Problem Set 4

If debugging is the process of removing software bugs,
then programming must be the process of putting them in.
— *Dijkstra*

- This problem set is due **at 9:00am** on **February 24, 2025** .
- This problem set comprises 3 problems.
- Submit early. Do **not** wait till the last moment.
- Each solution should start on a new page.
- We will give full credit **only** for correct solutions that are described clearly and convincingly.

Problem 4-1. I am a Binary Type in a Binary World... [20 points]

Consider the definition of binary numbers as follows:

```
type bin =
  | E          (* Empty or Zero *)
  | O of bin   (* Zero bit *)
  | I of bin   (* One bit *)
```

Under this representation, binary numbers are encoded from left to right (unlike the traditional right-to-left encoding). For example:

```
let one   = I E            (* 1 *)
let two   = O (I E)        (* 10 *)
let three = I (I E)        (* 11 *)
let four  = O (O (I E))    (* 100 *)
```

Your task is to implement a module in OCaml that supports the following functionality:

1. Operator Overloading: Redefine the operators =, <, >, +, −, *, and / to work with the bin type.
2. String Representation: Implement functionality to convert a bin value to its string representation (e.g., I (I E) should print as "11").
3. Parsing: Implement a function to parse a binary string (e.g., "101") into a bin value.
4. Conversions: Implement functions to convert between integers and bin values (e.g., convert 5 to its bin representation and vice versa).

Ensure your module is well-structured and handles edge cases appropriately.

Problem 4-2. Addition is More Than Just Repeated Successor! [35 points]

Your module should include the definition of addition with the type signature:

```
bin_add : bin -> bin -> bin
```

The operator '+' should be overloaded to work as an infix operator for `bin_add`. Prove the following three theorems for your implementation:

(a) [10 points] First prove:

Theorem 1 *(Additive Identity) For all inputs* `n` *of type* `bin`*, we have:*

$$\texttt{bin\_add n E} = \texttt{bin\_add E n} = \texttt{n} \tag{1}$$

That is, `E` is a right and left identity for `bin_add`.

(b) [10 points] Then prove:

Theorem 2 *(Commutativity) For all inputs* `n` *and* `m` *of type* `bin`*, we have:*

$$\texttt{bin\_add}\ \texttt{n}\ \texttt{m} = \texttt{bin\_add}\ \texttt{m}\ \texttt{n} \tag{2}$$

That is, `bin_add`, is a commutative binary function.

(c) [15 points] And finally prove:

Theorem 3 *(Associativity) For all inputs* `n`, `m`, `p` *and of type* `bin`*, we have:*

$$\texttt{bin\_add}\ \texttt{n}\ (\texttt{bin\_add}\ \texttt{m}\ \texttt{p}) = \texttt{bin\_add}\ (\texttt{bin\_add}\ \texttt{n}\ \texttt{m})\ \texttt{p} \tag{3}$$

Problem 4-3. Multiplication is More Than Just Repeated Addition! [35 points]

Your module should also include the definition of multiplication with the type signature:

```
bin_mult : bin -> bin -> bin
```

The operator '*' should be overloaded to work as an infix operator for `bin_mult`. Prove the following three theorems for your implementation:

(a) [10 points] First prove:

Theorem 4 *(Zero) For all inputs* `n` *of type* `bin`*, we have:*

$$\texttt{bin\_mult}\ \texttt{n}\ \texttt{E} = \texttt{bin\_mult}\ \texttt{E}\ \texttt{n} = \texttt{E} \tag{4}$$

That is, multiplication by `E` yields `E`.

(b) [10 points] Then prove:

Theorem 5 *(Commutativity) For all inputs* `n` *and* `m` *of type* `bin`*, we have:*

$$\texttt{bin\_mult}\ \texttt{n}\ \texttt{m} = \texttt{bin\_mult}\ \texttt{m}\ \texttt{n} \tag{5}$$

That is, `bin_mult`, alike `bin_add`, is a commutative function.

(c) [15 points] And finally prove:

Theorem 6 *(Distributivity) For all inputs* `n`, `m`, `p` *and of type* `bin`*, we have:*

$$\texttt{bin\_mult}\ \texttt{n}\ (\texttt{bin\_add}\ \texttt{m}\ \texttt{p}) = \texttt{bin\_add}\ (\texttt{bin\_mult}\ \texttt{n}\ \texttt{m})\ (\texttt{bin\_mult}\ \texttt{n}\ \texttt{p}) \tag{6}$$