Homework 4
SNU 4190.210 Spring 2006

due: 5/30(Tue) 24:00

Exercise 1 "merge ordered list"

순서대로 나열된 정수 리스트 두개를 받아서 하나의 순서 리스트로 만드는 함수 `merge`를 정의하라. 리스트에는 같은 값이 반복해서 들어있지 않습니다.

Exercise 2 정수식들의 구조를 다음의 타입으로 정의했습니다:

```
type expr = NUM of int
          | PLUS of expr * expr
          | MINUS of expr * expr
          | MULT of expr * expr
          | DIVIDE of expr * expr
          | MAX of expr list
```

주어진 `expr`를 받아서 정수값을 만들어내는 함수 `eval`

```
eval: expr -> int
```

를 정의하세요. 이때, `MAX [1,3,2] = 3`, 즉 `MAX`는 정수식 리스트에서 가장 큰 정수를 찾아내는 정수식입니다. 빈 리스트의 경우는 0을 의미하는 정수식 입니다.

□

Exercise 3 "Leftist Heaps: 왼쪽편에 쏠려있는 힙"

우선큐(priority queue, "유별난 큐")라는 구조의 핵심은, 원소를 넣고 빼는 것 보다는, 제일가는 원소를 알아보는 데에 유난히 특화되 있다는 것입니다. 힙(heap)이 대표적인 것이지요. 그중에서도 왼쪽으로 쏠린 힙(leftist heap, 왼쏠힙)이라는 것을 구현해 봅시다.

- 왼쏠힙: 힙은 힙인데 모든 왼쪽 노드의 급수가 오른쪽 형제 노드의 급수보다 크거나 같다.
- 노드의 급수: 그 노드에서 오른쪽으로만 타고 내려가서 끝날 때 까지 내려선 횟수, 즉 오른편 척추의 길이.
- 힙: 이진 나무 구조로서 모든 갈래길 길목의 값이 갈라진 후의 모든 노드들의 값보다 작거나 같다.

왼쏠힙은 다음의 타입으로 정의됩니다:

```
type heap = EMPTY | NODE of rank * value * heap * heap
and rank = int
and value = int
```

넣고, 빼고, 하는 등의 함수는 다음으로 정의됩니다:

```
exception EmptyHeap
fun rank EMPTY = 0
  | rank NODE(r,_,_,_) = r
fun insert(x,h) = merge(h, NODE(0,x,EMPTY,EMPTY))
fun findMin EMPTY = raise EmptyHeap
  | findMin NODE(_,x,_,_) = x
fun deleteMin EMPTY = raise EmptyHeap
  | deleteMin NODE(_,x,lh,rh) = merge(lh,rh)
```

나머지 함수 `merge`

```
merge: heap * heap -> heap
```

를 정의하세요. 이 때, 왼쏠힙의 장점을 살려서 여러분이 정의한 `merge`는 $O(\log n)$으로 끝나도록 해야 합니다 (n은 힙의 노드 수). (참고사실: 왼쏠힙에서 오른쪽 척추에 붙어있는 노드수는 많아야 $\lfloor \log(n+1) \rfloor$입니다.) 정의할 때 다음의 함수를 이용하시기를:

```
fun shake (x,lh,rh) = if (rank lh) >= (rank rh)
                      then NODE(rank rh + 1, x, lh, rh)
                      else NODE(rank lh + 1, x, rh, lh)
```

□

Exercise 4 "nMathematica"

고등학교때는 손으로하고, Maple이나 Mathematica에서는 자동으로 해주던 미분식 전개를 만들어보자.

Write a program `diff`

```
diff: ae * string -> ae
```

that performs symbolic differentiation of algebraic expressions. For example, if the arguments to the program are ax^2+bx+c and x, the program should return $2ax+b$. The input algebraic expression is a value of type `ae`.

```
type ae = CONST of int
        | VAR of string
        | POWER of string * int
        | TIMES of ae list
        | SUM of ae list
```

For example, $ax^2 + bx + c$ is

```
SUM [TIMES [VAR "a", POWER("x",2)],
     TIMES [VAR "b", VAR "x"],
     VAR"c"]
```

□

Exercise 5 "Queue = 2 Stacks"

Queue can be implemented such that both the `enqueue` and `dequeue` operations have almost-constant time complexities.

Types of each operations are:

```
empty: queue
enqueue: queue * elmt -> queue
dequeue: queue -> elmt * queue
```

A queue $[a_1, \cdots, a_m, b_1, \cdots, b_n]$ (b_n is the head) is represented as a pair of two lists L and R:

$$L = [a_1, \cdots, a_m], \quad R = [b_n, \cdots, b_1].$$

After adding an element `x`, new queue becomes

$$[x, a_1, \cdots, a_m], [b_n, \cdots, b_1].$$

After deleting an element, new queue becomes

$$[a_1, \cdots, a_m], [b_{n-1}, \cdots, b_1].$$

When deleting, we sometimes have to reverse the L list and let it be the R list. Empty queue is $([], [])$.

Implement a structure `IntQuickQueue` that contains the queque type and functions `enqueue` and `dequeue`. Queue element is `int`. □