

פקולטה: מדעי הטבע

מחלקה: מדעי המחשב ומתמטיקה

שם הקורס: שפות תכנות

קוד הקורס: 2-7036010

תאריך בחינה: 23/06/2021 סמ' ב' מועד א' _____

משך הבחינה: שתיים 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 יודעת/ (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 108 נקודות במבחן. לנוחיותכם מצורפים שני קטעי קוד עבור ה- interpreter של FLANG בסוף טופס המבחן. הראשון – במודל ההחלפות והשני במודל ה- substitution cache (השני ניתן ללא החלק של הניתוח התחבירי, שכן הוא זהה בכולם). בנוסף, כחלק משאלה 4, נתון לכם האינטרפרטר במודל הסביבות.

שאלה 1 – מושגים בסיסיים – (15 נקודות):

במהלך הקורס ראינו/כתבנו מספר שפות:

1. Racket
2. PL
3. FLANG (substitution model)
4. FLANG (substitution-cache model)
5. FLANG (environment model)

חלקו את השפות הנ"ל לקבוצות על-פי כל אחת מן השאלות הבאות (הסבירו את תשובותיכם – תשובה ללא הסבר תחשב כניחוש):

- א. האם השפה מתנהגת לפונקציות כ- first order (פונקציה היא תווית בזיכרון), כ- higher order (פונקציה היא אובייקט שניתן לשלוח לפונקציה אחרת או לקבל כערך מוחזר מפונקציה) או כ- first class (פונקציה היא טיפוס וניתן לייצרה בזמן ריצה)?
- ב. האם השפה מתייחסת לטיפוסים באופן סטאטי (צריך להצהיר על הטיפוס של שמות מזהים) או דינאמי (הטיפוס של שם מזהה נקבע בזמן ריצה) – static or dynamic typing?
- ג. האם השפה מממשת קריאה לפונקציות באופן סטאטי או דינאמי – static or dynamic scoping?

הערה: החלוקות עבור א, ב, ו-ג עשויות להיות שונות זו מזו. ההסברים צריכים להיות תמציתיים.

שאלה 2 – שמות מזהים בשפה – (25 נקודות):

תזכורת: במהלך הקורס החלטנו שרצוי להוסיף לשפה את האפשרות לתת שמות מזהים לערכים ולקטעי קוד.

סעיף א' – (6 נקודות): מנו לפחות ארבעה יתרונות (שונים זה מזה) שהזכרנו להוספת שמות מזהים – כתבו הסבר קצר וממצה ליד כל יתרון (לא יותר משני משפטים).

סעיף ב' – (9 נקודות): באימפרטור של השפה FLANG במודל ההחלפות כתבנו את הפרוצדורה eval. הקוד הבא הוא חלק מפרוצדורה זאת:

```
[ (With bound-id named-expr bound-body)
  (eval (subst bound-body
               bound-id
               (eval named-expr))) ]
```

הסבירו בקצרה מה היה משתנה אם היינו מחליפים את הביטוי הנ"ל בביטוי:

```
[ (With bound-id named-expr bound-body)
  (eval (subst bound-body
               bound-id
               named-expr)) ]
```

מדוע בחרנו באפשרות הראשונה? היו תמציתיים ומדויקים בתשובתכם. תנו קוד המדגים את טענתכם.

סעיף ג' – (10 נקודות): באינטרפרטר של השפה **FLANG** במודל ה-substitution-cache כתבנו את הפרוצדורה eval. הקוד הבא הוא חלק מפרוצדורה זאת:

```
[(With bound-id named-expr bound-body)
 (eval bound-body
  (extend bound-id (eval named-expr sc) sc))]
```

הסבירו מה היה משתנה אם היינו מחליפים את הביטוי (eval named-expr sc) בביטוי named-exp ומדוע בחרנו באפשרות זאת.

(הניחו שגם החלפנו את הביטוי [(Id name) (lookup name sc)] בביטוי [(Id name) (eval (lookup name sc) sc)] – על מנת לאפשר את חישוב הערך).

הסבירו את השוני מהסעיף הקודם. היו תמציתיים ומדויקים בתשובתכם.

רמז: התבוננו בקוד הבא:

```
(run "{with {x 1}
      {with {y {+ x 1}}
        {+ y
          {with {x 2} y}}}}")
```

שאלה 3 – (28 נקודות): נתון הקוד הבא:

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

סעיף א' (18 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.

```
AST1 = <--fill in 1-->
Cache1 = '()
RES1 = <--fill in 2-->
AST2 = <--fill in 3-->
Cache2 = '()
RES2 = <--fill in 4-->
AST3 = (Fun 'z (Id 'y))
Cache3 = <--fill in 5-->
RES3 = <--fill in 6-->
AST4 = (With 'y (Num 7) (Call (Id 'w) (Id 'y)))
```

```
Cache4 = <--fill in 7-->
RES4 = <--fill in 2-->
AST5 = (Num 7)
Cache5 = <--fill in 7-->
RES5 = (Num 7)
AST6 = (Call (Id 'w) (Id 'y))
Cache6 = <--fill in 8-->
RES6 = <--fill in 2-->
AST7 = (Id 'w)
Cache7 = <--fill in --8>
RES7 = <--fill in --9>
AST8 = (Id 'y)
Cache8 = <--fill in 8-->
RES8 = <--fill in --10>
AST9 = <--fill in --11>
Cache9 = <--fill in --12>
RES9 = <--fill in --13>
```

Final result: ?

סעיף ב' (10 נקודות):

הסבירו מה היה קורה אם היינו מבצעים הערכה של הקוד מעלה במודל ההחלפות – אין צורך בחישוב מלא (הסבירו) [6 נקודות].

מהי התשובה שעליה החלטנו בקורס כרצויה? מדוע? [4 נקודות].

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

שאלה 4 – הרחבת השפה ושינוי המימוש – (40 נקודות):

מבוא: לצורך פתרון שאלה זו נעזר בקוד ה- interpreter של FLANG במודל הסביבות (הקוד כולו נתון לכם במהלך השאלה). נרצה להרחיב את השפה ולשנות את המימוש בשני האופנים הבאים:

1. סוגי קריאה לפונקציות – אנחנו נאפשר בשפה שלנו שני סוגי קריאה לפונקציה:
 - a. על-פי static-scoping – ראו ביטויי call-static – בקריאה כזאת, הפונקציה רצה בסביבה המרחיבה את סביבת ה-closure.
 - b. על-פי dynamic-scoping – ראו ביטויי call-dynamic – בקריאה כזאת, הפונקציה רצה בסביבה המרחיבה את הסביבה בה מתבצעת הקריאה.
2. טיפול בביטויי with – אנו נטפל בביטויי with **ללא בנאי מיוחד** עבורם (בפרט, לא יהיה בנאי With). לחילופין, אנו נשתמש בבנאים קיימים (כלומר, נתייחס לביטויים אלה כ-syntactic sugar עבור ביטוי אחר).

להלן דוגמאות לטסטים שאמורים לעבוד:

```
;; tests
(test (run "{with {add3 {fun {x} {+ x 3}}}
            {call-static add3 1}}")
      => 4)
(test (run "{with {add3 {fun {x} {+ x 3}}}
            {with {add1 {fun {x} {+ x 1}}}
              {with {x 3}
                {call-static add1 {call-static add3 x}}}}}")
      => 7)
(test (run "{call-dynamic {fun {x} {+ x 1}} 4}")
      => 5)
(test (run "{with {add3 {fun {x} {+ x 3}}}
            {call-dynamic add3 1}}")
      => 4)
(test (run "{with {add3 {fun {x} {+ x 3}}}
            {with {add1 {fun {x} {+ x 1}}}
              {with {x 3}
                {call-static add1 {call-dynamic add3 x}}}}}")
      => 7)
(test (run "{call {fun {x} {+ x 1}} 4}")
      =error> "parse-sexpr: bad syntax in (call (fun (x) (+ x 1)) 4)")
```

סעיף א' – עדכון הדקדוק והטיפוס FLANG (7 נקודות):

בהמשך למבוא לשאלה ולטסטים מעלה, עדכנו את הדקדוק והטיפוס. עליכם רק לציין בצורה ברורה את השורות/חלקים שתמצאו למחוק ואת השורות/חלקים שתמצאו להוסיף. **הסבירו את בחירותיכם.** שימו לב ששורת הבנאי With כבר נמחקה עבורכם.

```
#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> }

|#

(define-type FLANG
  [Num Number]
  [Add FLANG FLANG]
  [Sub FLANG FLANG]
  [Mul FLANG FLANG]
  [Div FLANG FLANG]
  [Id Symbol]
  ;[With Symbol FLANG FLANG] - there should be no such variant
  [Fun Symbol FLANG]
  [Call FLANG FLANG])
```

סעיף ב' parse (10 נקודות):

עדכנו את parse-sexpr בהתאם. עליכם רק לציין בצורה ברורה את השורות\חלקים שתרצו למחוק ואת השורות\חלקים שתרצו להוסיף. **הסבירו את בחירותיכם.** בפרט, **הסבירו** באילו בנאים חלופיים יכולתם להשתמש במקום With ומה חשיבות בחירתכם.

```
(: 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)))
```

סעיף ג' (11 נקודות):

נתון לכם כל החלק על הניתוח הסמנטי באינטרפרטר שכתבנו. עליכם לעדכן רק את ההגדרות הפורמליות ואת הפרוצדורה eval, בהתאם למבוא לשאלה ולטסטים מעלה. עליכם רק לציין בצורה ברורה את השורות\חלקים שתרצו למחוק ואת השורות\חלקים שתרצו להוסיף. **הסבירו את בחירותיכם.**

```
;; 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))])

#|      .... עליכם לעדכן חלק זה
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

|#

(: 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)      .... עליכם לעדכן פרוצדורה זאת
;; 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) - this variant is not used
    ;; {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)])))
```

סעיף ד' (6 נקודות):

בהינתן האינטרפרטר המעודכן שכתבתם, השלימו את הטסטים הבאים. הסבירו את תשובותיכם בקצרה.

```
(test (run "{with {f {fun {y} {+ x y}}}
           {with {x 7}
             {call-static f 2}}}")
      <-- fill in -->)

(test (run "{with {f {fun {y} {+ x y}}}
           {with {x 7}
             {call-dynamic f 2}}}")
      <-- fill in -->)
```

סעיף ה' (6 נקודות):

בהינתן האינטרפרטר המעודכן שכתבתם, הראו כיצד על-ידי החלפה אחת ויחידה של `call-static` ל-`call-dynamic` בביטוי הנתון מטה, ניתן לקבל ביטוי שתוצאת הערכתו שונה. הסבירו את תשובתכם.

```
(run "{with {make-adder {fun {y}
                          {fun {x} {+ x y}}}}
      {with {add7 {call-static make-adder 7}}
        {with {add8 {call-static make-adder 8}}
          {* {call-static add7 2}
             {call-static add8 2}}}}}")
```

-----<<<FLANG>>>-----

```
;; The Flang interpreter (substitution model)
```

```
#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> }
```



```
|#
(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)))
#| Evaluation rules:
subst:
  N[v/x] = N
  {+ E1 E2}[v/x] = {+ E1[v/x] E2[v/x]}
  {- E1 E2}[v/x] = {- E1[v/x] E2[v/x]}
  {* E1 E2}[v/x] = {* E1[v/x] E2[v/x]}
  {/ E1 E2}[v/x] = {/ E1[v/x] E2[v/x]}
  y[v/x] = y
  x[v/x] = v
  {with {y E1} E2}[v/x] = {with {y E1[v/x]} E2[v/x]} ; if y /= x
  {with {x E1} E2}[v/x] = {with {x E1[v/x]} E2}
  {call E1 E2}[v/x] = {call E1[v/x] E2[v/x]}
  {fun {y} E}[v/x] = {fun {y} E[v/x]} ; if y /= x
  {fun {x} E}[v/x] = {fun {x} E}

eval:
```

```

eval(N)                = N
eval({+ E1 E2})        = eval(E1) + eval(E2)  \ if both E1 and E2
eval({- E1 E2})        = eval(E1) - eval(E2)  \ evaluate to numbers
eval({* E1 E2})        = eval(E1) * eval(E2)  / otherwise error!
eval({/ E1 E2})        = eval(E1) / eval(E2)  /
eval(id)               = error!
eval({with {x E1} E2}) = eval(E2[eval(E1)/x])
eval(FUN)              = FUN ; assuming FUN is a function expression
eval({call E1 E2})     = eval(Ef[eval(E2)/x]) if eval(E1)={fun {x} Ef}
                      = error!                otherwise

|#
(: 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)))]))

(: 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 -> 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)])))]

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

--<<<FLANG-SubstCache>>>-----
;; The Flang interpreter, using substitution cache model.
#lang pl --- only evaluation part ---

#| Evaluation rules:
eval(N,sc)                = N
eval({+ E1 E2},sc)        = eval(E1,sc) + eval(E2,sc)
eval({- E1 E2},sc)        = eval(E1,sc) - eval(E2,sc)
eval({* E1 E2},sc)        = eval(E1,sc) * eval(E2,sc)
eval({/ E1 E2},sc)        = eval(E1,sc) / eval(E2,sc)
eval(x,sc)                 = lookup(x,sc)
eval({with {x E1} E2},sc) = eval(E2,extend(x,eval(E1,sc),sc))
eval({fun {x} E},sc)       = {fun {x} E}
eval({call E1 E2},sc)
  = eval(Ef,extend(x,eval(E2,sc),sc))
                        if eval(E1,sc) = {fun {x} Ef}
  = error!              Otherwise      |#

(: 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)])))
```