Quiz 8

CSCI 334: Spring 2024

Your name: _____

A MOVELANG program computes a final location after a sequence of moves starting from location { `x = 0; y = 0` }. Going north (`N`) subtracts `1` from the current `y` while going south (`S`) adds `1`.

A MOVELANG program has the following grammar:

```
<expr> ::= <move>*
<move> ::= N | S
```

For example, `NNNS` returns { `x = 0; y = -2` }.

A complete MOVELANG implementation is below.

```
type Location = { x: int; y: int }

type Move = North | South

let move = (pchar 'N' |>> (fun _ -> North)) <|>
           (pchar 'S' |>> (fun _ -> South))

let grammar = pleft (pmany0 move) peof

let parse (input: string) : Option<Move list> =
    match grammar (prepare input) with
    | Success (ast,_) -> Some ast
    | Failure   (_,_) -> None

let evalMove (loc: Location)(move: Move) : Location =
    match move with
    | North -> { x = loc.x; y = loc.y - 1 }
    | South -> { x = loc.x; y = loc.y + 1 }

let rec eval (loc: Location)(moves: Move list) : Location =
    match moves with
    |    [] -> loc
    | p::ps ->
        let loc' = evalMove loc p
        eval loc' ps

[<EntryPoint>]
let main (args: string[]) : int =
    let loc = { x = 0; y = 0 }
    match parse args[0] with
    | Some moves ->
        let loc2 = eval loc moves
        printfn "Final location: (%d, %d)" loc2.x loc2.y
        0
    | None ->
        printfn "Invalid program"
        1
```

Suppose the specification of MOVELANG is updated to the following. Solution highlighted in red below.

> A MOVELANG program computes a final location after a sequence of moves starting from location `{ x = 0; y = 0 }`. Going north (`N`) subtracts `1` from the current `y` while going south (`S`) adds `1`. Going west (`W`) subtracts `1` from the current `x` while going east (`E`) adds `1`.
>
> A MOVELANG program has the following grammar:
>
> ```
> <expr> ::= <move>*
> <move> ::= N | S | E | W
> ```
>
> For example, `NNNSWW` returns `{ x = -2; y = -2 }`.

```
type Location =  x: int; y: int

type Move = North | South | East | West

let move = (pchar 'N' |>> (fun _ -> North)) <|>
           (pchar 'S' |>> (fun _ -> South)) <|>
           (pchar 'E' |>> (fun _ -> East)) <|>
           (pchar 'W' |>> (fun _ -> West))

let grammar = pleft (pmany0 move) peof

let parse (input: string) : Option<Move list> =
    match grammar (prepare input) with
    | Success (ast,_) -> Some ast
    | Failure   (_,_) -> None

let evalMove (loc: Location)(move: Move) : Location =
    match move with
    | North -> { x = loc.x     ; y = loc.y - 1 }
    | South -> { x = loc.x     ; y = loc.y + 1 }
    | East  -> { x = loc.x + 1; y = loc.y      }
    | West  -> { x = loc.x - 1; y = loc.y      }

let rec eval (loc: Location)(moves: Move list) : Location =
    match moves with
    |    [] -> loc
    | p::ps ->
        let loc' = evalMove loc p
        eval loc' ps

[<EntryPoint>]
let main (args: string[]) : int =
    let loc = { x = 0; y = 0 }
    match parse args[0] with
    | Some moves ->
        let loc2 = eval loc moves
        printfn "Final location: (%d, %d)" loc2.x loc2.y
        0
    | None ->
        printfn "Invalid program"
        1
```