פקולטה: מדעי הטבע

מחלקה: מדעי המחשב ומתמטיקה

שם הקורס: שפות תכנות

קוד הקורס: 7036010

תאריך בחינה: 15/07/2019 סמ' ב' מועד ב'

משך הבחינה: שעתיים ורבע

שם המרצה: ערן עמרי

חומר עזר: אסור

שימוש במחשבון: לא

הוראות כלליות:

- **קראו היטב את הוראות המבחן.**
- כתבו את תשובותיכם בכתב קריא ומרווח (**במחברת התשובות בלבד** ולא על גבי טופס המבחן עצמו.
- בכל שאלה או סעיף (**שבהם נדרשת כתיבה, מעבר לסימון תשובות נכונות**), ניתן לכתוב – **לא יודע/ת** (מבלי להוסיף דבר מעבר לכך) – ולקבל 20% מהניקוד על השאלה או הסעיף (אין זה תקף לחלקי סעיף).
- אפשר להסתמך על סעיפים קודמים גם אם לא עניתם עליהם.
- יש לענות על כל השאלות.
- בכל מקום הדורש תכנות, מותר להגדיר פרוצדורות עזר (אלא אם נאמר אחרת במפורש).
- ניתן להשיג עד 106 נקודות במבחן.
- לנוחיותכם מצורפים שני קטעי קוד עבור ה- interpreter של FLANG בסוף טופס המבחן. הראשון – במודל הסביבות והשני – במודל ה-substitution cache. כמו כן, מופיעות פרוצדורות eval, subst של מודל ההחלפות.

בהצלחה!

אוניברסיטת
אריאל
בשומרון

שאלה 1 — with vs. call — (27 נקודות):

בכיתה דיברנו על כך שכל קוד חוקי (תכנית) בשפה FLANG יכול להיות מוחלף בקוד חוקי שקול, אשר אינו מכיל ביטויי with. בכיתה דיברנו על החלפת בנאי With בעזרת הבנאים Call ו-Fun. אולם, למעשה, ההחלפה יכולה להתבצע כבר ברמת הקוד.

סעיף א' – מקרה פרטי – (12 נקודות): נתון הקוד הבא בשפה FLANG.

```
"{with {foo {with {x 3} {fun {y} {+ x y}}}}
    {with {h {+ 4 5}}
      {call foo {- 7 1}}}}"
```

מצאו עבורו קוד שקול (המגדיר את אותו חישוב בדיוק) שאינו מכיל המילה השמורה with (כמוסבר למעלה). הקפידו על הזחה (אינדנטציה) נכונה. הסבירו בקצרה מדוע הביטוי שקיבלתם שקול.

הדרכה: מומלץ לעבוד בשלבים. מצאו ביטוי העומד בפני עצמו (דורש שינוי יחיד) – אשר יש צורך לשנותו. שנו אותו ראשון והמשיכו באותו אופן עם הביטוי שהתקבל.

סעיף ב' – פתרון כללי – (15 נקודות):

הניחו שהבנאי With הושמט מהקוד של האינטרפטר (למשל, במודל הסביבות). כלומר, השורה [With Symbol FLANG FLANG] נמחקה מהגדרת הטיפוס FLANG.

כתבו מהו החלק בקוד של האינטרפטר במודל הסביבות שיש לשנותו על מנת שהאינטפרטר יבצע חישוב שקול על-פי הרעיונות שבראש השאלה. שימו לב שישנם בדיוק שני מקומות (בשתי פרוצדורות שונות של האינטרפרטר) הדורשים שינוי מקומי. כתבו מהו השינוי הנדרש (אם יש קוד שצריך להוסיף או לשנות, כתבו במפורש את הקוד החדש. אינכם רשאים לשנות את הגדרת הטיפוס FLANG (ובפרט, לא ניתן להוסיף מחדש את הבנאי שנמחק).

כמובן, שתחביר השפה לא השתנה וביטויים המכילים with כמילה שמורה – עדיין חוקיים.

שאלה 2 — Functions as first class — (30 נקודות):

סעיף א' – המושגים first class, higher order, first order – (8 נקודות):

בכיתה דיברנו על שלושה שלבים בהתפתחות של ההתייחסות לפונקציות בשפות תכנות.

1. **הסבירו במשפט או שניים על כל אחד מהמושגים בכותרת הסעיף**
2. **תנו דוגמה לשפה המתייחסת לפונקציות על פי הגדרת מושג זה.**

המושגים הם first class, higher order, first order.

אוניברסיטת
אריאל
בשומרון

סעיף ב' – יצירת פונקציה המייצרת זוגות של מספרים – (15 נקודות):

בשפה pl **(הווריאנט של ראקט שבו השתמשנו במהלך הקורס) לא ניתן לייצר זוג** (pair) **שאינו רשימה. בסעיף זה נכתוב מימוש שלנו לייצור זוגות (של מספרים בלבד).**

כתבו פונקציה, `make-pair` אשר מייצרת זוג של שני מספרים. בפרט, היא מקבלת כקלט שני מספרים **ומחזירה כפלט פונקציה מסימבול (אחד מתוך שניים אפשריים) למספר (ראו דוגמאות מטה).**

להלן דוגמאות הרצה (מועתק מהממשק של סביבת העבודה):

```
> (: pair1 : ((U 'first 'second) -> Number))
(define pair1 (make-pair 3 4))
> pair1
- : ((U 'first 'second) -> Number)
#<procedure:pair>
> (pair1 'first)
- : Number
3
> (+ (pair1 'second) (pair1 'first))
- : Number
7
> (pair1 'third)
  Type Checker: type mismatch
  expected: (U 'first 'second)
  given: 'third in: (quote third)
```

שימו לב:

1. `make-pair` אמורה להחזיר פונקציה. עליכם להסיק את הטיפוס של הפונקציה מהדוגמאות מעלה. בדקו על מה היא מופעלת וכן מה הטיפוס שהיא מחזירה. ניתן לראות זאת מהטיפוס של האובייקט המוחזר.
2. בדקו את עצמכם שהקוד שכתבתם מתנהג בהתאם לדוגמאות מעלה.
3. בדוגמה התחתונה, סביבת האחרונה מוציאה הודעת השגיאה בגלל שימוש לא נכון.

השלימו את החלקים החסרים בקוד החלקי הבא.

```
(: make-pair :  <--fill in 1--> )
(define (make-pair l r)
   <--fill in 2-->
  <--fill in 3-->)
```

סעיף ג' – (7 נקודות): הסבירו במשפט או שניים, מה היה קורה אם היינו מריצים את הקוד שלכם ואחריו את דוגמאות ההרצה מהסעיף הקודם (בתוך המלבן) בשפה pl dynamic **(וריאנט של** pl **במודל** dynamic scoping**). השוו להרצה בשפה** pl **(במודל הסטטי) והזכירו בתשובתכם את המושגים סביבה,** closure**. אפשרי, אך לא חובה להשתמש בציור המתאר את הסביבות שנפתחות.**

שאלה 3 — (24 נקודות): נתון הקוד הבא:

```
(run "{with {y 4}
        {with {z {fun {z} y}}
           {with {y 7}
              {call z z}}}}")
```

סעיף א' (17 נקודות):

נתון התיאור החסר הבא עבור הפעלות הפונקציה **eval** בתהליך ההערכה של הקוד מעלה במודל ה-**substitution-cache** (על-פי ה-interpreter התחתון מבין השניים המצורפים מטה). השלימו את המקומות החסרים - באופן הבא – לכל הפעלה מספר *i* תארו את AST_i – הפרמטר האקטואלי הראשון בהפעלה מספר *i (עץ התחביר האבסטרקטי), את* $Cache_i$ – הפרמטר האקטואלי השני בהפעלה מספר *i (רשימת ההחלפות) ואת* RES_i – הערך המוחזר מהפעלה מספר *i*.

הסבירו בקצרה כל מעבר שהשלמתם (אין צורך להסביר מעברים קיימים). ציינו מהי התוצאה הסופית.
שימו לב: ישנן השלמות עם אותו שם – מה שמצביע על השלמה זהה – אין צורך להשלים מספר פעמים, אך ודאו כי מספרתם נכון את ההשלמות במחברת.

AST_1 = `<--fill in 1-->`
$Cache_1$ = `'()`
RES_1 = `<--fill in 2-->`
AST_2 = `(Num 4)`
$Cache_2$ = `'()`
RES_2 = `(Num 4)`
AST_3 = `(With 'z (Fun 'z (Id 'y)) (With 'y (Num 7) (Call (Id 'z) (Id 'z))))`
$Cache_3$ = `(('y (Num 4)))`
RES_3 = `<--fill in 2-->`
AST_4 = `(Fun 'z (Id 'y))`
$Cache_4$ = `(('y (Num 4)))`
RES_4 = `<--fill in 3-->`
AST_5 = `(With 'y (Num 7) (Call (Id 'z) (Id 'z)))`
$Cache_5$ = `<--fill in 4-->`
RES_5 = `<--fill in 2-->`
AST_6 = `(Num 7)`
$Cache_6$ = `<--fill in 4-->`
RES_6 = `(Num 7)`
AST_7 = `(Call (Id 'z) (Id 'z))`
$Cache_7$ = `<--fill in --5>`
RES_7 = `<--fill in --2>`
AST_8 = `<--fill in --6>`
$Cache_8$ = `<--fill in --7>`
RES_8 = `<--fill in --8>`
AST_9 = `(Id 'z)`
$Cache_9$ = `<--fill in --9>`
RES_9 = `<--fill in --10>`
AST_{10} = `<--fill in --11>`
$Cache_{10}$ = `<--fill in --12>`
RES_{10} = `<--fill in --13>`

`Final result: ?`

סעיף ב' (7 נקודות):

הסבירו מה היה קורה אם היינו מבצעים הערכה של הקוד מעלה במודל ההחלפות – אין צורך בחישוב מלא (הסבירו). מהי התשובה שעליה החלטנו בקורס כרצויה? מדוע?

תשובה מלאה לסעיף זה לא תהיה ארוכה מחמש שורות (תשובה ארוכה מדי תקרא חלקית בלבד).

שאלה 4 — שינוי השפה FLANG – הוספת טיפול בזוגות — (25 נקודות):

לצורך פתרון שאלה זו נעזר בקוד ה- interpreter של FLANG במודל הסביבות, המופיע בסוף טופס המבחן. נרצה להרחיב את השפה ולאפשר שימוש בזוגות (של מספרים). ביתר פירוט – נוסיף ביטויי **cons** ואופרטורים **get first** כמו גם **get second** (על-פי תחביר מיוחד המודגם בטסטים מטה).

להלן דוגמאות לטסטים שאמורים לעבוד (בפרט, כל טסט שעבד לפני הרחבת השפה, ימשיך לפעול גם לאחר ההרחבה):

```
;; tests

(test (run "{get first {cons 2 3}}") => 2)

(test (run "{with {x 4}
                {get first {cons x 3}}}") => 4)
(test (run "{with {x 1}
              {with {foo {fun {r} {get second r}}}
                {with {pair1 {cons 4 x}}
                   {call foo pair1}}}}") => 1)
(test (run "{get second {cons {+ 5 2}
                              {call {call {fun {x} {call x 1}}
                                          {fun {x} {fun {y} {+ x y}}}}
                                    123}}}") => 124)
(test (run "{with {pair {fun {x} {fun {y} {+ x y}}}}
              {get second pair}}")
=error> "extract-val-from-pair: expected a pair, got (FunV x (Fun y (Add
                                          (Id x) (Id y))) (EmptyEnv))")
(test (run "{cons 1 3}")
=error> "run: evaluation returned a non-number: (PairV #<procedure:pair>)")
```

רוב הקוד הנדרש ניתן לכם מטה ואתם נדרשים רק להשלים את הדקדוק, את הפארסר ואת הפונקציות שיאפשרו הערכת ביטויים. למען הקיצור, מובאות בפניכם רק התוספות לקוד הקיים. (שלוש נקודות "..." מופיעות במקום הקוד המושמט). **הקוד לשימושכם, אך אין צורך להתעמק בכולו**. השתמשו רק בחלקים שחשובים לכם לפתרון סעיפים א', ב' ו-ג', הנתונים מטה.

סעיף א' — עדכון הדקדוק — (5 נקודות):
עדכנו את הדקדוק, כך שיתאים לטסטים וההגדרות הכתובות מעלה. השלימו את השורות החסרות.

```
  ;; The Flang interpreter with pairs, using environments
  #lang pl
  #|  The grammar:
    The grammar:
    <FLANG> ::= <num>
              ...
              | <<-- fill in 1 -->>
              | <<-- fill in 2 -->>
              | <<-- fill in 3 -->>
              ...
|#
  (define-type FLANG
  ...
    [Cons FLANG FLANG]
    [First FLANG]
    [Second FLANG]))
```

סעיף ב' — עדכון ה-parser — (6 נקודות):
עדכנו את הקוד הבא, כך שיתאים לטסטים וההגדרות הכתובות מעלה. השלימו את השורות החסרות.

```
(: parse-sexpr : Sexpr -> FLANG)
;; to convert s-expressions into FLANGs
(define (parse-sexpr sexpr)
  (match sexpr
    [(number: n) (Num n)]
...
    [<<-- fill in 1 -->> (First (parse-sexpr p-expr))]
    [<<-- fill in 2 -->>]
    [<<-- fill in 3 -->>]
    ...))

(: parse : String -> FLANG)
;; parses a string containing a FLANG expression to a FLANG AST
(define (parse str)
  (parse-sexpr (string->sexpr str)))
```

סעיף ג' — eval — (14 נקודות): נתונות לכם ההגדרות הבאות עבור הפונקציה eval. הן עודכנו בהתאם להגדרת השפה המורחבת. **קראו היטב את הדרישות הפורמאליות הבאות** וכתבו את eval על-פיהן. הוסיפו את הקוד הנדרש (היכן שכתוב `—«fill-in»—` -- סה"כ חמישה מקומות).

```
  #|  Evaluation rules:
    eval(N,env)            = N
    ...
    eval({ cons E1 E2})     = makePair (N1, N2)
                           if eval (E1, env) = N1 and eval (E2, env) = N2
                            = error!           otherwise
    eval({get first E})     = N1
                           if eval (E, env) = pair (N1,N2)
                           = error!           otherwise
    eval({get second E})   = N2
```

```
                                if eval (E, env) = pair (N1,N2)
                                = error!          otherwise
  |#
;; ;; Types for environments, values, and a lookup function

 (define-type ENV
   [EmptyEnv]
   [Extend Symbol VAL ENV])

 (define-type VAL; הקפידו להתאים את הפתרון להגדרות הבאות
   [NumV Number]
   [FunV Symbol FLANG ENV]
   [PairV ((U 'first 'second) -> Number)])

(: lookup : Symbol ENV -> VAL)
(define (lookup name env)
  (...))

  (: NumV->number : VAL -> Number); arith-op-פונקציה זאת חיצונית ל
    (define (NumV->number v)
      (cases v
        [(NumV n) n]
        [else (error 'arith-op "expects a number, got: ~s" v)]))

  (: arith-op : (Number Number -> Number) VAL VAL -> VAL)
  ;; gets a Racket numeric binary operator, and uses it within a NumV
  ;; wrapper
  (define (arith-op op val1 val2)
    (NumV (op (NumV->number val1) (NumV->number val2))))
```

גם הפונקציה הבאה נועדה לסייע לכם. היא מקבלת VAL, שאמור להיות זוג (עטוף ב-PairV) וסימבול (אחד מתוך שניים אפשריים) ומחזירה את הערך המספרי (עטוף בהתאם) הנמצא בזוג במקום המתאים לסימבול. השלימו בה את הקוד החסר.

```
(: extract-val-from-pair : (U 'first 'second) VAL -> VAL)
(define (extract-val-from-pair location pair)
  (cases pair
    [<<-- fill in 1 -->>]
    [else <<-- fill in 2 -->>]))
```

לבסוף, השלימו את הגדרת הפונקציה eval תוכלו להשתמש בקוד שכתבתם מעלה, כמו גם הב

```
  (: eval : FLANG ENV -> VAL)
  ;; evaluates FLANG expressions by reducing them to values
  (define (eval expr env)
    (cases expr
      [(Num n) (NumV n)]
      ...
      [(First p) <<-- fill in 3 -->>]
      [<<-- fill in 4 -->>]
      [(Cons f s) <<-- fill in 5 -->>]))
```

<u>הערה:</u> הקפידו לשלוח את הטיפוס הנכון בכל קריאה לפונקציה ולהחזיר את הטיפוס הנכון מכל פונקציה.

```
--<<<FLANG-ENV>>>-----------------------------------------------------
  ;; The Flang interpreter, using environments
  #lang pl
  #|  The grammar:
    <FLANG> ::= <num>
              | { + <FLANG> <FLANG> }
              | { - <FLANG> <FLANG> }
              | { * <FLANG> <FLANG> }
              | { / <FLANG> <FLANG> }
              | { with { <id> <FLANG> } <FLANG> }
              | <id>
              | { fun { <id> } <FLANG> }
              | { call <FLANG> <FLANG> }
  Evaluation rules:
    eval(N,env)                = N
    eval({+ E1 E2},env)        = eval(E1,env) + eval(E2,env)
    eval({- E1 E2},env)        = eval(E1,env) - eval(E2,env)
    eval({* E1 E2},env)        = eval(E1,env) * eval(E2,env)
    eval({/ E1 E2},env)        = eval(E1,env) / eval(E2,env)
    eval(x,env)                = lookup(x,env)
    eval({with {x E1} E2},env) = eval(E2,extend(x,eval(E1,env),env))
    eval({fun {x} E},env)      = <{fun {x} E}, env>
    eval({call E1 E2},env1)
             = eval(Ef,extend(x,eval(E2,env1),env2))
                               if eval(E1,env1) = <{fun {x} Ef}, env2>
             = error!          otherwise
  |#
  (define-type FLANG
    [Num  Number]
    [Add  FLANG FLANG]
    [Sub  FLANG FLANG]
    [Mul  FLANG FLANG]
    [Div  FLANG FLANG]
    [Id   Symbol]
    [With Symbol FLANG FLANG]
    [Fun  Symbol FLANG]
    [Call FLANG FLANG])
  (: parse-sexpr : Sexpr -> FLANG)
  ;; to convert s-expressions into FLANGs
  (define (parse-sexpr sexpr)
    (match sexpr
      [(number: n)     (Num n)]
      [(symbol: name)  (Id name)]
      [(cons 'with more)
       (match sexpr
         [(list 'with (list (symbol: name) named) body)
          (With name (parse-sexpr named) (parse-sexpr body))]
         [else (error 'parse-sexpr "bad `with' syntax in ~s" sexpr)])]
      [(cons 'fun more)
       (match sexpr
         [(list 'fun (list (symbol: name)) body)
          (Fun name (parse-sexpr body))]
         [else (error 'parse-sexpr "bad `fun' syntax in ~s" sexpr)])]
```

```
    [(list '+ lhs rhs) (Add (parse-sexpr lhs) (parse-sexpr rhs))]
    [(list '- lhs rhs) (Sub (parse-sexpr lhs) (parse-sexpr rhs))]
    [(list '* lhs rhs) (Mul (parse-sexpr lhs) (parse-sexpr rhs))]
    [(list '/ lhs rhs) (Div (parse-sexpr lhs) (parse-sexpr rhs))]
    [(list 'call fun arg) (Call (parse-sexpr fun) (parse-sexpr arg))]
    [else (error 'parse-sexpr "bad syntax in ~s" sexpr)]))
(: parse : String -> FLANG)
;; parses a string containing a FLANG expression to a FLANG AST
(define (parse str)
  (parse-sexpr (string->sexpr str)))

;; Types for environments, values, and a lookup function
(define-type ENV
  [EmptyEnv]
  [Extend Symbol VAL ENV])
(define-type VAL
  [NumV Number]
  [FunV Symbol FLANG ENV])
(: lookup : Symbol ENV -> VAL)
(define (lookup name env)
  (cases env
    [(EmptyEnv) (error 'lookup "no binding for ~s" name)]
    [(Extend id val rest-env)
     (if (eq? id name) val (lookup name rest-env))]))

(: arith-op : (Number Number -> Number) VAL VAL -> VAL)
;; gets a Racket numeric binary operator, and uses it within a NumV
;; wrapper
(define (arith-op op val1 val2)
  (: NumV->number : VAL -> Number)
  (define (NumV->number v)
    (cases v
      [(NumV n) n]
      [else (error 'arith-op "expects a number, got: ~s" v)]))
  (NumV (op (NumV->number val1) (NumV->number val2))))

(: eval : FLANG ENV -> VAL)
;; evaluates FLANG expressions by reducing them to values
(define (eval expr env)
  (cases expr
    [(Num n) (NumV n)]
    [(Add l r) (arith-op + (eval l env) (eval r env))]
    [(Sub l r) (arith-op - (eval l env) (eval r env))]
    [(Mul l r) (arith-op * (eval l env) (eval r env))]
    [(Div l r) (arith-op / (eval l env) (eval r env))]
    [(With bound-id named-expr bound-body)
     (eval bound-body
           (Extend bound-id (eval named-expr env) env))]
    [(Id name) (lookup name env)]
    [(Fun bound-id bound-body)
     (FunV bound-id bound-body env)]
    [(Call fun-expr arg-expr)
     (let ([fval (eval fun-expr env)])
       (cases fval
         [(FunV bound-id bound-body f-env)
```

```
            (eval bound-body
                  (Extend bound-id (eval arg-expr env) f-env))]
           [else (error 'eval "`call' expects a function, got: ~s"
                                fval)]))]))
  (: run : String -> Number)
  ;; evaluate a FLANG program contained in a string
  (define (run str)
    (let ([result (eval (parse str) (EmptyEnv))])
      (cases result
        [(NumV n) n]
        [else (error 'run
                     "evaluation returned a non-number: ~s" result)])))
```

```
--<<<FLANG-Substitution-cache>>>-----------------------------------------
  ;; The Flang interpreter, using Substitution-cache
                        החלק של ה-parser מושמט, כיוון שהוא זהה למודל הסביבות
  ;; a type for substitution caches:
  (define-type SubstCache = (Listof (List Symbol FLANG)))
  (: empty-subst : SubstCache)
  (define empty-subst null)

  (: extend : Symbol FLANG SubstCache -> SubstCache)
  (define (extend name val sc)
    (cons (list name val) sc))

  (: lookup : Symbol SubstCache -> FLANG)
  (define (lookup name sc)
    (let ([cell (assq name sc)])
      (if cell
        (second cell)
        (error 'lookup "no binding for ~s" name))))

  (: arith-op : (Number Number -> Number) FLANG FLANG -> FLANG)
  ;; gets a Racket numeric binary operator, and uses it within a FLANG
  ;; `Num' wrapper
  (define (arith-op op expr1 expr2)
    (: Num->number : FLANG -> Number)
    (define (Num->number e)
      (cases e
        [(Num n) n]
        [else (error 'arith-op "expects a number, got: ~s" e)]))
    (Num (op (Num->number expr1) (Num->number expr2))))

  (: eval : FLANG SubstCache -> FLANG)
  ;; evaluates FLANG expressions by reducing them to expressions
  (define (eval expr sc)
    (cases expr
      [(Num n) expr]
      [(Add l r) (arith-op + (eval l sc) (eval r sc))]
      [(Sub l r) (arith-op - (eval l sc) (eval r sc))]
      [(Mul l r) (arith-op * (eval l sc) (eval r sc))]
      [(Div l r) (arith-op / (eval l sc) (eval r sc))]
      [(With bound-id named-expr bound-body)
       (eval bound-body
```

```
            (extend bound-id (eval named-expr sc) sc))]
      [(Id name) (lookup name sc)]
      [(Fun bound-id bound-body) expr]
      [(Call fun-expr arg-expr)
       (let ([fval (eval fun-expr sc)])
         (cases fval
           [(Fun bound-id bound-body)
            (eval bound-body
                  (extend bound-id (eval arg-expr sc) sc))]
           [else (error 'eval "`call' expects a function, got: ~s"
                        fval)]))]))

  (: run : String -> Number)
  ;; evaluate a FLANG program contained in a string
  (define (run str)
    (let ([result (eval (parse str) empty-subst)])
      (cases result
        [(Num n) n]
        [else (error 'run
                     "evaluation returned a non-number: ~s" result)])))
```

הפרוצדורות eval **ו-** subst **מתוך האינטרפרטר של מודל ההחלפות:**

```
(: subst : FLANG Symbol FLANG -> FLANG)
  ;; substitutes the second argument with the third argument in the
  ;; first argument, as per the rules of substitution; the resulting
  ;; expression contains no free instances of the second argument
  (define (subst expr from to)
    (cases expr
      [(Num n) expr]
      [(Add l r) (Add (subst l from to) (subst r from to))]
      [(Sub l r) (Sub (subst l from to) (subst r from to))]
      [(Mul l r) (Mul (subst l from to) (subst r from to))]
      [(Div l r) (Div (subst l from to) (subst r from to))]
      [(Id name) (if (eq? name from) to expr)]
      [(With bound-id named-expr bound-body)
       (With bound-id
             (subst named-expr from to)
             (if (eq? bound-id from)
               bound-body
               (subst bound-body from to)))]
      [(Call l r) (Call (subst l from to) (subst r from to))]
      [(Fun bound-id bound-body)
       (if (eq? bound-id from)
         expr
         (Fun bound-id (subst bound-body from to)))]))
(: eval : FLANG -> FLANG)
  ;; evaluates FLANG expressions by reducing them to *expressions*
  (define (eval expr)
    (cases expr
      [(Num n) expr]
      [(Add l r) (arith-op + (eval l) (eval r))]
      [(Sub l r) (arith-op - (eval l) (eval r))]
      [(Mul l r) (arith-op * (eval l) (eval r))]
      [(Div l r) (arith-op / (eval l) (eval r))]
      [(With bound-id named-expr bound-body)
       (eval (subst bound-body
```

```
              bound-id
              (eval named-expr)))]
[(Id name) (error 'eval "free identifier: ~s" name)]
[(Fun bound-id bound-body) expr]
[(Call fun-expr arg-expr)
 (let ([fval (eval fun-expr)])
   (cases fval
     [(Fun bound-id bound-body)
      (eval (subst bound-body bound-id (eval arg-expr)))]
     [else (error 'eval "`call' expects a function, got: ~s"
                   fval)]))]))
```