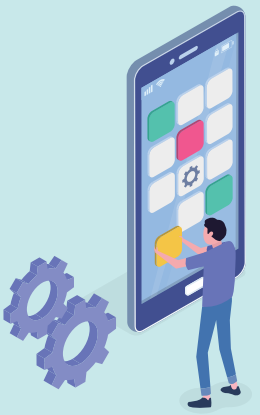

변화하는 프로그래밍 언어. '함수형 프로그래밍'이 뜬다

Introduction to Functional Programming and Supported Language Trends

김정민 Kim, Jung Min • **연구원** Researcher, SPRi • jungmink26@spri.kr

최근 객체지향 프로그래밍의 대안으로 함수형 프로그래밍(Functional Programming)이 다시 주목받고 있다. 함수형 프로그래밍은 객체(Object) 단위로 문제를 분해하지 않고 함수(Function)들의 집합으로 문제를 분해하는 방법론으로, 객체지향 프로그래밍이 가진 문제점을 극복할 수 있는 장점을 지닌다. 최근 JAVA, 파이썬 등 기존 개발자에게 친숙한 언어 또한 함수형 프로그래밍을 지원하도록 변화하는 추세로, 미래를 위해 관련 교육에 대한 고려가 필요한 시점이다.

Recently, functional programming has emerged as an alternative to object-oriented programming. Functional programming is a method of decomposing a problem into a set of functions without decomposing the problem in units of objects, and has the advantage of overcoming the problems of object-oriented programming. Recently, languages familiar to existing developers such as JAVA and Python are also changing to support functional programming, and it is time to consider related education for the future.

함수형 프로그래밍, 획일화된 프로그래밍 기법에 다양성을 부여하다

SW개발자라면 절차적(Procedural) 또는 객체지향(Object-oriented)이라는 용어에 친숙할 것이다. 오늘날 SW개발 원리의 기초가 되는 이론이면서도 현업에서 SW개발자의 실력을 판가름하는 지표로서 활용되기 때문이다.[1] 이 중 객체지향 프로그래밍(Object-Oriented-Programming, 이하 OOP)은 효율적인 개발 패러다임으로서 매우 오랜 시간 준용되며 SW개발자가 반드시 익혀야 할 방법론으로 인지되었다.

최근 SW개발자들이 앞서 언급한 잘 정립된(Well-Defined) SW개발 패러다임이 존재함에도 함수형 프로그래밍(Functional Programming, 이하 FP)에 주목하는 사례가 종종 등장하고 있다. 유명 SW분야 서적 "The Pragmatic Programmer"의 저자 Andrew Hunt는 2019년 9월 언론과의 인터뷰서 병렬컴퓨팅의 어려움을 해결하는데 FP가 유용하다 언급하였으며,[2] 국내 모바일 게임 '달빛조각사'는 FP기반 언어인 엘릭서(Elixir)로 개발한 운영서버를 기존 게임과의 차별성으로 내세워 화제가 되기도 하였다.[3] 한발 더 나아가 함수형 언어 자체를 신규 제작하는 작업도 진행 중인데, 마이크로소프트(Microsoft)는 2019년 3월부터 FP언어를 개발하는 프로젝트인 'Bosque Language'[4]를 추진 중이다.

이처럼 FP는 객체지향 패러다임이 지배하고 있는 SW시장에 활기를 불러일으키고 있다. 본고에서는 기존 객체지향 패러다임이 가진 특성과 한계점에 대해 살펴보고, FP의 전반적인 개념 및 이를 지원하는 프로그래밍 언어에 대해 살펴보고자 한다.

프로그래밍 패러다임의 변화 흐름

OOP는 절차적 프로그래밍(Procedural Programming, 이하 PP)에 대한 대안으로 등장하였다. PP란 소프트웨어가 동작하는 순서에 맞춰 순차적으로 기능을 코딩하는 방법이며, 포트란(Fortran), 알골(ALGOL) 등으로 대표되는 초기 프로그래밍 언어가 이에 해당한다.

PP는 소프트웨어가 동작하는 절차에 주안점을 두기 때문에 소프트웨어가 반복된 기능을 수행해야 할 시, 코드가 중복되는 문제점이 있었다. 이를 해결할 수 있는 명령어가 'GOTO'[5]였는데, 소프트웨어가 다음 번에 실행해야 할 소스코드 지점을 자유로이 제어할 수 있어 중복된 소스코드를 줄이고 코드의 유연성을 가져다주었다.

1 http://www.econovill.com/news/articleView.html?idxno=361625
2 https://www.techrepublic.com/article/the-pragmatic-programmer-classic-developer-handbook-is-retooled-for-the-21st-century/
3 http://www.inven.co.kr/webzine/news/?news=219071
4 https://github.com/microsoft/BosqueLanguage
5 GOTO문은 초기 절차적 프로그래밍 언어에서 활용되던 명령어로서, "GOTO + 목적지" 형태를 통해 소스코드 실행 순서를 자유로이 바꿀 수 있었다.

그러나 소프트웨어의 규모가 상승하고 복잡성이 높아지면서 'GOTO'문의 활용이 유지보수에 악영향을 끼치기 시작한다는 지적이 제기되었다. 에츠허르 다익스트라가 1968년 발표한 GOTO문의 해로움[6] 논문이 대표적인 연구 결과로 'GOTO' 명령어가 코드의 질을 하락시킴을 지적하였다.

이후 프로그래밍 언어는 'GOTO' 명령어의 사용을 지양하기 위해, 작은 단위의 프로시저와 구조체(Structure)[7] 사용을 지원하는 방향으로 개선되었다. 구조체는 추후 객체(Object)라는 추상적 개념을 담을 수 있는 클래스(Class) 개념으로 발전하였으며 이 변화는 곧 OOP의 등장을 알렸다. 이윽고 OOP에 상응하는 언어(1972년: 스몰토크, 1983년: C++, 1995년: Java)가 연달아 발표되어 현재까지 프로그래밍 언어의 주류가 되고 있다.

객체지향 프로그래밍의 특성과 한계

OOP는 상태(State)를 객체(Object) 내 변수(Variable)로 간주한다.[8] PP가 과일을 표현하기 위해 과일이란 이름의 변수와 더불어 다양한 과일의 상태(색상, 모양, 맛 등)를 별도로 정의(예: 과일_색, 과일_맛, 과일_모양 등)했다면, OOP는 과일을 하나의 객체로 보고 과일을 표현할 수 있는 다양한 속성(Attribute)들을 객체 내 변수로 선언한다. 즉 OOP는 소스코드의 구성요소를 객체라는 개념으로 구조화시켜 복잡하고 규모가 큰 소프트웨어도 개발 및 관리하기 용이하다는 장점을 가진다. 이처럼 OOP는 여러 객체들을 조합해서 큰 문제를 해결하는 상향식(Bottom-up) 해결법이란 장점이 있으나, 그럼에도 불구하고 다음과 같은 몇 가지 문제점이 존재한다.

- **함수의 비일관성** : OOP에서 사용되는 함수(Function)는 동일한 입력에 대해서 다른 결과를 반환(Return)하는 경우가 있다.

자동차를 a미터 전진시키고 결과 좌표(x, y)를 반환하는 메소드(Method)[9]가 있다고 가정하자. 이 메소드는 자동차의 현재 위치에 의존하여 동작한다. 즉, 동일한 입력값 10을 넣는다 하더라도 매 호출 시마다 다른 좌표 값이 반환된다. 지극히 당연한 동작으로 여겨지지만 이러한 구조는 테스트 케이스를 작성하기 어렵게 하며, 실제 버그가 발생했을 때 입력 값을 이용하여 출력에 문제가 있는지 여부를 확인하기가 쉽지 않다.

6 Edsger W. Dijkstra(1968), "Go To Statement Considered Harmful," Communications of the ACM, Vol. 11, No. 3, pp. 147-148
7 함수의 확장된 개념으로, 구조체는 일반 함수와 달리 변수(Variable) 할당이 가능하였기 때문에 단순 계산 이상의 목적으로 활용 가능하다.
8 예) 오렌지란 객체가 존재한다면, 오렌지의 색상, 맛 등은 오렌지 객체의 변수로서 저장됨
9 프로그래밍 언어에서 함수와 같은 맥락에서 사용하는 표현으로, 입력 값에 의존해 일련의 과정을 거쳐 결과를 도출해내는 개념

■ **객체 간 의존성 증가** : OOP에서는 객체 간 상호작용을 위해 함수가 다른 객체의 함수를 호출한다. 이 경우 함수는 외부 세계에 의존적으로 동작하기 때문에 재사용성(Reusability)이 떨어지며 프로그램의 복잡도가 증가하는 주요 원인이 된다.

예를 들어, 데이터를 하나 입력받고 이를 외부에 존재하는 저장소에 추가하는 함수를 가정해보자. 이 함수는 외부 객체에 의존성(Dependency)을 갖게 되므로 구조적 복잡성을 야기한다. 의존성을 줄이려는 다양한 노력이 시도되고 있지만 여전히 현재의 OOP에서는 의존성 관리가 어렵다.

■ **객체 내 상태 변화 제어의 어려움** : 객체 내 변수는 객체에서 제공하는 함수에 의해 변경된다. 대부분의 함수는 객체 내 변수들을 외부에서 변경 가능하도록 마련하는 공식적인 통로다. 다수의 외부 경로를 통해 객체 내 함수가 호출되는 경우, 어느 시점에 어떤 외부경로를 통해 상태가 변경되었는지 추적하는 것은 쉬운 일이 아니다.

가령 멀티스레딩 환경에서는 객체 내 함수를 동시에 다수가 접근할 가능성이 있어 버그 발생에 대한 원인 분석에 많은 비용이 소모될 수 있다.

함수형 프로그래밍의 특성과 장단점

상기 언급한 몇 가지 문제점에 대한 대안으로 함수형 프로그래밍(FP)이 등장하였다. FP는 상태를 제어하기보다는 상태를 없애고 목적에 집중하는 프로그래밍 패러다임이다. 객체 단위로 문제를 분해하지 않고 함수들의 세트로 문제를 분해하며, 주어진 입력에 대하여 항상 같은 결과를 출력으로 내보낸다는 특징을 갖는다. 이하에서는 FP의 주요 특징을 살펴보고 어떠한 언어들이 FP를 지원하는지 살펴보기로 한다.

■ 함수형 프로그래밍의 기원과 정의

FP는 데이터 처리를 수학적 함수의 계산으로 취급하며 함수들의 집합으로 프로그램을 작성한다. FP의 기원은 1930년대 수학자 알론소 처치가 개발한 람다 대수(Lambda Calculus)에서 찾을 수 있다. 람다 대수는 함수의 정의, 함수 적용, 귀납적 함수 추상화, 수학 연산을 표현하는 형식 체계이다. 한마디로 표현하면 함수의 수학적 연산을 표현하는 방법이며 다음의 예제로 간단히 설명할 수 있다.

표 1 람다 표현식의 간단한 예제

함수 설명	람다 표현
숫자 x를 입력받고, x의 제곱을 반환	x => x * x
숫자 x와 y를 입력받고, 두 수의 합을 반환	(x, y) => x + y

표 2 동일 기능에 대한 FP와 OOP의 차이점

- 함수 square(x)는 파라미터 x에 대해 x*x의 값을 반환
- 함수 sum(x,y)는 파라미터 x,y에 대해 x+y의 값을 반환

함수 설명		표현 예
함수형	(함수 1) 숫자 x를 입력받고, x의 제곱을 반환하는 함수	square(x)
	(함수 2) 숫자 x와 y를 입력받고, 두 수의 합을 반환하는 함수	sum(x,y)
	→ 동일한 입력에 대해 항상 동일한 출력을 보장 (함수를 구성 시 외부 접근을 통해 변경될 소지가 있는 변수를 허용하지 않음)	

객체지향	• 'Calculator'라는 객체의 상태(state)에 의존해 두 가지 함수가 선택적으로 실행
	• 객체 내 change_state 함수를 외부에서 접근하여 state 값이 변경될 시, 동일 파라미터라도 출력 값이 의도와 달라질 소지가 존재
		Calculator(x , y) {
		private state = 'square' or 'sum'
		
		change_state(keyword){
		state = keyword
		}
		
		run(){
		if(state == square)
		return square(x)
		else
		return sum(x, y)
		}
		}

FP는 위와 같이 람다 대수 이론을 프로그래밍에 도입한다. 따라서 (1) 동일한 입력에 대해서 항상 동일한 출력이 나오는 것을 보장하고, (2) 코드 복잡도를 낮추며, (3) 멀티스레딩에 대한 안정성을 보장하는 등 상기 소개한 OOP의 문제를 해결했다는 점에서 주목할 만하다.

■ 함수형 프로그래밍의 구현

FP의 철학은 선언형(Declarative) 프로그래밍 기법으로 구현 된다. 선언형 프로그래밍은 "무엇"을 할 것인지를 코드로 작성하는 프로그래밍 기법으로, "어떻게" 작동할 것인지 표현하는데 초점을 맞춘 명령형(Imperative) 프로그래밍 기법과 궤를 달리한다. 명령형 프로그래밍은 C++, Java와 같은 대부분의 OOP를 지원하는 언어가 채택한 개발 형태로, 수행해야 하는 단계를 알고리즘으로 자세히 표현한다. 반면 선언형 프로그래밍에서는 특정 선언으로만 프로그램을 동작시킨다. 가령 HTML 또는 SQL처럼 어떤 데이터를 가져올 것인지 명확히 표현하는 방식이 이에 해당한다.

[표 3]은 명령형인 자바와 명령형/선언형을 지원하는 스칼라의 코드 비교이다. 자바 코드와 비교하면 스칼라는 단 3개의 단어를 수행하는 것으로 동일한 작업을 수행한다.

표 3 numbers에 포함된 값을 화면에 출력하는 코드 예제

자바 코드	스칼라 코드

for(int value : numbers) {
System.out.println(value);
}
	numbers.foreach { e => println(e) }
	//또는 다음과 같이 축약 가능
	numbers foreach println

[해설] ① numbers에 저장 된 숫자를 하나 꺼낸다(value : numbers) ② 화면에 숫자를 출력(println(value)) ③ numbers에 저장된 다른 숫자에 대해 ①~② 절차를 반복수행(for문)	[해설] ① numbers에 들어있는 모든 숫자를 (numbers.foreach) ② 화면에 출력(e => println(e))

좌측 코드는 numbers라는 리스트에서 값을 하나씩 꺼내어 화면에 출력하는 행위를 반복적으로 수행하는 절차를 표현하고 있다. 반면 우측 코드의 경우 수행 단계의 표현 없이 수행하고자 하는 목적을 직관적으로 표현하고 있다. FP는 선언형 프로그래밍 기법을 활용하는 대표적 방법론으로서 우측 코드와 동일한 특성을 지닌다.

함수형 프로그래밍의 장점과 한계

FP의 장점은 함수의 수행결과를 입력 값에만 의존하도록 설계하기 때문에 시스템의 불규칙한 상태 변화에 의한 예기치 못한 영향을 최소화할 수 있다는 점이다. 즉 버그, 외부의 시스템 공격 등과 같은 문제가 나타났을 때 보다 손쉽게 원인을 찾아 고칠 수 있음을 의미한다.

또한 OOP의 한계로 지적되었던 멀티스레드 환경에서의 객체 내 상태 변화 제어의 어려움이 근본적으로 발생하지 않는 구조(객체 내 상태변화가 없도록 설계)이므로 유지보수 측면에서 매우 뛰어나다는 이점을 지닌다.

FP가 갖는 한계는 단순함을 강요받는다는 점이다. 객체지향에서의 상태(State)는 상기 언급했던 바와 같이 다양한 부작용을 야기하지만, 외부에 인한 객체 특성의 변화를 코드로 표현하는데 있어 직관적이다. 반면 FP는 입출력의 일관성을 위해 상태를 허용하지 않으므로 같은 기능의 구현을 위해 OOP에서 고려할 필요가 없는 다양한 함수의 조합을 고안해야만 한다.

이는 복잡한 소프트웨어를 개발함에 있어 사람에게 친숙하지 않은 별개의 코드 구조를 고민해 내야만 하는 상황에 직면할 수 있음을 의미한다. FP의 이러한 특성은 기존 개발자가 FP를 수용하는데 큰 장벽이 되고 있다.

함수형 프로그래밍을 지원하는 언어

FP를 지원하는 언어들로는 하스켈[10], 리스프[11]가 대표적이다. [표 4]는 FP를 지원하는 다양한 언어이다.

표 4 주요 함수형 프로그래밍 언어

명칭	특징
Haskell	함수형 프로그래밍 지원
Scala	자바의 변형된 언어로서 자바 API를 지원 객체지향·함수형 프로그래밍 지원
Rust	객체지향·함수형 프로그래밍 지원

10 Haskell. 1987년 "함수형 프로그래밍 언어와 컴퓨터 구조에 관한 총회"에서 위원회가 발족되어, 1990년 첫 버전이 발표됨. https://www.haskell.org
11 Lisp. 1958년 MIT의 존 메카시가 고안한 프로그래밍 언어

명칭	특징
Clojure	리스프 변형 언어로 함수형 프로그래밍 지원 JVM, CLR, 자바스크립트 엔진에서 실행
Common Lisp	리스프 변형 언어로 모든 종류의 방법론 채택 가능 절차적·객체지향·함수형 프로그래밍 지원
Scheme	리스프 변형 언어 절차적·함수형 프로그래밍 지원
OCaml	Caml 프로그래밍 언어에 객체지향 추가 객체지향·함수형 프로그래밍 지원
F#	OCaml 언어를 계승해 동일 방법론 지원 .NET프레임워크, 자바스크립트에서 실행
Erlang	함수형 프로그래밍 지원

전문적인 FP 언어가 아니라도 FP를 접할 수 있다. 기존의 OOP 기반 언어들도 FP가 갖는 장점을 인식하고 점진적으로 FP 기법을 늘려가는 추세이다. 자바(JAVA) 8버전은 람다 함수를 지원하며, 자바스크립트는 익명 함수를 지원, 파이썬(Python)도 람다, 고차 함수 등을 지원하는 등 최근 들어 FP의 장점을 도입하고 있는 추세이다.

시사점

함수형 프로그래밍은 객체지향 프로그램이 갖는 패러다임을 벗어나 새로운 사고방식, 즉 함수단위의 조합으로 프로그램을 구성하는 방법이다. 객체의 상태로부터 자유로워지고, 코드가 줄어들며, 복잡한 의존성을 없애는 장점이 있는 반면, 수학적 표현식의 이해와 새로운 문법에 대한 적응이 필요하기도 하다. 함수형 프로그래밍을 지원하는 다양한 언어가 아직까지 주류를 이루지는 못하고 있으나, 기존의 언어들도 함수형 프로그래밍 패러다임을 도입하고 있는 등 향후 발전 가능성이 높다. 이처럼 함수형 프로그래밍에 대한 관심이 높아지는 추세를 볼 때 관련 역량이 국내 SW인력에게 있어 중요해질 것이라 짐작할 수 있다.

이와 관련한 국내의 교육은 흥미롭게도 블록체인을 매개로 확산되고 있는데, 3세대 블록체인 플랫폼으로 꼽히는 테조스(Tezos)[12]의 기반 언어가 함수형 프로그래밍을 채택하고 있기 때문이다. 이와 별개로 상기 언급한 비주류 언어의 교수 과정에서 관련 개념이 일부 소개되기는 하나, SW 개발방법론의 기초가 형성되는 대학에서의 정규 커리큘럼은 찾아보기 힘들다.

함수형 프로그래밍이 조명받는 현 시점에서, 획일화된 SW개발 방법론 교육에 다양성을 부여할 수 있도록 학계 및 업계의 전향적인 관심이 필요한 시점이다.

12 기존 블록체인 플랫폼의 문제로 떠오른 하드포크(hard fork)의 잠재적 위협을 해결한 스마트 컨트랙트(Smart Contract) 플랫폼이다. 프로그래밍 언어로 미켈슨(Michelson)과 오캐멀(Ocaml)을 채택하고 있는데 두 언어는 함수형 프로그래밍을 지원한다.