פקולטה: מדעי הטבע

מחלקה: מדעי המחשב ומתמטיקה

שם הקורס: שפות תכנות

קוד הקורס: 2-7036010-1/2

תאריך בחינה: \_05/06/2014 סמ' \_ב'\_\_\_מועד \_\_א'\_\_

משך הבחינה: 3 שעות

שם המרצה: ערן עמרי

חומר עזר: אסור

שימוש במחשבון: לא

הוראות כלליות:

* כתבו את תשובותיכם בכתב קריא ומרווח.
* בכל שאלה או סעיף, ניתן לכתוב – **לא יודע/ת** (מבלי להוסיף דבר מעבר לכך) – ולקבל 20% מהניקוד על השאלה או הסעיף.
* אפשר להסתמך על סעיפים קודמים גם אם לא עניתם עליהם.
* יש לענות על כל השאלות.
* ניתן להשיג עד 107 נקודות במבחן.

שאלה 1 — BNF — (28 נקודות):

סעיף א' (7 נקודות):

הסבירו את מושג הרב משמעיות (ambiguity) עבור BNF. תנו דוגמה לדקדוק (כלשהו) אשר אינו חד-משמעי (כלומר, הוא סובל מ-ambiguity). הדגימו מדוע הדקדוק שלכם אינו חד-משמעי.

סעיף ב' (8 נקודות):

רוצים להוסיף לשפת הביטויים האריתמטיים הבסיסית **AE** את האפשרות להשתמש בזכרון – בעזרת פעולות:

set – הכנסת תוצאת חישוב לזכרון.

get – שליפת הערך מהזכרון (בכל שלב נשמר ערך יחיד בזכרון).

בשאלה זו נטפל רק בכתיבת BNF עבור שפה זו (נקרא לשפה זו ‘**MAE**’).

פתרון נאיבי מציע את הדקדוק הבא:

|  |
| --- |
| <MAE> ::= <num>  | { + <MAE> <MAE> } ; Rule 1  | { - <MAE> <MAE> } ; Rule 2  | { \* <MAE> <MAE> } ; Rule 3  | { / <MAE> <MAE> } ; Rule 4  | { set <MAE> } ; Rule 5  | get ; Rule 6 |

כאן הכוונה בביטוי **{set E}** (באשר, **E** הינו ביטוי כלשהו) הינה לחשב את הערך של **E** ולהכניס ערך זה לזכרון.

הביטוי הבא מדגים בעיה בדקדוק המוצע מעלה:

|  |
| --- |
| {\* {+ {set 1} {set 2}} get} |

* הראו כיצד נגזר הביטוי מהדקדוק המוצע.
* הסבירו מה הבעיה שמודגמת בביטוי זה.

סעיף ג' (10 נקודות):

בכדי לפתור את הבעיה הקודמת ובכדי לתת לביטויים בשפה תאימות גדולה יותר לאופן שבו אנחנו משתמשים בזכרון במחשבון, הגדירו דקדוק **MAE** המקיים את התנאים הבאים:

* ביטוי בשפה הינו **סדרה**, לא ריקה, של תתי ביטויים המתארים חישוב. הביטוי כולו מתחיל בסימן הפעולה **seq** ועטוף בסוגריים מסולסלים.
* כל אחד מתתי הביטויים הללו (מלבד האחרון בסדרה) מתחיל בסימן הפעולה **set**, יש לו אופרנד יחיד והוא עטוף בסוגריים מסולסלים. האחרון בסדרה נראה כמו אופרנד (כמתואר מטה). הראשון אינו כולל את סימן הפעולה **get** .
* **אופרנד** של תת-ביטוי הינו אחת משלוש אפשרויות:
  + **מספר**
  + ביטוי אריתמטי עטוף בסוגריים מסולסלים עם אחד מארבעת סימני הפעולה **+**, -, /, \* ושני אופרנדים.
  + סימן הפעולה **get**. שימו לב! אסור ל-get להופיע בביטוי הראשון בסדרה כולה.

דוגמאות:

|  |
| --- |
| ;; **valid sequences**  {seq {set {+ 8 7}}  {set {\* get get}}  {/ get 2}}  {seq {- 8 2}}  {seq 25}  ;; **invalid sequences**  {seq {set {\* 8 get}} ; cannot begin with a `get'  24}  {seq {\* 8 7} ; must be a `set'  24}  {seq {set {+ 1 2}}  {set {- get 2}}} ; cannot end with a `set'  {seq {\* 2 {set {+ 1 2}}} ; `set' must be outside  {- get 2}} |

סעיף ד' (3 נקודות):

הראו כיצד ניתן לגזור ביטוי בשפה של הדקדוק שהגדרתם. בביטוי זה (שעליכם להמציא) יופיעו לפחות שלושה מופעים של **set**, לפחות שלושה מופעים של **get** ולפחות שלוש פעולות חשבון.

שאלה 2 — המימוש של ה-interpreter במודל הסביבות ובמודל ההחלפות — (15 נקודות):

לצורך פתרון שאלה זו מצורפים שני קטעי קוד עבור ה- interpreter של FLANG בסוף טופס המבחן. הראשון – במודל ה-substitution והשני במודל הסביבות.

לפניכם מספר שאלות פשוטות. ענו תשובות קצרות (שלוש שורות לכל היותר).

סעיף א' (5 נקודות):

מה מייצג הטיפוס FLANG? באיזה חלק של תהליך האינטרפרטציה הוא משמש אותנו (מה השתנה בנושא זה בין שני המימושים של האינטרפרטר -- הראשון במודל ה-substitution והשני במודל הסביבות) .

סעיף ב' (5 נקודות):

תהליך האינטרפרטציה בשני המודלים שונה בפונקציה eval בטיפול בבנאי Id. הסבירו מה מתבצע במקרה זה בכל אחד מהמימושים ומדוע.

סעיף ב' (5 נקודות):

מהו ההבדל המרכזי בתהליך האינטרפרטציה בין שני המודלים? מה הייתה המוטיבציה מאחורי שינוי זה?

שאלה 3 — FLANG — (35 נקודות):

לצורך פתרון שאלה זו מצורפים שני קטעי קוד עבור ה- interpreter של FLANG בסוף טופס המבחן. הראשון – במודל ה-substitution והשני במודל הסביבות.

נתון הקוד הבא:

**(run "{with {Mul-x {fun {x} {\* x z}}}**

**{with {z 3}**

**{call Mul-x z}}}")**

סעיף א' (6 נקודות):

ציירו את עץ התחביר האבסטרקטי המתאר את הביטוי הנתון במרכאות (כלומר את התוצאה של הפעלת parse על ביטוי זה).

דוגמא 1: העץ המתאר את הביטוי **{+ 1 2}** הוא:

תאור חלופי עבור עץ זה הוא:

**(Add (Num 1) (Num 2))**

סעיף ב' (12 נקודות):

בתהליך ההערכה של ביטוי זה במודל ה-substitution (על-פי ה-interpreter העליון מבין השניים המצורפים מטה) תופעל הפונקציה eval 10 פעמים. תארו את 10 ההפעלות הללו על-פי סדר הופעתן בחישוב. לכל הפעלה תארו את הפרמטר האקטואלי עליו מופעלת הפונקציה eval וכן את הערך המוחזר מהפעלה זו. הסבירו בקצרה כל מעבר.

דוגמא 2:

בתהליך החישוב של העץ מדוגמא 1, יתבצעו ההפעלות הבאות של eval (מימין מופיעות תוצאות החישוב).

1. (eval (Add (Num 1) (Num 2))) => (Num 3)
2. (eval (Num 1)) => (Num 1)
3. (eval (Num 1)) => (Num 2)

כדי להציג את תשובתכם באופן נוח, כתבו מהם עבור , באשר מייצג את הפרמטר הפורמלי בקריאה ה-*i* ל-eval ו- מייצג את הערך המוחזר מהפעלה זו.

סעיף ג' (12 נקודות):

בתהליך ההערכה של ביטוי זה במודל ה-environment (על-פי ה-interpreter התחתון מבין השניים המצורפים מטה) תופעל הפונקציה eval 10 פעמים. תארו את 10 ההפעלות הללו על-פי סדר הופעתן בחישוב. לכל הפעלה תארו את שני הפרמטרים האקטואלים עליהם מופעלת הפונקציה eval וכן את הערך המוחזר מהפעלה זו. הסבירו בקצרה כל מעבר.

דוגמא 3:

בתהליך החישוב של העץ מדוגמא 1, יתבצעו ההפעלות הבאות של eval (מימין מופיעות תוצאות החישוב).

1. (eval (Add (Num 1) (Num 2))

(EmptyEnv)) => (NumV 3)

1. (eval (Num 1)

(EmptyEnv)) => (NumV 1)

1. (eval (Num 2)

(EmptyEnv)) => (NumV 2)

סעיף ד' (5 נקודות):

הסבירו ממה נובע ההבדל בתוצאת החישוב בשני בתהליכי ההערכה של הביטוי זה במודל ה-substitution ובמודל ה-environment . מהי התוצאה הרצויה מבחינתנו? הסבירו.

שאלה 4 — הרחבת השפה — (29 נקודות):

לצורך פתרון שאלה זו שוב נעזר בקוד ה- interpreter של FLANG במודל הsubstitution , המופיע בסוף טופס המבחן (העליון מבין השניים המופיעים שם).

נרצה להרחיב את השפה ולאפשר מציאת סכום הריבועים מ-0 ועד הפרמטר n (כולל).

להלן דוגמאות לטסטים שאמורים לעבוד:

**(test (run "{sumsqrbelow 2}") => 5)**

**(test (run "{+ 2 {sumsqrbelow {+ 3 2}}}") => 57)**

**(test (run "{call {fun {x}**

**{+ {sumsqrbelow x} 1}}**

**4}") => 31)**

סעיף א' (10 נקודות):

ראשית נגדיר פונקציה בשפה **pl** (הגרסה של **Racket** בה אנו משתמשים). שורת ההכרזה על הפונקציה תהיה

**(: sum-square-below : Number -> Number)**

ניתן להניח (ואין צורך בבדיקת נכונות) שהקלט x הוא מספר טבעי (יכול להיות 0). הפונקציה תחשב את סכום **הריבועים** של הטבעיים 0…x . כתבו את הפונקציה כך שכל הקריאות הרקורסיביות הן קריאות זנב.

להלן דוגמאות לטסטים שאמורים לעבוד:

**(test (sum-square-below 1) => 1)**

**(test (sum-square-below 2) => 5)**

**(test (sum-square-below 3) => 14)**

סעיף ב' (2 נקודות):

הרחיבו את הדקדוק בהתאם (הוסיפו את הקוד הנדרש היכן שכתוב **—«fill-in»—**):

**#| The grammar:**

**<FLANG> ::= <num>**

**| { + <FLANG> <FLANG> }**

**| { - <FLANG> <FLANG> }**

**| { \* <FLANG> <FLANG> }**

**| { / <FLANG> <FLANG> }**

**| { with { <id> <FLANG> } <FLANG> }**

**| <id>**

**| { fun { <id> } <FLANG> }**

**| { call <FLANG> <FLANG> }**

**| { —«fill-in»—} ; Add**

**|#**

סעיף ג' (2 נקודות):

הוסיפו את הקוד הנדרש (היכן שכתוב **—«fill-in»—**) ל –

**(define-type FLANG**

**... ראו קוד ה-interpreter מטה ...**

**[Ssb —«fill-in»— ]) ; Add**

סעיף ד' (3 נקודות):

הוסיפו את הקוד הנדרש (בתוך הסוגריים המרובעים) ל –

**(: parse-sexpr : Sexpr -> FLANG)**

**;; to convert s-expressions into FLANGs**

**(define (parse-sexpr sexpr)**

**(match sexpr**

**... ראו קוד ה-interpreter מטה ...**

**[—«fill-in»—] ; Add**

**[else (error 'parse-sexpr "bad syntax in ~s" sexpr)]))**

סעיף ה' (4 נקודות):

הוסיפו את הקוד הנדרש (בתוך הסוגריים המרובעים) ל –

**(: 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**

**... ראו קוד ה-interpreter מטה ...**

**[—«fill-in»—])) ; Add**

סעיף ו' (8 נקודות):

עתה נרצה לאפשר לפונקציה eval לטפל במקרה שנוסף עבור פעולת sumsqrbelow.

**הערה:** אינכם צריכים לטפל במקרה שבו ערך הביטוי עליו מופעלת sumsqrbelow הינו מספר לא טבעי. אולם, עליכם עדיין לוודא שאינו פונקציה.

לצורך כך נגדיר פונקצית עזר: **הוסיפו את הקוד הנדרש במקום המתאים**

**(: ssb-op : FLANG -> FLANG)**

**(define (ssb-op expr)**

**(: Num->number : FLANG -> Number)**

**(define (Num->number e)**

**(cases e**

**[(Num n) n]**

**[else (error 'ssb-op "expects a number, got: ~s" e)]))**

**(Num —«fill-in»—)) ; Add**

**הוסיפו את הקוד הנדרש (בתוך הסוגריים המרובעים) ל –**

**(: eval : FLANG -> FLANG)**

**;; evaluates FLANG expressions by reducing them to \*expressions\***

**(define (eval expr)**

**(cases expr**

**... ראו קוד ה-interpreter מטה ...**

**[—«fill-in»—])) ; Add**

**הדרכה:** השתמשו בפונקציה שלכם מסעיף א'.

**---<<<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> }**

**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**

**|#**

**(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)))**

**(: 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)])))**

**;; tests**

**(test (run "{call {fun {x} {+ x 1}} 4}")**

**=> 5)**

**(test (run "{with {add3 {fun {x} {+ x 3}}}**

**{call add3 1}}")**

**=> 4)**

**--<<<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)])))**

**;; tests**

**(test (run "{call {fun {x} {+ x 1}} 4}")**

**=> 5)**

**(test (run "{with {add3 {fun {x} {+ x 3}}}**

**{call add3 1}}")**

**=> 4)**

**(test (run "{with {add3 {fun {x} {+ x 3}}}**

**{with {add1 {fun {x} {+ x 1}}}**

**{with {x 3}**

**{call add1 {call add3 x}}}}}")**

**=> 7)**

**(test (run "{with {identity {fun {x} x}}**

**{with {foo {fun {x} {+ x 1}}}**

**{call {call identity foo} 123}}}")**

**=> 124)**

**(test (run "{with {x 3}**

**{with {f {fun {y} {+ x y}}}**

**{with {x 5}**

**{call f 4}}}}")**

**=> 7) ;; the example we considered for subst-caches**

**(test (run "{call {with {x 3}**

**{fun {y} {+ x y}}}**

**4}")**

**=> 7)**

**(test (run "{call {call {fun {x} {call x 1}}**

**{fun {x} {fun {y} {+ x y}}}}**

**123}")**

**=> 124)**