```
expr := <number> | <name> | true | false            type expr =
     |  (if <expr> <expr> <expr>)                      | ENum of int | EBool of bool | EId of string
     |  (let (<name> <expr>) <expr>)                   | EIf of expr * expr * expr
     |  (+ <expr> <expr>)                              | ELet of string * expr * expr
     |  (< <expr> <expr>)                              | EPlus of expr * expr
     |  (set <name> <expr>)                            | ELess of expr * expr
     |  (fun (<name> : <t>) : <t> <expr>)              | ESet of string * expr
     |  (<expr> <expr>)                                | EApp of string * expr * expr
                                                       | EFun of string * typ * expr


t := Num | Bool | (<t> <t> -> <t>)                    and typ = TNum | TBool | TArrow of typ * typ


prog := <expr>                                        type prog = expr
```

```
expr := <number> | <name> | true | false            type expr =
     |  (if <expr> <expr> <expr>)                      | ENum of int | EBool of bool | EId of string
     |  (let (<name> <expr>) <expr>)                   | EIf of expr * expr * expr
     |  (+ <expr> <expr>)                              | ELet of string * expr * expr
     |  (< <expr> <expr>)                              | EPlus of expr * expr
     |  (set <name> <expr>)                            | ELess of expr * expr
     |  (<name> <expr> <expr>)                         | ESet of string * expr
                                                       | EApp of string * expr


def := (def <name> (<name> : <t>) : <t>              type def =
         <expr>)                                       | DFun of string * string * typ * typ * expr


t := Num | Bool                                       type typ = TNum | TBool


prog := def ... <expr>                                type prog = def list * expr
```