Homework 2
ENE4014 Programming Languages, Spring 2024
due: 4/17(Wed), 23:59

- Submit one file per problem via the submission system in the course website. Make sure that your files are compiled and run without errors.
- Do not use any external libraries.

Exercise 1 Write a function

```
npower: int -> int -> float
```

that returns $\frac{1}{x^n}$ for two given integers x and $n(\geq 0)$. x^0 is defined to be 1. □

Exercise 2 Write a function

```
gcd: int -> int -> int
```

that returns the greatest common divisor (GCD) of two given non-negative integers. Use the Euclidean algorithm based on the following definition (for two integers n and m ($n \geq m$)):

$$\texttt{gcd}\ n\ m = \begin{cases} n & (m = 0) \\ \texttt{gcd}\ (n-m)\ m & \end{cases}$$

□

Exercise 3 Write a function

```
min: int list -> int
```

that returns the minimum value of a given list of integers. If the list is empty, return 0. □

Exercise 4 Write a function

```
cartesian:  'a list -> 'b list -> ('a * 'b) list
```

that returns a list of from two lists. That is, for lists A and B, the Cartesian product $A \times B$ is the list of all ordered pairs (a, b) where $a \in A$ and $b \in B$. For example, if $A = [\text{“}a''; \text{“}b''; \text{“}c'']$ and $B = [1; 2; 3]$, $A \times B$ is defined to be

$$[(\text{“}a'', 1); (\text{“}a'', 2); (\text{“}a'', 3); (\text{“}b'', 1); (\text{“}b'', 2); (\text{“}b'', 3); (\text{“}c'', 1); (\text{“}c'', 2); (\text{“}c'', 3)]$$

□

Binary trees can be defined as follows:

```
type btree = Leaf | Node of int * btree * btree
```

The number in the `Node` constructor is called the key of the node.

Exercise 5 Write a function

```
count_leaves :  btree -> int
```

that takes a binary tree and returns the number of all leaves in the tree. For example,

```
# let t =  Node (2, Node (2, Leaf, Leaf), Node (3, Leaf, Leaf)) ;;
val t : btree = Node (2, Node (2, Leaf, Leaf), Node (3, Leaf, Leaf))
# count_leaves t ;;
- : int = 4
```

□

Exercise 6 Write a function

```
count_oddnode :  btree -> int
```

that takes a binary tree and returns the number of odd keys in the tree. For example,

```
# let t =  Node (1, Node (2, Leaf, Leaf), Node (3, Leaf, Leaf)) ;;
val t : btree = Node (2, Node (2, Leaf, Leaf), Node (3, Leaf, Leaf))
# count_oddnode t ;;
- : int = 2
```

□

Exercise 7 Write a function

```
insert_btree :  int -> btree -> btree
```

that takes an integer and a binary search tree and returns a new binary search tree with the integer properly inserted in the tree. A binary search tree (BST) is a tree where the key of each node is greater than all keys in its left subtree and less than all keys in its right subtree. For example,

```
# let t =  Node (2, Node (2, Leaf, Leaf), Node (3, Leaf, Leaf)) ;;
val t : btree = Node (2, Node (2, Leaf, Leaf), Node (3, Leaf, Leaf))
# insert_btree 1 t ;;
- : btree = Node (2, Node (2, Node (1, Leaf, Leaf), Leaf), Node (3, Leaf, Leaf))
```

□

Exercise 8 Write a function

```
duplicate:  'a list -> 'a list
```

that duplicates the elements of a list. For example,

```
duplicate [1; 2; 3] = [1; 1; 2; 2; 3; 3].
```

□

Exercise 9 Write a function

```
replicate:  'a list -> int -> 'a list
```

that replicates the elements of a list a given number $n (\geq 0)$ of times. If n is 0, the function should return an empty list. For example,

```
replicate [1; 2; 3] 3 = [1; 1; 1; 2; 2; 2; 3; 3; 3].
```

□

Exercise 10 Write a function

```
deduplicate:  'a list -> 'a list
```

that takes a list and returns a list with all duplicates removed. The order of the elements in the result should be the same as the order in the original list. For example,

```
deduplicate [1; 1; 2; 2; 3; 3; 2; 2] = [1; 2; 3].
```

Exercise 11 Write a function

```
lall:  'a list -> ('a -> bool) -> bool
```

such that

$$\texttt{lall}\ l\ p = \begin{cases} \texttt{true} & \text{(if } p \text{ holds for all elements of } l\text{)} \\ \texttt{false} & \text{(otherwise)} \end{cases}$$

For example,

```
lall [1; 2; 3] (fun x -> x > 0) = true
```

and

```
lall [1; 2; 3] (fun x -> x > 1) = false.
```

□

Exercise 12 Write a function

```
lany:  'a list -> ('a -> bool) -> bool
```

such that

$$\texttt{lany}\ l\ p = \begin{cases} \texttt{true} & \text{(if } p \text{ holds for at least one element of } l\text{)} \\ \texttt{false} & \text{(otherwise)} \end{cases}$$

For example,

```
lany [1; 2; 3] (fun x -> x mod 2 = 0) = true
```

and

```
lany [1; 2; 3] (fun x -> x < 0) = false.
```

□

Exercise 13 Write a function

```
powerset:  'a list -> 'a list list
```

such that `powerset` l returns the list of all subsets of l. For example, if $l = [1; 2; 3]$, then `powerset` l is defined to be

$$[[]; [1]; [2]; [3]; [1; 2]; [1; 3]; [2; 3]; [1; 2; 3]].$$

You don't have to consider the order of the elements in the result. For example, both $[[2; 1]; [1]; [2]; []]$ and $[[1]; [1; 2]; [2]; []]$ are correct answers for `powerset` $[1; 2]$.
□