

# 프로그래밍 언어 개론

## 과제 보고서

[PL00] HW1\_201601980\_KimSeongHan

개발환경 : Ubuntu

제출일 (2021-03-08)

201601980 / 김성한(00분반)

# <과제 설명>

## Ocaml : HW1

### ## HW1

Write a function `sum` that, given two integer bounds `n` and `m` and a function `f`, computes a summation.

$$\text{sum } n \ m \ f = \sum_{i=n}^m f(i).$$

[문제 HW1]

● 해당 과제는 문제에 제시된 `sum` 함수를 구현하는 문제로 세 개의 인자 `n`, `m`, `f(함수)`를 받아 어떤 함수가 들어오던지 해당 함수(`f`)의 `n~m(n≤m)` 범위에서의 `f(n) ~ f(m)`을 모두 더하는 함수(`sum`)를 Ocaml로 구현하는 과제이다.

● Ocaml언어에서는 대부분의 코드에 반복문을 사용하지 않고 재귀적인 방법으로 반복을 실행한다고 공부하였다. 이에 따라 해당 함수의 구현에 있어서 재귀를 사용해야 한다는 점을 염두하고, 과제를 시작하였다.

## <코드 구현 및 실행 결과 화면>

```
k0s0a7@DESKTOP-MLPLCFD: ~$  
let f x = x * 3 in  
let rec sum n m f =  
    if n = m then (f m)  
    else (f n) + (sum (n+1) m f)  
in  
Format.printf "The result is : %d \n" (sum 1 100 f)
```

[코드를 구현한 후 출력을 위해 임의의 함수 f 및 출력문을 작성한 코드이다.]

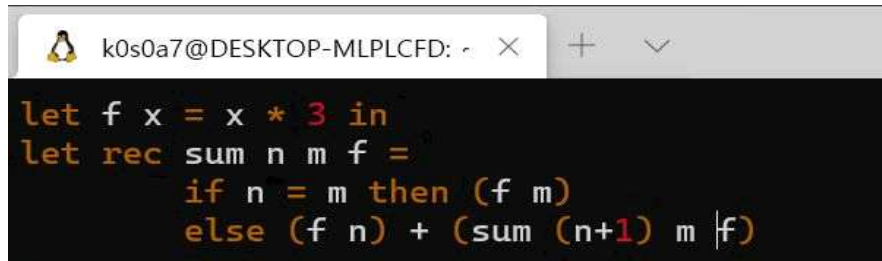
```
k0s0a7@DESKTOP-MLPLCFD: ~$  
k0s0a7@DESKTOP-MLPLCFD: ~/sum$ dune exec ./sum.exe  
The result is : 15150
```

[해당 코드에 대한 결과이다.]

● 초기 sum 함수를 구현함에 있어 재귀를 사용해야 한다는 점을 인지하고 let rec을 이용해 sum함수를 정의하였다. 이때 인자는 3개가 필요하여 n m f를 작성해주었다. 이 함수는 재귀적으로 반복되며 n부터 시작하여 n이 1씩 증가하면서 n이 m과 같은 경우 f(m)을 리턴하고 결과적으로  $\sum_{i=n}^m f(i)$ 의 값을 리턴하는 함수가 된다. 그 후 다음 expression을 위해 in을 선언한 뒤 Format.printf 문으로 결과값을 출력하였다.

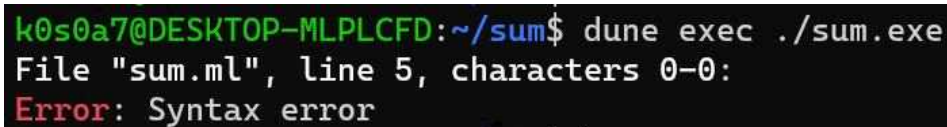
● 두 번째 사진은 sum함수에 대해 1부터 100까지  $f(x) = x*3$ ; 함수로 값을 구한 결과값을 보인다.

## <과제 구현 중 어려웠던 부분 및 해결 과정>



```
k0s0a7@DESKTOP-MLPLCFD: ~  
let f x = x * 3 in  
let rec sum n m f =  
    if n = m then (f m)  
    else (f n) + (sum (n+1) m f)
```

[1. sum 함수를 정의하고 출력을 위한 예시로 작성한 f 함수를 정의만 한 경우]

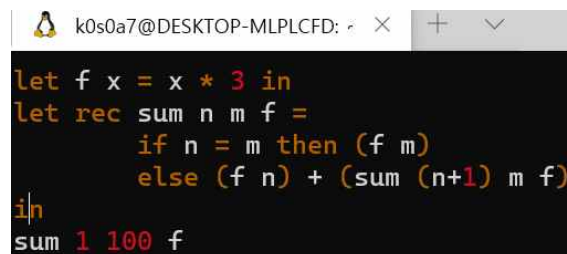


```
k0s0a7@DESKTOP-MLPLCFD:~/sum$ dune exec ./sum.exe  
File "sum.ml", line 5, characters 0-0:  
Error: Syntax error
```

[1번 사진에 대한 오류]

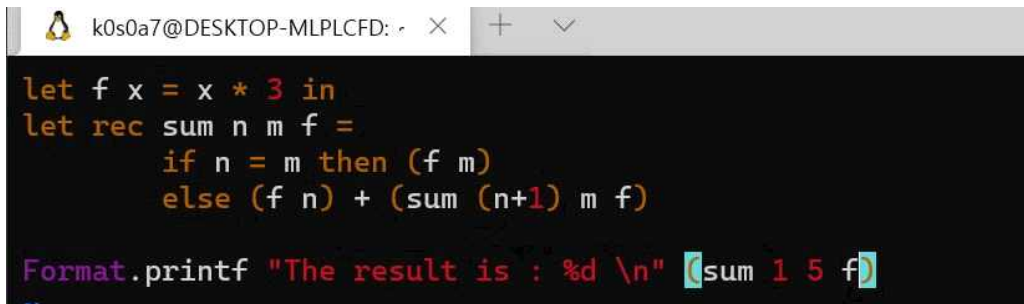
● 1번사진(코드)을 확인해보면 초기 let-in으로 함수 f를 정의하고 let 구문으로 sum 함수를 선언하였다. let은 먼저 어떠한 함수 또는 변수를 선언하는 구문이고, let-in은 어떠한 함수 또는 변수선언을 이용하여 계산에 이용하는 구문이다. 즉 let-in은 선언한 이후 다른 함수 혹은 변수에서 사용하여야 할 값이 있어야 하며 위의 경우 let-in으로 선언한 함수 f의 값이 정의되지 않아 어떠한 값도 도출되지 않기 때문에 타입을 정의할 수 없게 되고, 이에 따라 구문 오류(Syntax error)가 발생하게 되었다.

● 해당 부분을 해결하기 위해서는 f의 함수값을 지정해줘야하고 이에대한 해결코드는 다음과 같다.



```
k0s0a7@DESKTOP-MLPLCFD: ~  
let f x = x * 3 in  
let rec sum n m f =  
    if n = m then (f m)  
    else (f n) + (sum (n+1) m f)  
in  
sum 1 100 f
```

[1번 사진에 대해 찾은 해결코드(f의 값을 정의해주기 위해 sum함수를 실행)]



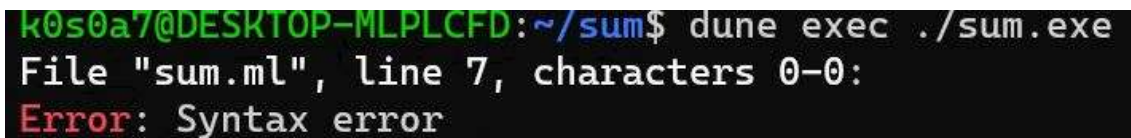
```

k0s0a7@DESKTOP-MLPLCFD: ~
let f x = x * 3 in
let rec sum n m f =
  if n = m then (f m)
  else (f n) + (sum (n+1) m f)

Format.printf "The result is : %d \n" (sum 1 5 f)

```

[2. sum함수를 전역 함수로 선언하기 위해 in을 제거한 모습]



```

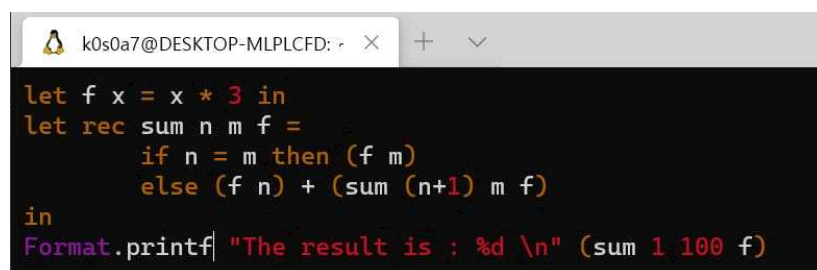
k0s0a7@DESKTOP-MLPLCFD: ~/sum$ dune exec ./sum.exe
File "sum.ml", line 7, characters 0-0:
Error: Syntax error

```

[2번 사진에 대한 오류]

● 2번 사진에서는 과제제출을 위해 in을 제거하라는 말을 듣고 무작정 in을 제거했다가 난관에 봉착했을 당시의 캡처본이다. 이때 처음에는 1번 사진에서의 오류와 비슷한 오류가 나오길래 같은 이유일거라는 생각을 했지만 질문에 대한 답변을 보고 나서 내가 작성한 2번 사진의 코드는 현재 expression이 하나가 아니기 때문이라는 것을 알게 되었다. 먼저 Ocaml에서는 function body 및 main file은 하나의 구문으로만 구성되어야 하는데 위의 코드를 보면 알수 있듯이 Format.printf 문은 현재 let in 구문의 sub expression이 아니기 때문에 별도의 구문으로 인식이 되고 따라서 구문오류(Syntax error)가 발생하게 된 것이다.

● 2번사진의 오류에 대한 해결책으로 in을 이용하여 하나의 expression으로 만들어 줘야하며, 이를 적용하여 작성한 코드는 아래와 같다.




```

k0s0a7@DESKTOP-MLPLCFD: ~
let f x = x * 3 in
let rec sum n m f =
  if n = m then (f m)
  else (f n) + (sum (n+1) m f)
in
Format.printf "The result is : %d \n" (sum 1 100 f)

```

[2번 사진에 대하여 작성한 해결코드(expression을 하나로 만들기 위해 in을 추가)]

## <최종 제출 코드 및 결과 화면>



```
k0s0a7@DESKTOP-MLPLCFD: ~  
let rec sum n m f =  
    if n = m then (f m)  
    else (f n) + (sum (n+1) m f)
```

[과제 제출을 위해 다음은 최종 코드 (sum.ml)]



```
k0s0a7@DESKTOP-MLPLCFD:~/sum$ dune exec ./sum.exe  
k0s0a7@DESKTOP-MLPLCFD:~/sum$
```

[최종 코드(sum.ml) 실행 화면(출력문을 작성하지 않아 별도의 출력은 없다)]

● 최종 코드에서는 기존에 출력을 위한 예시로 정의했던 함수 f와 출력문(Format.printf)을 제거하고 전역함수로서의 기능을 위해 let rec sum에 대응되는 in을 제거해 주었다. 또한 해당 sum.ml 파일 실행 결과 출력문이 없어 아무것도 표시되지 않지만 오류가 생성되지 않았음을 알 수 있었다.

## <과제를 하며 어려웠던 부분 및 새로 고찰한 내용>

● 이번 과제를 하면서 가장 어려웠던 부분은 역시 `let` 과 `let-in`의 차이, 그리고 `expression`을 하나로 합치는 내용이 이번 과제를 하면서 스스로 이해하고자 했던 목표였다. 사실 과제를 진행하기 전 해당 부분에 대한 이해가 잘 되지않아 강의를 두 번 정도 돌려봤는데, Ocaml이란 언어를 처음 접하다보니 막상 이해가 쉽게 되지는 않았다. 하지만 과제를 진행하면서 모르는 부분에 대해 바로 질문하며 피드백을 받으니 이제 어느 정도 틀이 잡히면서 강의를 한번 더 들었을 때 이해되는 부분이 상당했다. 알게된 내용을 정리해 보면

1. `let` → 어떠한 함수 혹은 변수를 선언.
2. `let-in` → 어떠한 함수 혹은 변수를 선언하고 이를 이용하여 그 후에 이용하여 계산할 수 있도록 하는 구문
3. `in`을 이용하여 `expression`들을 접합시켜 하나의 `expression`으로 만들어줄 수 있음.
4. 그 외 오류를 통해 얻은 지식들 ex) 사용하지 않은 변수가 있으면 안됨. 등등

몰랐지만 새롭게 알게된 내용이 위와 같이 나온다.

아직 새로운 언어를 배움에 있어서 시작점이라 많이 낯설지만 이해할수록 코드도 잘보이고 원하는 결과를 내면서 보람을 느낀 것 같다. 어렵지만 열심히 공부해서 흥미를 키우고 많은 내용을 배우고 싶다.