CIS341 Midterm 2022 Appendices

(Do not write answers in the appendices. They will not be graded)

Appendix A: Interpreter Code

```
type var = string

type exp =
  | Var of var
  | Imm of int
  | Add of exp * exp
  | Mul of exp * exp

type cmd =
  | Skip
  | Assn of var * exp
  | Seq of cmd * cmd
  | IfNZ of exp * cmd * cmd
  | WhileNZ of exp * cmd
  | For of var * exp * cmd      (* <------ This is the new construct *)

type state = (var * int) list

let set (s:state) (x:var) (v:int) =
  (x,v)::s

let rec get (s:state) (x:var) : int =
  begin match s with
    | [] -> 0   (* uninitialized variables are 0 *)
    | (y,v)::rest -> if x = y then v else get rest x
  end

let rec interp_exp (e:exp) (s:state) : int =
  begin match e with
    | Var x -> get s x
    | Imm v -> v
    | Add(e1, e2) -> (interp_exp e1 s) + (interp_exp e2 s)
    | Mul(e1, e2) -> (interp_exp e1 s) * (interp_exp e2 s)
  end
```

```
let rec interp_cmd (c:cmd) (s:state) : state =
  begin match c with
    | Skip -> s

    | Assn(x, e)   -> set s x (interp_exp e s)

    | Seq(c1, c2)  -> interp_cmd c2 (interp_cmd c1 s)

    | IfNZ(e, c1, c2) ->
      interp_cmd (if (interp_exp e s) <> 0 then c1 else c2) s

    | WhileNZ(e, c) ->
      interp_cmd (IfNZ(e, Seq(c, WhileNZ(e, c)), Skip)) s

    | For(x, e, c) ->
      let s0 = set s x 0 in
      let rec loop s =
        let e = interp_exp e s in
        let v = get s x in
        if v = e then s else
          let s' = interp_cmd c s in
          let v' = get s' x in
          loop (set s' x (v'+1))
      in
      loop s0
  end

(* The cmd [factorial_for] computes factorial of 6 using a for loop
   (and the available SIMPLE arithmetic instructions):  *)

let factorial_for : cmd =
  let x = "x" in
  let ans = "ans" in
  Seq(Assn(x, Imm 6),
      Seq(Assn(ans, Imm 1),
          For("y", Var x,
              Assn(ans, Mul(Var ans, (Add(Var x, Mul(Var "y", Imm(-1)))))))))
```

Appendix B: Lexer Code

```
{
open Lexing
type token =  | Int    of int64  | Ident  of string  | LPAREN
              | LPARENSTAR  | STARRPAREN  | IF
let print_token t =
  begin match t with
    | Int x   -> (Printf.printf "Int %Ld\n%!" x)
    | Ident s -> (Printf.printf "Ident  %s\n%!" s)
    | IF      -> (Printf.printf "IF\n%!")
    | LPAREN  -> (Printf.printf "LPAREN\n%!")
    | LPARENSTAR -> (Printf.printf "LPARENSTAR\n%!")
    | STARRPAREN -> (Printf.printf "STARRPAREN\n%!")
  end
let acc = ref []
let emit t = acc := t::(!acc)
exception Lex_error of char
}

let character = ['a'-'z''A'-'Z']
let digit = ['0'-'9']
let underscore = ['_']
let whitespace = [' ' '\t' '\n' '\r']
let identifier =  character (character|digit|underscore)*

rule lex = parse
  | "if"           { emit IF; lex lexbuf }
  | identifier     { emit (Ident (lexeme lexbuf)); lex lexbuf }
  | '('            { emit LPAREN; lex lexbuf }
  | "(*"           { emit LPARENSTAR; lex lexbuf }
  | "*)"           { emit STARRPAREN; lex lexbuf }
  | whitespace+    { lex lexbuf }
  | digit+         { emit (Int (Int64.of_string (lexeme lexbuf))); lex lexbuf }
  | eof            { List.rev (!acc) }
  | _ as c         { raise (Lex_error c) }

{
let _ =
  try
    List.iter print_token (lex (from_channel stdin))
  with
    | Lex_error c -> Printf.printf "Char %s is unexpected.\n" (Char.escaped c)
}
```