으로 이루어 지는데 struct와 enum은 객체의 데이터를 정의하는데 사용되고, implementation은 객체의 멤버 함수를 정의하는데 사용된다. 객체의 상속은 trait을 통해 이루어지며, C++의 다형 상속 문제를 피하기 위해 trait은 객체의 멤버를 정의할 수 없다.

**평가**

Rust는 C++의 대체 언어라 불리는데 그만큼 C++의 문제들을 해결할 수 있도록 설계되었다 특히 메모리 오류를 잡아주는 것이 가장 큰 장점이다. 동시에 LLVM이라는 컴파일러 구조를 통해 최적화를 지원하여 메모리 할당과 해제 시점을 프로그래머가 직접 제어할 수 있어 사용자에게 편리함을 준다. 하지만 Rust는 소유권이라는 독특한 개념이 있어 초급 개발자들이 배우는데 어려울 수 있다.

🖉느낀점

이렇게 많은 언어를 찾아보면서 세상엔 똑똑한 사람들이 많아서 언어를 많이도 만들어 내는구나 라는 것을 느꼈고, 다양한 특징들을 필요에 의해 저런 특징들을 생각해 낸 것이 정말 대단하다고 생각했습니다.