```
#lang plai

; HW2 due 10.10 Tue
; 22100579 JinjuLee

; Task1 Implement F1WAE with deferred substitution.
; Solved by myself: half Y (I learnd from textbook)
; Time taken: about 3 hour

; <F1WAE> ::= <num>
;       | {+ <F1WAE> <F1WAE>}
;       | {- <F1WAE> <F1WAE>}
;       | {* <F1WAE> <F1WAE>}
;       | {with {<id> <F1WAE>} <F1WAE>}
;       | <id>
;       | {<id> <F1WAE>}


;; abstract
(define-type F1WAE
  [num (n number?)] ; for number
  [add (lhs F1WAE?) (rhs F1WAE?)] ; add
  [sub (lhs F1WAE?) (rhs F1WAE?)] ; sub
  [with (name symbol?) (named-expr F1WAE?) (body F1WAE?)] ; variable binding
  [id (name symbol?)] ; name of variabl
  [app (fun-name symbol?) (arg F1WAE?)]) ;name of function

; function abstract
(define-type FunDef
  [fundef (fun-name symbol?) ; name of function
          (arg-name symbol?) ; name of parameter of function
          (body F1WAE?)]) ; body of funtion

; data abstract, for deferred substitution
(define-type DefrdSub
  [mtSub] ; empty one for no substitution
  [aSub (name symbol?) (value number?) (ds DefrdSub?)]) ; substitute name to
value, ds is next one(if no need, use mtSub)
```

```
; Task2. Update the language to support multiplication by applying syntactic
sugar and desugaring.
; You must define the `desugar' function.
; Solved by myself: Y
; Time taken: about 3 days

;; desugar function for mul
; [contract] desugar: number number -> F1WAE expr
; [purpose] implement mul(desugaring) to F1WAE expression consist of others
; [test]    (test (desugar 2 3) (add (num 3) (add (num 3) (num 0))))
;           (test (desugar -2 3) (add (num -3) (add (num -3) (num 0))))
;           (test (desugar -3 2) (add (num -2) (add (num -2) (add (num -2)
(num 0)))))
(define (desugar l r)
    (cond
        [(= l 0)     (num 0)]
        [(> l 0)     (add (num r) (desugar (- l 1) r))]
        [else        (add (num (- 0 r)) (desugar (+ l 1) r))])
)
(test (desugar 2 3) (add (num 3) (add (num 3) (num 0))))
(test (desugar -2 3) (add (num -3) (add (num -3) (num 0))))
(test (desugar -3 2) (add (num -2) (add (num -2) (add (num -2) (num 0)))))


;; parser
; [contract] parse : sexp -> F1WAE
; [purpose] parse expression to F1WAE abstract form. (using desugar func for
mul case)
; [test]    (test (parse'(+ 10 5)) (add (num 10) (num 5)))
;           (test (parse '(* 3 2)) (add (num 2) (add (num 2) (add (num 2) (num
0)))))
;           (test (parse '(with (x 10) (- x 3))) (with 'x (num 10) (sub (id
'x) (num 3))))
(define (parse sexp)
  (match sexp
    [(? number?)               (num sexp)]
    [(list '+ l r)             (add (parse l) (parse r))]
    [(list '- l r)             (sub (parse l) (parse r))]
    [(list '* l r)             (desugar l r)]
    [(list 'with (list i v) e)  (with i (parse v) (parse e))]
    [(? symbol?)               (id sexp)]
    [(list f a)                (app f (parse a))]
    [else                      (error 'parse "Bad syntax: ~a" sexp)]))
(test (parse'(+ 10 5)) (add (num 10) (num 5)))
(test (parse '(* 3 2)) (add (num 2) (add (num 2) (add (num 2) (num 0)))))
(test (parse '(with (x 10) (- x 3))) (with 'x (num 10) (sub (id 'x) (num 3))))
```

```
;; interpreter
; [contract] lookup-fundef : symbol listof(FunDef) --> FunDef
; [purpose] find the funtion in funcdefs list with fun-name recursively
; [test]    (test (lookup-fundef 'fun_1 fun-defs) (fundef 'fun_1 'x (add (id
'x) (num 1))))
;           (test (lookup-fundef 'fun_2 fun-defs) (fundef 'fun_2 'n (sub (id
'n) (num 10))))
;           (test (lookup-fundef 'fun_3 fun-defs) (fundef 'fun_3 'x (add (id
'x) (num 3))))
(define (lookup-fundef fun-name fundefs)
  (cond
    [(empty? fundefs) (error fun-name "function not found")] ; if not found
    [else (if (symbol=? fun-name (fundef-fun-name (first fundefs)))
              (first fundefs) ; if it is, return it
              (lookup-fundef fun-name (rest fundefs)))])) ; if its not, lookup
next element
(define fun-defs
    (list (fundef 'fun_1 'x (add (id 'x) (num 1)))
        (fundef 'fun_2 'n (sub (id 'n) (num 10)))
        (fundef 'fun_3 'x (add (id 'x) (num 3)))))
(test (lookup-fundef 'fun_1 fun-defs) (fundef 'fun_1 'x (add (id 'x) (num
1))))
(test (lookup-fundef 'fun_2 fun-defs) (fundef 'fun_2 'n (sub (id 'n) (num
10))))
(test (lookup-fundef 'fun_3 fun-defs) (fundef 'fun_3 'x (add (id 'x) (num
3))))

; [contract] lookup : symbol DefrdSub → number
; [purpose] find the value of the variable in ds with the name recursively
; [test]
(define (lookup name ds)
  (type-case DefrdSub ds
    [mtSub () (error 'lookup "no binding for identifier")] ; if not found
    [aSub (bound-name bound-value rest-ds)
          (if (symbol=? bound-name name) ; find in ds recursively
              bound-value
              (lookup name rest-ds))]))
(define vars (aSub 'x 10 (aSub 'y 20 (aSub 'z 30 (mtSub)))))
(test (lookup 'x vars) 10)
(test (lookup 'y vars) 20)
(test (lookup 'z vars) 30)
```

```
; [contract] interp : F1WAE listof(fundef) DefrdSub → number
; [purpose] for the expression of F1WAE expr, return the appropriate valule
from fun-defs or ds
; [test]    (test (interp (parse '(+ 1 3)) empty (mtSub)) 4)
;           (test (interp (parse '(* -2 -3)) empty (mtSub)) 6)
;           (test (interp (parse '(with (x 10) (+ x x))) empty (mtSub)) 20)
(define (interp expr fun-defs ds)
  (type-case F1WAE expr
    [num (n) n] ; return just number
    [add (l r) (+ (interp l fun-defs ds) (interp r fun-defs ds))] ; return the
value added
    [sub (l r) (- (interp l fun-defs ds) (interp r fun-defs ds))] ; return the
value subtracted
    [with (bound-id named-expr bound-body) ; call the interp again with aSub
          (interp bound-body
                  fun-defs
                  (aSub bound-id ; replace bound-id to interpreted named-expr
                        (interp named-expr fun-defs ds) ds))]
    [id   (v) (lookup v ds)] ; infd value of variabl using lookup
    [app  (fun-name arg-expr) ; applicate the function using lookup-fundef
          (local ([define the-fun-def (lookup-fundef fun-name fun-defs)]) ;
find the actual function using lookup-fundef and define it in here
                 (interp (fundef-body the-fun-def) ; interprete the body of
function with...
                         fun-defs
                         (aSub (fundef-arg-name the-fun-def) ; replace the
arg-name of the-fun-def as
                               (interp arg-expr fun-defs ds) ; result of
interpreted arg-expr(input)
                               (mtSub))))]
))
(test (interp (parse '(+ 1 3)) empty (mtSub)) 4)
(test (interp (parse '(* -2 -3)) empty (mtSub)) 6)
(test (interp (parse '(with (x 10) (+ x x))) empty (mtSub)) 20)
```

```
; Task3. At the bottom of your code, put the following comments and put
answers.
; Solved by myself: Y
; Time taken: about 8 hour

; Q1. Which scope is supported for a free identifier in a function call in
your implementation, static scope, dynamic scope or both?
; -> my implementation supports the static scope.

; Put the three test cases that show your answer for Q1 is correct.
(define fd
  (list (fundef 'f1 'x (add (id 'x) (num 10)))
        (fundef 'f2 'y (add (id 'y) (app 'f1 (num 5))))
        (fundef 'f3 'z (add (id 'z) (id 'x)))
        (fundef 'f4 'k (id 'x))
        ))
(define ds (aSub 'x 5 (aSub 'y 7 (aSub 'z 10 (mtSub)))))


; 1) This test case shows my implementation supports static scope.
(test (interp (parse '(f2 2)) fd ds) 17)

; 2) This test case shows my implementation does not support dynamic scope
(test/exn (interp (parse '(with (x 2) (f3 10))) fd (mtSub)) "no binding for
identifier")

; 3) This test case shows my implementation does not support dynamic scope
(test/exn (interp (parse '(with (x 10) (f4 0))) fd ds) "no binding for
identifier")
```