

Homework 4
SNU 4190.310, 2026 봄
이 광근
Due: 4/18(토) 24:00

이번 숙제의 목적은:

- 수업시간에 살펴본, 상식적인 명령형 언어의 정확한 정의를 이해하고 그 실행기를 구현해 보기.
- 그 언어로 프로그램 해보면서 아쉬운 점에 눈뜨기.
- 앞으로 프로그래밍 언어 구현에서 넘어야 할 산이, 상식만 가지고는 넘기 어렵다는 것을 겪어보기. 마지막 문제입니다.

Exercise 1 (30pts) “K- 실행기”

수업시간에 정의해간 명령형 언어 K-를 생각하자. 그 문법과 의미의 정확한 정의는 강의페이지 [숙제]란에 제공된다. 이번 숙제는 K- 프로그램을 의미정의대로 실행시키는 함수(interpreter)를 작성하는 것이다.

아래의 KMINUS 꼴을 가지는 모듈 K를 정의하라.

```
module type KMINUS =  
sig  
  exception Error of string  
  type id = string  
  type exp = NUM of int | TRUE | FALSE  
            | VAR of id  
            | ADD of exp * exp  
            | SUB of exp * exp
```

```

| MUL of exp * exp
| DIV of exp * exp
| EQUAL of exp * exp
| LESS of exp * exp
| NOT of exp
| SEQ of exp * exp (* sequence *)
| IF of exp * exp * exp (* if-then-else *)
| WHILE of exp * exp (* while loop *)
| LETV of id * exp * exp (* variable binding *)
| ASSIGN of id * exp (* assign to variable *)
| READ of id
| WRITE of exp
| LETF of id * id list * exp * exp (* NEW: procedure binding *)
| CALLV of id * exp list (* NEW: call by value *)
| CALLR of id * exp list (* NEW: call by reference *)
| RECORD of (id * exp) list (* NEW: record construction *)
| FIELD of exp * id (* NEW: access record field *)
| ASSIGNF of exp * id * exp (* NEW: assign to record field *)

type program = exp
type memory
type env
type value
val emptyMemory: memory
val emptyEnv: env
val run: memory * env * program -> value
end

```

K- 프로그램이 어떻게 `exp`들로 표현될지는 쉽게 추측할 수 있을 것이다. `exp`으로 표현된 K- 프로그램이 S 라고 하면,

$$K.run (K.emptyMemory, K.emptyEnv, S)$$

는 프로그램 S 를 실행시키게 되는데, 성공적으로 끝나면 최후의 값을 내어주게 된다. 이때 프로그램은 실행중에 I/O를 하면서 프로그램이 하는 일을 바깥세상에 드러내게 된다. 실행중에 타입이 맞지 않는 프로그램이면 `Error`라는 예외상황을 발생시키고 프로그램 실행이 중단되어야 한다. “Error”란 (\iff) 정의된 의미 규칙

으로는 그 프로그램의 의미가 정의될 수 없는 경우다. 입출력은 정수만 가능하다. 출력은 정수를 화면에 뿌리고 다음 줄로 넘어간다. □

Exercise 2 (10pts) “K- 프로그래밍: 구조물 데이터”

다음은 K- 로 작성하고, 위에서 구현한 실행기 `K.run`로 실행시켜 제대로 실행되는 지를 확인한다.

두갈래 나무구조(binary tree)를 만들고 쓸 수 있는 아래의 함수들을 정의하라:

```
leaf: int → tree (* leaf tree *)
makeLtree: int × tree → tree (* tree with only left subtree *)
makeRtree: int × tree → tree (* tree with only right subtree *)
makeTree: int × tree × tree → tree (* tree with both subtrees *)

isLeaf: tree → bool (* see if leaf tree *)
isLtree: tree → bool (* see if tree with only left subtree *)
isRtree: tree → bool (* see if tree with only right subtree *)
isLRtree: tree → bool (* see if tree with both subtrees *)
rTree: tree → tree (* right subtree *)
lTree: tree → tree (* left subtree *)
nodeValue: tree → int (* node value *)
dft: tree → - (* print node values in depth-first order *)
bft: tree → - (* print node values in breath-first order *)
```

위의 함수들 만을 이용해서 나무 구조를 만들고 `dft`와 `bft`를 돌려서 제대로 된 순서로 출력되는 지를 확인하라. □

Exercise 3 (30pts) “뉴 보물섬”

탐사해야 할 지역의 지도를 보고 탐사를 성공리에 마치기 위해 필요한 최소의 준비물을 알아내는 프로그램을 작성해 보자.

탐사는 지도에 나타난 길을 따라 이동하면서 길에 놓인 보물상자를 열고 보물을 모아가는 것이고, 모든 보물이 모아지면 그 탐사는 성공한 것이다. 준비물은 모든 보물상자를 열 수 있는 열쇠들이다.

보물상자와 열쇠:

- 보물상자에는 고유의 알파벳 이름이 표시되어 있다.
- 이름없이 “*”라고 찍혀있는 보물상자도 있다.
- 같은 이름의 보물 상자는 같은 열쇠로 열린다.
- 열쇠는 반복해서 사용할 수 있다.

- 열쇠 종류는 금열쇠와 은열쇠가 있다.
- 열쇠 모양은 외갈래 혹은 두갈래로 갈라진 가지구조(complete binary tree)이다.
- 금열쇠는 필요하면 몇 꺼풀이건 펼칠 수 있다. 한 꺼풀 펼치기란, 금열쇠의 모든 가지들을 두 갈래로 펼치는 것이다.
- 준비물 크기는 열쇠들 가지 갯수로 결정하는데, “올림픽 메달순위”로 한다: 은열쇠 가지 갯수는 아무리 많아도 금열쇠 가지 하나보다 적다.

보물상자와 열쇠를 OCaml 타입으로 정의하면,

```
type treasure = StarBox | NameBox of string
type key = Gold of t | Silver of t
and t = Bar | Node of t * t
```

탐사지도:

- 시작 지점은 하나이다.
- 길들은 모두 외길이거나 두 갈래로 나뉘어 진다.
- 보물상자들은 모두 막다른 골목의 끝에 있다.
- 갔던 길을 되돌아 오지 않고 왔던 곳으로 다시 오는 방법은 없다(tree).
- 길목에 세워진 안내판에는 앞으로 만날 보물상자의 알파벳 이름이 쓰여져 있다.
- 모든 안내판의 이름은 모두 다르다.

탐사지도를 OCaml 타입으로 정의하면,

```
type map = End of treasure
         | Branch of map * map
         | Guide of string * map
```

보물상자마다 필요한 열쇠는 보물상자의 위치가 전체 탐사지도에서 어디냐에 따라 결정되는데, 지도에서 각 지역이 암시하는 열쇠는 다음의 조건으로 결정된다:

현재위치(지도) e	위치의 뜻	열쇠모양의 조건
*	* 보물상자	- (은열쇠 Bar)
x	x 라는 이름의 보물상자	현재 위치에서 x 를 열어줄 열쇠 모양
$\boxed{x}e_1$	안내판 \boxed{x} 이 앞에있는 지도 e_1	e_1 안에서 만날 보물상자 x 의 열쇠가 α 이고 e_1 의 시작점이 암시하는 열쇠모양을 β 라고 하면, 현재 위치가 암시하는 열쇠모양은 (α, β) (왼쪽가지 α , 오른쪽가지 β).
$e_1 e_2$	e_1 과 e_2 로 갈라지는 갈림길	e_1 의 시작점이 암시하는 열쇠모양은 (α, β) 이어야 하고 e_2 의 시작점이 암시하는 열쇠모양은 α 이어야 한다. 이때, 현재 위치가 암시하는 열쇠모양은 β .

예를들어, 각 지도를 성공적으로 탐험할 최소의 열쇠꾸러미는 다음과 같다:

- : 은열쇠 Bar

- : 금열쇠 Bar

1. 지도 x 에는 $\{-\}$.
2. 지도 $\boxed{x}x$ 에는 $\{-\}$.
3. 지도 $(\boxed{x}x)|*$ 에는 $\{-\}$.
4. 지도 $(\boxed{x}(x|x))|*$ 를 성공적으로 탐험하는 것은 불가능.
5. 지도 $(\boxed{x}x)|(\boxed{y}y)|*$ 에는 $\{-\}$.
6. 지도 $(\boxed{x}x)|(\boxed{y}y)$ 에는 $\{-, (-, -)\}$.
7. 지도 $x|*$ 에는 $\{-, (-, -)\}$.
8. 지도 $(\boxed{x}(x|x))|(\boxed{y}y)$ 에는 $\{-\}$. (은열쇠로는 불가능)
9. 지도 $(\boxed{x}((x|x)(x|x))|(\boxed{y}y)$ 에는 $\{-\}$. (은열쇠로는 불가능)

다음의 타입에 맞도록, 위와같은 일을 하는 `getReady`함수

`getReady: map → key list`

를 정의하라.

□