# Homework 5
# SNU 4910.210 Fall 2013

Kwangkeun Yi

**due: 10/26(Sat) 24:00**

이번 숙제의 목적은

- 데이터의 속 구현을 여러가지 방식으로 프로그램 해 보기.
- 속 구현이 여럿인 경우도 인터페이스만 알고 프로그램하는 것을 익히기.
- 올바른 프로그램인지 확신하기가 쉽지 않은 문제를 겪어보기. (여러분이 짜는 프로그램이 항상 올바른 답을 낸다는 것을 확신할 수 있기를 바랍니다.)
- 알아야 할 이론공부들이 많겠다는 동기를 가지게 하기.

**Exercise 1** "종이 벽지 디자인"

벽지 무늬의 전체구조는 대게가 같은 무늬들의 반복이다. 기본 무늬는 검거나 흰 정사각형이다. 무늬를 디자인하는 작업은 기본 정사각형들 4개를 연결해서 4배 큰 정사각형 무늬를 만들고, 이것들 4개를 다시 연결해서 4배 더 큰 정사각형을 만들고, 등등. 되었다 싶으면 디자인된 정 사각형들을 반복해서 종이에 짜넣는 방법을 취한다.

이러한 무늬 데이타의 속 구현을 감추고 다음의 것들만 드러나도록 기획하였다. 각각을 구현하라.

$$\texttt{black} : \mathit{form}$$
$$\texttt{white} : \mathit{form}$$
$$\texttt{glue} : \mathit{form} * \mathit{form} * \mathit{form} * \mathit{form} \to \mathit{form}$$
$$\texttt{rotate} : \mathit{form} \to \mathit{form}$$
$$\texttt{neighbor} : \mathit{location} * \mathit{form} \to \mathit{int}$$
$$\texttt{pprint} : \mathit{form} \to \mathit{void}$$

각각의 정의는 다음과 같다:

- `black`: 기본크기의 검은 정사각형 무늬.
- `white`: 기본크기의 흰 정사각형 무늬.
- `glue`: 같은 크키의 정사각형 무늬 4개를 NW, NE, SE, SW방향의 순서로 받아서 그 위치에 놓고 연결한 4배 크기의 정사각형 무늬를 만든다.
- `rotate`: 정사각형 무늬를 받아서 90도 시계방향으로 돌려진 무늬를 만든다.
- `neighbor`: 주어진 위치의 기본 정사각형의 주변에 있는 최대 8개의 정사각형중 검은 정사각형의 갯수. 기본 정사각형의 위치는 전체 정사각형에서 부터 시작해서 계속 4등분 해 가면서 그 정사각형이 포함된 구역의 번호들의 리스트이다. NW 구역은 0, NE 구역은 1, SE 구역은 2, SW 구역은 3이다. 무늬가 기본 정사각형 하나일 때는 그 정사각형의 위치는 빈 리스트가 된다. 예를 들어, 위치가 (3 3) 인 정사각형은 16개의 기본 정사각형으로 구성된 정사각형 판에서 가장 왼쪽 아래의 기본 정사각형을 말한다. 한 무늬가 가지는 기본 정사각형의 갯수는 $4^i$개 이고, 기본 정사각형의 위치를 표현하는 리스트의 길이는 항상 $i$가 된다. 이 조건을 만족할 때에만 `neighbor`가 정의된다.
- `pprint`: 정사각형 무늬를 화면에 그려준다.

예를 들어서 다음과 같이 벽지무늬들을 만들어서 프린트할 수 있겠다 (어떤 무늬가 프린트될까?)

```
(define B black)
(define W white)
(define Basic (glue B B B W))
(define (turn pattern i)
  (if (<= i 0) pattern else (turn (rotate pattern) (- i 1))))
(define Compound (glue Basic (turn Basic 1) (turn Basic 2) (turn Basic 3)))
```

위와 같은 프로그램을 고안하는 데, 무늬를 구현하는 방법이 몇 가지가 있다:

- 방법 1. 정사각형 무늬안에 있는 기본 정사각형들의 가로줄(row)의 리스트로 표현하는 방법. 예를 들어, 위의 예에서 `Basic`은 `((B B) (W B))`로, `Compound`는 `((B B W B) (W B B B) (B B B W) (B W B B))`로 표현되겠다.

- 방법 2. 잎새에 기본 정사각형이 매달린, 모든 가지가 4갈래로 갈라지는 트리 구조로 표현하는 방법.

위 두 가지 구현 방안을 구현하라. 이 두 가지 구현 방안을 모두 가지고 위의 여섯가지(상수 두 개, 함수 네 개)를 정의하라. 이 때 데이타의 표현방식 두 가지가 적절히 모두 사용되도록 정의한다.

배열로 구현하는 경우, 드러나는(인터페이스) 함수들:

$$\begin{aligned}
\texttt{glue-array-from-tree}: &\quad form * form * form * form \to form \\
\texttt{glue-array-from-array}: &\quad form * form * form * form \to form \\
\texttt{rotate-array}: &\quad form \to form \\
\texttt{neighbor-array}: &\quad location * form \to int \\
\texttt{pprint-array}: &\quad form \to void \\
\texttt{is-array?}: &\quad form \to bool
\end{aligned}$$

트리로 구현하는 경우, 드러나는 함수들:

$$\begin{aligned}
\texttt{glue-tree-from-tree}: &\quad form * form * form * form \to form \\
\texttt{glue-tree-from-array}: &\quad form * form * form * form \to form \\
\texttt{rotate-tree}: &\quad form \to form \\
\texttt{neighbor-tree}: &\quad location * form \to int \\
\texttt{pprint-tree}: &\quad form \to void \\
\texttt{is-tree?}: &\quad form \to bool
\end{aligned}$$

두 구현사이의 변환 함수들:

$$\begin{aligned}
\texttt{array-to-tree}: &\quad form \to form \\
\texttt{tree-to-array}: &\quad form \to form
\end{aligned}$$

□

**Exercise 2** "벽지 아가씨 심사위원"

벽지 무늬를 다루는 함수들에 다음의 함수를 추가로 정의하고:

$$\begin{aligned}
&\texttt{equal} : form * form \to bool \\
&\texttt{size} : form \to int
\end{aligned}$$

`equal`은 두 무늬가 같은지를 판별하고, `size`는 기본 정사각형의 갯수가 $4^i$일 때 $i$를 내놓는다. `equal`이 받아들이는 두개의 무늬들은 다르게 표현된 것들일 수 있다.

그렇게 드러난 함수들을 이용해서 함수 `beautiful`을 정의하라.

$$\texttt{beautiful} : \mathit{form} \to \mathit{bool}$$

함수 `beautiful`은 벽지 무늬가 중앙점을 기준으로 대칭이거나, 대칭이지 않다면 모든 정사각형의 이웃한 검은 정사각형들의 갯수가 1개보다 많고 6개보다 작을 때이다. □

**Exercise 3** "튜링기계(Turing machine)"

튜링기계에 대한 아래글을 읽고 튜링기계를 구현해 봅시다.

"튜링기계(Turing Machine)의 고안"

`ropas.snu.ac.kr/~kwang/paper/cs-ch1.pdf`

아래의 함수들을 만들어서 제출합니다.

- 테잎을 만들고 사용하는 함수들:

  $$\texttt{init-tape} : \mathit{symbol\ \ list} \to \mathit{tape}$$
  $$\texttt{read-tape} : \mathit{tape} \to \mathit{symbol}$$
  $$\texttt{write-tape} : \mathit{tape} * \mathit{symbol} \to \mathit{tape}$$
  $$\texttt{move-tape-left} : \mathit{tape} \to \mathit{tape}$$
  $$\texttt{move-tape-right} : \mathit{tape} \to \mathit{tape}$$
  $$\texttt{print-tape} : \mathit{tape} \to \mathit{void}$$

  테잎에 기록되는 *symbol*은 문자열입니다. 테잎의 빈칸은 "-" 문자가 있는 것으로 합니다. 읽기쓰기 헤드를 오른쪽/왼쪽으로 한 칸 움직이기 위해서 테잎을 반대방향인 왼쪽/오른쪽으로 한 칸을 움직여 주는 함수들을 사용하게됩니다. `print-tape`은 현재 헤드가 가르키는 테잎칸의 심볼을 프린트 합니다.

- 작동규칙표를 만들고 사용하는 함수들:

  $$\texttt{empty-ruletable} : \mathit{ruletable}$$
  $$\texttt{add-rule} : \mathit{rule} * \mathit{ruletable} \to \mathit{ruletable}$$
  $$\texttt{make-rule} : \mathit{state} * \mathit{symbol} * \mathit{symbol} * \mathit{move} * \mathit{state} \to \mathit{rule}$$
  $$\texttt{match-rule} : \mathit{state} * \mathit{symbol} * \mathit{ruletable} \to \mathit{symbol} \times \mathit{move} \times \mathit{state}$$

  튜링머신의 상태를 나타내는 *state*는 문자열(string)입니다. `make-rule`은 현재 상태, 읽은 심볼, 쓸 심볼, 헤드의 움직일 방향, 그리고 다음 상태로 구성된 룰을 만듭니다. `match-rule`은 현재 상태와 읽은 심볼로 규칙표를 보고, 매치되는 일거리(쓸 심볼, 헤드의 움직일 방향, 다음 상태)를 받습니다. 헤드의 움직일 방향(*move*)는 `'left`, `'right`, 혹은 `'stay`입니다.

- 튜링기계를 만들고 사용하는 함수들:

  `make-tm` : $symbol\ \ list * state * ruletable \rightarrow tm$
  `step-tm` : $tm \rightarrow tm$
  `run-tm` : $tm \rightarrow tm$
  `print-tm` : $tm * int \rightarrow void$

  함수 `make-tm`는 주어진 심볼들을 테잎에 순서대로 써서 초기화하고, 테잎의 첫번째 심볼의 위치에 읽기쓰기 헤드가 위치하고, 주어진 초기상태로 기계상태가 셋팅되고, 주어진 작동규칙표를 갖추게됩니다.

  함수 `step-tm`은 튜링기계의 작동규칙표에 따라 주어진 튜링기계를 한 스텝 실행시킨 후의 튜링기계를 내놓고, `run-tm`은 실행이 끝날 때까지 실행시키고 최종 튜링기계를 내놓습니다. 함수 `print-tm`은 주어진 튜링기계의 테잎을 프린트 하는데, 현재의 헤드 위치를 중심으로 좌우 $|n|$(두 번째 인자 $n$)개(총 $2|n|+1$개)의 심볼을 프린트 합니다. 프린트할때 심볼들마다 구분표는 "."으로 합니다. □

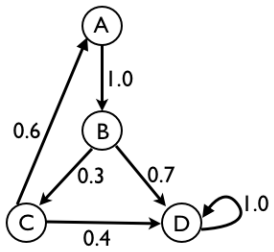
**Challenge 1** "그녀의 행방"

그녀와 백화점을 가면 우리는 각자 따로따로 매장을 돌아다닌다. 중간에 그녀를 만나려면 어느 매장으로 가봐야 할까? 그녀는 백화점 쇼핑 중에는 전화벨로 방해받고 싶지 않아서 핸드폰을 꺼놓고 있다.

우리는 주어진 시간에 그녀가 어느 매장에 있을 지를 예측하는 프로그램을 만들려고 한다. 주어진 시간에 각 매장별로 그녀가 그 곳에 있을만한 확률을 보여주는.

입력은 유한한 그래프와 양의 정수이다. 그래프는 그녀의 움직임을 모델한 것이고, 움직임은 10분 단위로 일어난다고 하자. 양의 정수는 백화점에서 헤어진지 몇 10분 째인지를 나타낸다. 그래프의 노드는 매장을 의미하고, 노드 사이의 화살표는 한 매장에서 다른 매장으로 이동하는 관계이고, 화살표에는 그 이동의 확률이 표현되어 있다. 한 노드에서 바깥으로 가는 화살표가 여럿 있을 수 있는데 그 화살표들에 적혀있는 확률들의 합은 반드시 1이어야 한다. 그녀가 백화점으로 들어와서 처음으로 방문하는 매장은 주어진 매장들 중에 같은 확률로(무작위로) 정해진다고 하자.

예를 들어, 그래프가




이면, A매장에서 B매장으로 항상 가고, B매장에서는 30% 확률로 C매장으로 움직이고, 등등.

이 경우, 임의의 매장에서 쇼핑을 시작해서 그녀가 10분후에 각 매장에 있을 확률은 A 15%, B 25%, C 7.5%, D 52.5% 이다. 20분후는 각각 4.5%, 15%, 7.5%, 73% 이다.

위와같은 결과를 계산하는 함수 catchYou를 작성하기 바랍니다. 매장은 다섯 개(A, B, C, D, E)로 정해져 있다고 합시다. 입력은 다섯 매장을 다니는 그녀의 움직임 그래프와 양의 정수입니다.

$$\texttt{catchYou} : \mathit{graph} * \mathit{int} \rightarrow (\mathit{store} \times \mathit{real})\mathit{list}$$

입력 그래프는 다음과 같이 표현합니다. 위의 그래프는
`(define model '((A B 1.0) (B C 0.3) (B D 0.7) (C A 0.6) (C D 0.4) (D D 1.0)))`
출력은 매장이름과 최종 확률 쌍들의 리스트로 합니다. 예를 들어,

```
(catchYou model 2)
```

은 다음의 리스트를 내 놓습니다:

```
((A . 4.5) (B . 15) (C . 7.5) (D . 73))
```

□