패턴을 찾아가는 접근 방식으로 어떤 방식이 좋을까?

패턴이라고 해서 거창한 이야기를 하려는 것은 아니다. 지난 번 스터디에서 나왔던 이야기를 같이 공유하고 싶은 마음에 글을 적는다. 패턴과 관련한 논의를 시작하게 된 계기는 clojure 스터디에서 배열에 담겨 있는 여러 개의 원소 중 홀수 번째 원소를 찾는 문제를 해결하면서 시작하게 되었다. 여러 원소 중 홀수 번째 원소만을 찾아 반환하는 함수를 구현하는 과정은 odds 구현과정(작성중) 에 잘 정리되어 있다.

odds 구현과정(작성중) 문서를 보면 원소가 없는 상태에서부터 시작해 원소의 숫자를 하나씩 증가시켜가면서 패턴을 찾아가는 과정을 보여주고 있다. 접근 방식으로 본다면 bottom up 방식으로 문제를 해결하고 있다. 이 과정을 공유하면서 일정한 패턴을 찾아가는 접근 방식에 대한 논의를 시작하게 되었다.

패턴을 찾기 위한 마지막 구현 과정은 다음과 같다.

```
(defn odds [coll]
     (cond
       (empty? coll) nil
       (empty? (rest coll)) (cons (first coll) (odds (rest (rest
coll))))
       (empty? (rest (rest coll))) (cons (first coll) (odds (rest (rest
coll))))
       (empty? (rest (rest (rest coll)))) (cons (first coll) (odds (rest
(rest coll))))
       (empty? (rest (rest (rest (rest coll))))) (cons (first coll)
(odds (rest (rest coll))) )))

(t (odds ()) nil)
(t (odds nil) nil)
(t (odds '(1)) '(1))
(t (odds '(1 2)) '(1))
(t (odds '(1 2 3)) '(1 3))
(t (odds '(1 2 3 4)) '(1 3))
```

소스 코드를 이해하지 못하더라도 일정한 패턴이 보이는 것은 확인할 수 있다. 그렇다면 우리가 일정한 패턴을 찾기 위한 접근은 어떻게 하는 것일까? 일단 스터디에서는 다음과 같이 두 가지 방식으로 분류했다.

top down 방식

top down 방식은 구현을 시작하기 전에 일정한 패턴을 먼저 찾는 과정을 거치는 것이다. 먼저 일정한 패턴을 짧막한 글로 적거나 pseudo-code로 구현하는 과정을 거친다. 이 과정에서 검증까지 마친 후 패턴에 대한 확신이 드는 경우 구현을 시작하는 것이다.

김창준씨가 쓴 어떻게 공부할까? 프로그래머를 위한 공부론에서 자료 구조와 알고리즘을 공부하는 방법에 대한 부분을 보아도 비슷한 내용이 있다.

> 제가 생각건대, 교육적인 목적에서는 자료구조나 알고리즘을 처음 공부할 때 우선은 특정 언어로 구현된 것을 보지 않는 것이 좋을 때가 많습니다. 대신 말로 된 설명이나 의사코드(pseudo-code) 등으로 그 개념까지만 이해하는 것이죠. 그 아이디어를 절차형(C, 어셈블리어)이나 함수형(LISP, Scheme, Haskell), 객체지향(자바, 스몰토크) 언어 등으로 직접 구현해 보는 겁니다. 그 다음에는 다른 사람이나 다른 책의 코드와 비교합니다. 이 경험을 애초에 박탈당한 사람은 귀중한 배움과 깨달음의 기회를 잃은 셈입니다.
>
> 만약 여러 사람이 함께 공부한다면 각자 동일한 아이디어를 같은 언어로 혹은 다른 언어로 어떻게 다르게 표현했는지를 서로 비교해 보면 배우는 것이 무척 많습니다.

bottom up 방식

bottom up은 앞에서 살펴본 odds 구현과정(작성중)와 같은 접근 방식이라고 생각하면 된다.

위 두 가지 접근 방식에서 어느 방식으로 접근하는 것이 프로그래밍을 처음 시작하는 프로그래머에게 적합할까? 모든 사람에게 위 두 가지 접근 방식 중 한 가지 접근 방식이 적합할까?

스터디에서 '이 방식이 정답이다.'라고 결론을 내리지 않았다. 하지만 스터디에서 나온 이야기를 공유해 보면 다음과 같다.

top down 방식처럼 구현 없이 패턴을 찾아가는 것은 정말 힘든 일이다. 이 패턴을 잘 찾는 사람들이 있는데 이런 사람들은 천재적으로 이런 능력이 뛰어나거나 이 방면으로 꾸준히 연습한 사람에게나 가능한 접근 방식이지 않을까? 우리 같은 평범한 사람들은 bottom up 방식으로 접근하는 것이 성능 가능성이 더 높은 것은 아닐까?

특히 우리나라의 수학 교육과 같이 정답을 맞추는 것에 익숙해져 있는 사람들에게는 처음부터 top down 방식으로 접근하도록 하는 것에는 한계가 있지 않을까? 예를 들어 우리는 수학 교육을 받을 쌔 4 * 5는 20이라고 외웠다. 곱셈은 반복적인 덧셈을 패턴화해서 만들어진 추상화된 형식이라는 형태로 배워본 경험이 없다. 이 과정을 연습해 본 경험이 없다. 이런 상태에서 프로그래밍을 처음 시작하는 친구들에게 패턴을 찾고, 이에 대한 추상화를 요구하는 것에는 한계가 있는 것이 아닐까?라는 의구심이 생겼다.

먼저 수학 시간에 경험하지 못했던 패턴을 찾아가는 과정에 대한 연습을 하고, 이 과정을 추상화해 하나의 수식 또는 함수로 표현해 가는 과정에 대한 연습이 선행되어야 할 듯하다. 이 연습도 한, 두번으로 해결되는 것이 아니라 일정 수준으로 반본적인 연습이 필요할 것이다. 이와 같은 연습이 된 상태에서 다음 단계의 패턴화가 가능하고, 이런 연습이 반복될 때 top down 방식으로 사고하는 것도 가능한 것이 아닌가?

이와 같이 패턴을 찾아가는 연습이 중요한 것이 이 과정 자체가 우리가 흔히 이야기하는 설계의 과정이고, 추상화 과정이라고 생각한다.몇년 동안 프로그래밍을 한 친구들도 설계와 추상화는 정말 힘들어 한다. 프로그래밍을 하면서 이와 같이 패턴을 찾는 연습, 추상화하는 연습을 단계적으로 하지 않았기 때문에 설계와 추상화 과정이 힘든 것은 아닐까? 이 연습을 힘들어하는 이유가 수학 시간에 연습이 되어 있지 않은 bottom up의 경험이 없기 때문이 아닐까? 고등학교까지의 수학 시간에 bottom up에 대한 연습이 되어 있지 않다면 프로그래밍을 처음 시작하는 친구들에게 먼저 이에 대한 연습을 시키는 것이 이에 대한 어려움과 거부감을 덜어줄 수 있는 것은 아닐까?

나 또한 지금까지 선배 프로그래머에게 설계와 관련해 top down에 대한 이야기를 많이 들었다. 물론 초반 설계와 추상화가 중요하다는 것을 안다. 하지만 작은 문제를 가지고 설계를 해본 경험이 많지 않은 상태에서는 너무 힘든 일이었다. 그러다보니 중간에 어정쩡한 상태로 개발을 하는 경우가 많다. 예를 들어 이런 경우이다. 앞 단계에 설계가 필요하다고 하니 일정 시간 설계에 투자한다. 설계 시간을 가지지만 정확한 모습이 그려지지 않은 상태에서 구현 단계로 넘어간다. 위 접근 방식에서 top down과 bottom up이 뒤섞여 있는 어정쩡한 상태가 된다. 물론 점진적으로 학습이 이루어지면서 설계를 개선해 나간다는 관점에서는 이런 접근 방식이 맞을 수도 있다. 하지만 이런 방식으로 연습해서는 top down과 bottom up의 제대로 된 맛을 느끼지 못할 듯하다. 프로그래밍을 처음 배워나가는 단계에서는 top down이든, bottom up이든 제대로 된 경험을 해보는 것이 중요하다고 생각한다.

나 또한 이에 대한 정답을 모른다. 이에 대한 접근 방식이 자료구조와 알고리즘을 학습할 때와 일반적으로 소프트웨어를 개발할 때의 설계와는 다를 수도 있다는 생각이 든다. 이 글을 쓰는 이유는 이와 관련해 같이 한번 고민해 보고 이야기해볼 수 있는 시간을 가졌으면 하는 바람 때문이다.