프로그래밍 의미론

내용

[개요 1](#_Toc487535476)

[상태와 함수 1](#_Toc487535477)

# 개요

상태 변경에 의존하거나 함수형 언어이거나 서술된 코드에서 의미를 파악하는 것이 핵심이다. Imperative / functional 각각 다른 세부 서술 구조를 갖고 있지만 변형된 상태 값들이 결국 중요하다.

사실 관계의 서술이나 상태 값의 서술 모두 그 크기가 방대해진다는 점이 의미론에서 어려운 점이다. 방대한 의미 덩어리를 추상화를 통해 이해 가능하게 만들어야 한다.

추상화의 도구는 다음과 같다.

* 레이어
* 인터페이스
* 자료구조 (타잎)
* 함수

위에 더하여 추상화를 포함하고 있는 각 영역별 언어를 갖추면 매우 편리하다. 또한 데이터를 정의하고 코드 자체도 데이터를 보는 기능이 있으면 좋다.

* DSL
* 인터프리터 (데이터와 통합된)

# 상태와 함수

하나의 함수 호출 그래프로 구성되는 함수형 언어와 달리 상태를 갖고 상태를 변경하여 처리하는 대부분의 언어에서 함수 호출과 상태 간의 관계가 의미 파악을 어렵게 만든다.

사전 조건, 사후 조건으로 함수의 동작을 정의할 수 있다. 이 조건이 복잡할수록 복잡한 함수이다.

**개념화를 통해 조건을 추상화하는 것이 서술의 핵심이다.**

개념화는 무엇을 의미하는가? 지금까지 타잎을 만들고 이들 타잎을 사용해서 전체를 구성하는 것에 관심을 가졌다면 타잎이 갖고 있는 세부 개념들에 주목해야 한다.

서술에서 함수의 기능을 분리하는 것이 중요하다. 제어 함수, 조건 함수, 변경 함수로 나눌 수 있고 더 세부 분류가 가능해 보인다.

하나의 동작으로 추상화하고 서술하는 것이 더 낫다면 기능 함수 하나로 정의할 수도 있다. Sort()와 같이 명확한 개념이 있는 경우 그렇다. 각 영역에서 이와 같이 명확한 개념들을 만들고 이를 사용해서 서술하는 것이 중요해 보인다.

Sort() 함수의 사전 조건은 각 배열에 자료가 있다는 것이고 Sort() 함수의 사후 조건은 졍렬 되어 있다는 것이다. 조건들의 확인과 서술을 프로그래밍의 일부로 보는 것이 중요하다.

함수:

* Precondition
* Post condition

또는 다음과 같이 서술할 수도 있다.

함수:

* when starts with A, then ends with B
* when starts with C, then ends with D and E or F

이를 사람이 이해할 수 있는 조건과 결과로 개념화 할 수 있다.

Assert()가 서술의 중요한 한 방법이 되고 필요하다고 생각되는 이상으로 사용해야 한다. 그렇게 훈련 되어야 하고 논리적으로 사용해야 한다.

# 타잎과 인터페이스

타잎 중 사용을 전제로 한 것을 인터페이스라고 볼 수 있다. 인터페이스를 구성하는 타잎들은 외부에서 사용을 전제로 한다. 이와 대비하여 구현을 위한 타잎들이 있는데 정확성을 보장하면 좀 더 자유롭게 사용할 수 있다.

## 내부 개념화

타잎 내에 함수 그룹이나 변수 그룹에 대해 개념화가 필요한 경우가 있다. 이런 경우 별도 타잎으로 만들 정도로 크지 않다면 그룹화를 통해 개념화 할 수 있다. 이의 서술은 #region으로 분류해서 지정할 수 있다.

# 관점들

## 그린스펀의 10번째 규칙

“복잡한 C/Fortran/java 프로그램은 반쯤 완성된 Common Lisp를 포함하고 있다”

두 가지를 반영한다. 하나는 Domain Specific Language (DSL)을 갖추면 추상화가 비약적으로 올라간다는 점이다. 다른 하나는 Lisp는 코드 자체를 데이터로 보고 데이터도 코드로 표현이 쉽다는 점이다. 괄호로 연결된 코드를 보면 Json과 유사해 보인다.

Clojure를 배우고 사용을 시도해 볼 필요가 있다.