Homework 1

SNU 4541.574, 2007 가을

Due: 10/10(수), 24:00

Exercise 1 "리스트합"

작은 순서대로(ascending order) 나열된 정수 리스트 두개를 받아서 하나의 순서 리스트로 만드는 함수

```
merge: int list -> int list -> int list
```

를 정의하세요. 리스트에는 같은 정수가 반복해서 들어있지 않습니다. □

Exercise 2 "씨그마"

우리가 중고등 수학시간에 숱하게 썼던 다음의 "씨그마"를 정의하세요:

$$\Sigma_{n=a}^{b} f(n)$$

씨그마의 타입은

```
sigma : int -> int -> (int -> int) -> int.
```

즉, `sigma a b f`로 표현하면 $\Sigma_{n=a}^{b} f(n)$과 같도록. □

Exercise 3 "좋은사이"

두개의 정수 리스트를 받아서 각 리스트의 원소들을 차례대로 사이사이에 끼워주는 함수 `zipper`를 작성하세요:

```
zipper: int list -> int list -> int list
```

즉, `zipper [1;2;3] [4])`는 `[1;4;2;3]`을 만들어낸다. □

Exercise 4 "좋은사이들"

정수 리스트의 리스트를 받아서 각 리스트의 원소들을 차례대로 사이사이에 끼워주는 함수 `zipperN`을 작성하세요:

```
zipperN: int list list -> int list
```

즉, `zipperN [[1;2;3]; [4]; [9;10;11]]`는 `[1;4;9;2;10;3;11]`을 만들어낸다. □

Exercise 5 "참거짓"

Propositional Logic 식들(`formula`)을 다음과 같이 정의했습니다:

```
type formula = TRUE
             | FALSE
             | NOT of formula
             | ANDALSO of formula * formula
             | ORELSE of formula * formula
             | IMPLY of formula * formula
             | LESS of expr * expr
and  expr = NUM of int
          | PLUS of expr * expr
          | MINUS of expr * expr
```

주어진 `formula`를 받아서 참값을 만들어내는 함수 `eval`

```
eval: formula -> bool
```

를 정의하세요. □

Exercise 6 "k-친수"

일반적으로 k진수($k > 1$)는 다음과 같이 표현한다.

$$d_0 \cdots d_n$$

여기서

$$\forall d_i \in \{0, \cdots, k-1\}.$$

그리고 "$d_0 \cdots d_n$"은 크기가

$$d_0 \times k^0 + \cdots + d_n \times k^n$$

인 정수를 표현한다.

이것을 살짝 확장해서 "k친수"를 다음과 같이 정의해보자. 표현은

$$d_0 \cdots d_n$$

여기서

$$\forall d_i \in \{1-k, \cdots, 0\} \cup \{0, \cdots, k-1\}.$$

그리고 "$d_0 \cdots d_n$"은 크기가

$$d_0 \times k^0 + \cdots + d_n \times k^n$$

인 정수를 표현한다.

예를 들어, 2친수의 경우를 생각하자. 베이스가 $\{-1, 0, 1\}$이 되겠다. `0`이 0을, `+`가 1을 `-`가 -1을 표현한다고 하면, `+` 는 1을, `+0+`는 5를, `+-`는 -1을, `+-0-`는 -9인 정수를 표현한다.

이러한 2친수 N의 집합을 귀납적으로 정의하면 다음과 같다:

$$\begin{array}{rcl} N & ::= & \texttt{0} \\ & | & \texttt{+} \\ & | & \texttt{-} \\ & | & \texttt{0}N \\ & | & \texttt{+}N \\ & | & \texttt{-}N \end{array}$$

nML로 2친수라는 타입을 다음과 같이 정의했습니다:

```
type crazy2 = NIL | ZERO of crazy2 | ONE of crazy2 | MONE of crazy2
```

예를 들어, `0+-`은

```
ZERO(ONE(MONE NIL))
```

로 표현됩니다.

자 이제, 위와 같이 표현되는 2친수를 받아서 그것의 값을 계산하는 함수 `crazy2val`을 정의하세요.

```
crazy2val: crazy2 -> int
```

□

Exercise 7 두 2친수를 받아서 2친수의 합에 해당하는 2친수를 내어놓는 함수 `crazy2add`를 정의하세요.

```
crazy2add: crazy2 -> crazy2 -> crazy2
```

위의 crazy2add는 다음의 성질이 만족되야 한다: 임의의 2친수 z 과 z'에 대해서

$$\texttt{crazy2val (crazy2add } z\ z'\texttt{)} = \texttt{(crazy2val } z\texttt{)} + \texttt{(crazy2val } z'\texttt{)}.$$

□

Exercise 8 "계산기 mathenatica"

다음의 계산기

```
mathenatica: exp -> real
```

를 만듭시다.

```
type exp = Var of string
         | INT of int
         | REAL of real
         | ADD of exp * exp
         | SUB of exp * exp
         | MUL of exp * exp
         | DIV of exp * exp
         | SIGMA of string * exp * exp * exp
         | INTEGRAL of string * exp * exp * exp
```

예를들어 우리가 쓰는 수식이 exp타입으로는 다음과 같이 표현된다:

$\sum_{x=1}^{10}(x-1)$	`SIGMA(‘‘x’’, INT 1, INT 10, SUB(Var ‘‘x’’, INT 1))`
$\int_{x=1.0}^{10.0}(x-1)\mathrm{d}x$	`INTEGRAL(‘‘x’’, REAL 1.0, REAL 10.0, SUB(Var ‘‘x’’, INT 1))`

적분식을 계산할때의 알갱이 크기($\mathrm{d}x$)는 0.1로 정한다.

□

Exercise 9 "실행기"

Implement an interpreter intp

```
intp: stm -> store -> store
```

of the following language. The input to the interpreter is a value of the type stm.

```
type id = string

type binop = PLUS | MINUS | TIMES | DIV

type stm = ASSIGN of id * exp
         | PRINT of exp list
         | SEQ of stm list

and exp = VAR of id
        | CONST of int
        | BINOP of exp * binop * exp
        | IF of exp * exp * exp
```

An example progam would be

```
  SEQ[ASSIGN("a", BINOP(CONST 5, PLUS, CONST 3))
      ;
      ASSIGN("b",IF(BINOP(VAR"a",MINUS,CONST 1),
                    BINOP(CONST 10, TIMES, VAR"a"),
                    BINOP(CONST 20, DIV, VAR"a")))
      ;
      PRINT[VAR "b"]]
```

The semantics is as follows. `SEQ` has a list of statement each of which is evaluated from left to right. `ASSIGN("x",e)` assigns a value of `e` to identifier `x`. `PRINT` prints out the values of expressions in the list from left to right separated by comma ending with newline. Expression `VAR "x"` is the value of `x`. `BINOP` applies the binary operator to the operands. `IF` is usual conditional expression, where depending on whether the first expression's value is zero or not it evaluates the second expression (in case of non-zero) or the third expression. □